Chapter 10
Decision Tree Learning

Abstract The main objective of this chapter is to introduce you to hierarchical supervised learning models. One of the main hierarchical models is the decision tree. It has two categories: classification tree and regression tree. The theory and applications of these decision trees are explained in this chapter. These techniques require tree split algorithms to build the decision trees and require quantitative measures to build an efficient tree via training. Hence, the chapter dedicates some discussion to the measures like entropy, cross-entropy, Gini impurity, and information gain. It also discusses the training algorithms suitable for classification tree and regression tree models. Simple examples and visual aids explain the difficult concepts so that readers can easily grasp the theory and applications of decision tree.

10.1 The Decision Tree

In practice, the decision tree-based supervised learning is defined as a rule-based, binary-tree building technique (see [1–3]), but it is easier to understand if it is interpreted as a hierarchical domain division technique. Therefore, in this book, the decision tree is defined as a supervised learning model that hierarchically maps a data domain onto a response set. It divides a data domain (node) recursively into two subdomains such that the subdomains have a higher information gain than the node that was split. We know the goal of supervised learning is the classification of the data, and therefore, the information gain means the ease of classification in the subdomains created by a split. Finding the best split that gives the maximum information gain (i.e., the ease of classification) is the goal of the optimization algorithm in the decision tree-based supervised learning.

Suppose we have a system that produces events (observations) that can be one of the classes 0 or 1 (e.g., rain or no rain, head or tail), and these events depend on only one feature. Hence, let us define the data domain as $D = \{e_1, e_2, \ldots, e_n\}$ (assume

S. Suthaharan, *Machine Learning Models and Algorithms for Big Data Classification*, Integrated Series in Information Systems 36,
DOI 10.1007/978-1-4899-7641-3_10

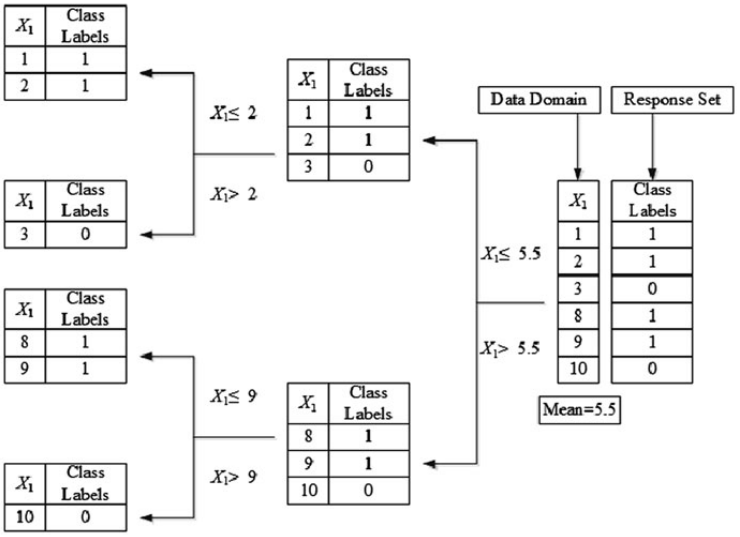

Fig. 10.1 Classification example. A decision tree building with a one-dimensional data domain—output is a discrete value

this is a sorted list), and their corresponding class labels are $L = \{r_1, r_2, \ldots, r_n\}$, where $r_i \in \{0, 1\}$, and $i = 1 \ldots n$. The spread (or the distribution pattern) of the class labels over the data domain determines the ease of classification. Let us represent the information gain of D with respect to L by I_i and split the sorted set at the location m to form two subdomains $D_1 = \{e_1, e_2, \ldots, e_m\}$ and $D_2 = \{e_{m+1}, e_2, \ldots, e_n\}$ with the corresponding response sets $L_1 = \{r_1, r_2, \ldots, r_m\}$ and $L_2 = \{r_{m+1}, r_2, \ldots, r_n\}$. If their information gains are I_{i1} and I_{i2}, then m will be considered as the best split if $average(I_{i1}, I_{i2}) > I_i$. Of course, we need a good quantitative measure to measure the information gain, or the ease of classification, with respect to the domain split.

Suppose p_0 and p_1 represent the probabilities that class 0 and class 1 can be drawn from the domain D, respectively. Take an example that $|p_0 - p_1| \to 1$; then we can see one particular class dominates highly in that domain, hence further domain division is not required. Similarly if $|p_0 - p_1| \to 0$, then the classes have equal domination in that domain; therefore, further split is needed. In that case, we generate two subdomains D_1 and D_2. Say, q_0 and q_1 are the probabilities that class 0 and class 1 can be drawn from the subdomain D_1, respectively. If the split is efficient, $q_0 > p_0$ or $q_1 > p_1$. Assume $q_0 > p_0$, then $q_0 = p_0 + \varepsilon$, where $\varepsilon > 0$.

$$|q_0 - q_1| = |2q_0 - 1| = |2(p_0 + \varepsilon) - 1| = |2p_0 + 2\varepsilon - 1| \tag{10.1}$$

$$|q_0 - q_1| = |p_0 + 1 - p_1 + 2\varepsilon - 1| = |p_0 - p_1 + 2\varepsilon|. \tag{10.2}$$

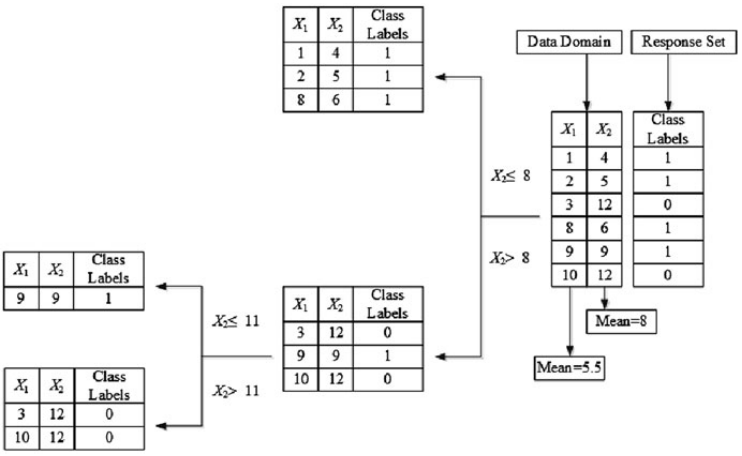

Fig. 10.2 Classification example. A decision tree building with a two-dimensional data domain—output is a discrete value

This mathematical equation emphasizes the following inequality (when $q_0 > p_0$):

$$|q_0 - q_1| > |p_0 - p_1|. \tag{10.3}$$

The absolute differences in the above inequality are the quantitative measures that measures the proportionality between the classes in the respective subdomains. This probabilistic measure is a good measure to address the optimization objectives of the decision tree. Let us look at some "Thinking with Examples" and understand the decision tree better in terms of domain division focusing on information gain.

Thinking with Example 10.1:

The purpose of this example is to show you how a data domain formed by a single feature may be divided and mapped to a two-class (discrete) response set. Suppose the data domain is a single feature set $X_1 = \{1,2,3,8,9,10\}$ with a set of assigned class labels $L = \{1,1,0,1,1,0\}$, respectively. Then the root node of the decision tree is the feature X_1, and its value is used to divide the data domain as shown in Fig. 10.1. We now need a parameter and an approach to divide this root node to build a decision tree. For simplicity, we can choose the mean value ($m = 5.5$) as the parameter value, and the approach as the values $\leq m$ form the left subdomain and the values $> m$ form the right subdomain. We can see these subdomains at the second level of the tree (assuming the root is the first level). The mean values of

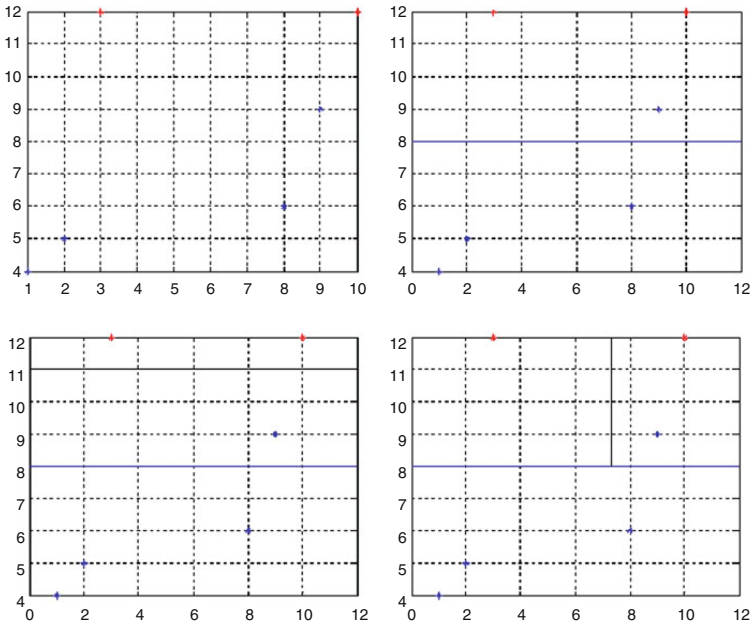

Fig. 10.3 Results from the code in Listing 10.1

the subdomains ($m = 2$ and $m = 9$) can be used to expand the tree as shown in Fig. 10.1. Finally, the leaves that show the class labels are determined by the mean as the split criterion and feature X_1 as the domain and subdomains variables. Let us now consider a two-dimensional data domain example.

Thinking with Example 10.2:

Suppose the system adds another feature set $\{4, 5, 12, 6, 9, 12\}$ to the same data with the same labeling order. Hence, it creates the two-dimensional data domain formed by the features X_1 and X_2. We now have two features, thus we must adopt an approach to select one feature for the root node first. For simplicity, assume that the feature X_2 is selected randomly. Then the data domain considered at the root of the tree is $\{4, 5, 12, 6, 9, 12\}$, as shown in Fig. 10.2. Once again, if we use the mean as the split criterion with the same splitting approach, then we get the subdomains as shown in the figure. The left node at this level has the same class (i.e., class 1), so no further split is required. The right node needs a split, and now we have a choice

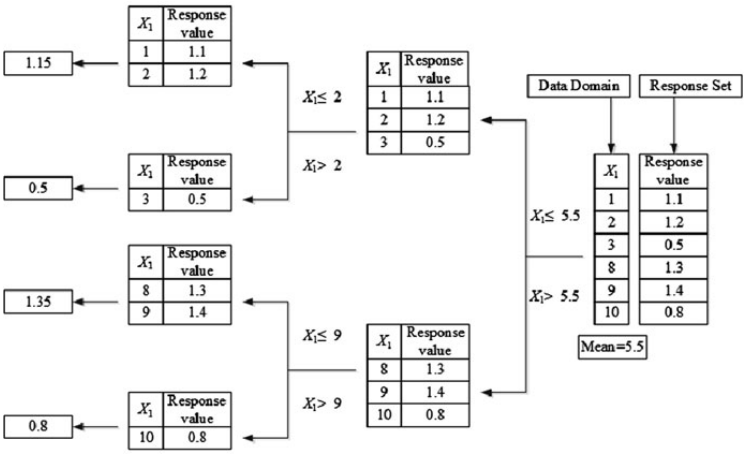

Fig. 10.4 Regression example. A decision tree building with a one-dimensional data domain—output is a real number

to have either X_1 or X_2. The tree shows the selection of the same feature. The mean is calculated, and the node is split. Now the leaves have the class labels from the same classes. The code in Listing 10.1 provides the results of the domain division example in Fig. 10.2.

10.1.1 A Coding Example—Classification Tree

The block of code in line 4 to line 14, a two-dimensional data set with two classes is created, a data domain and a response set are established, and these data points are plotted as shown in the first figure in Fig. 10.3. In lines 18–20, the data domain is divided into two subdomains using the mean value of feature 2 as the split location. This statistical mean value is a part of the classifier. Note that this is the place where the decision tree calculates the information gain using Gini index to find the optimal split location. The block of code in lines 22–23 creates the subdomains (or the children of the tree node). The classifier mean at this node is plotted in the second figure of Fig. 10.3.

Listing 10.1 A Matlab example—classification tree

```
1  clear all;
2  close all;
3
4  x1=[1 2 3 8 9 10];
```

```
5  x2=[4 5 12 6 9 12];
6  yy=[1 1 0 1 1 0];
7
8  xx=[x1' x2' yy'];
9
10 ind1=find(xx(:,3)==1);
11 ind2=find(xx(:,3)==0);
12
13 figure;plot(xx(ind1,1),xx(ind1,2),'b*');grid on;
14 hold on;plot(xx(ind2,1),xx(ind2,2),'r*');grid on;
15
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17
18 m1=mean(xx(:,2));
19 indL=find(xx(:,2)<m1);
20 indR=find(xx(:,2)>=m1);
21
22 xxL=xx(indL,:)
23 xxR=xx(indR,:)
24
25 hold on;line([0, max(xx(:,2))],[m1,m1]);
26
27 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
28
29 ind=2;
30 m2=mean(xxR(:,ind));
31 indRL=find(xxR(:,ind)<m2);
32 indRR=find(xxR(:,ind)>=m2);
33
34 xxRL=xxR(indRL,:)
35 xxRR=xxR(indRR,:)
36
37 if (ind==2)
38     hold on;line([0, max(xxR(:,ind))],[m2,m2],'color','k');
39 else
40     hold on;line([m2,m2],[8, 12],'color','k');
41 end
```

This is a toy example; therefore, we can see the intermediate output and observe that the left domain does not need the split. Only the right subdomain is further divided using the same feature (i.e., feature 2) in lines 29–32. By changing the value of the variable ind in line 29 to 1, we can perform the split based on feature 1. The third and fourth figures show the second split using the statistical means of features 1 and 2, respectively. The block of code in lines 37–41 performs this task. The numerical output of this program is (if ind = 2):

```
xxL =                          xxR =
     1     4     1                3    12     0
     2     5     1                9     9     1
     8     6     1               10    12     0
```

```
xxRL =                          xxRR =
      9      9      1                3     12      0
                                     10     12      0
```

These results are presented in a two-column format only for the purpose of improving the presentation of the results. Similarly, the output of this program is (if ind = 1):

```
xxL =                           xxR =
      1      4      1                3     12      0
      2      5      1                9      9      1
      8      6      1               10     12      0


xxRL =                          xxRR =
      3     12      0                9      9      1
                                     10     12      0
```

These outputs are also presented in a two-column format for the purpose of improving the presentation. This example leads us to ask the following questions: (1) Which features must be selected for the nodes? (2) How do we parametrize the node split? (3) What parameter values must be chosen? (4) How do we parametrize the depth of the tree? and (5) How do we optimize the tree structure?

We can simply state: in decision tree supervised learning, there is a data domain which must be split into two subdomains at a location on a feature set with the focus of obtaining an information gain at each split. This process must be recursively carried out until the data domain is divided into several subdomains, where each domain presents an optimal classification.

Thinking with Example 10.3:

The purpose of this example is to show you how a data domain formed by a single feature may be divided and mapped to a continuous response set. Suppose the data domain is a single feature set $X_1 = \{1, 2, 3, 8, 9, 10\}$ with a set of assigned continuous responses $R = \{1.1, 1.2, 0.5, 1.3, 1.4, 0.8\}$. We use the same decision tree building as shown in Fig. 10.1, and it is presented in Fig. 10.4 with minor differences. The responses are averaged over the values of the subdomains resulting in at the leaves of the tree. This is called the regression tree, and we will study this in detail later in this chapter. The code in Listing 10.2 provides the results of the domain division for a regression tree.

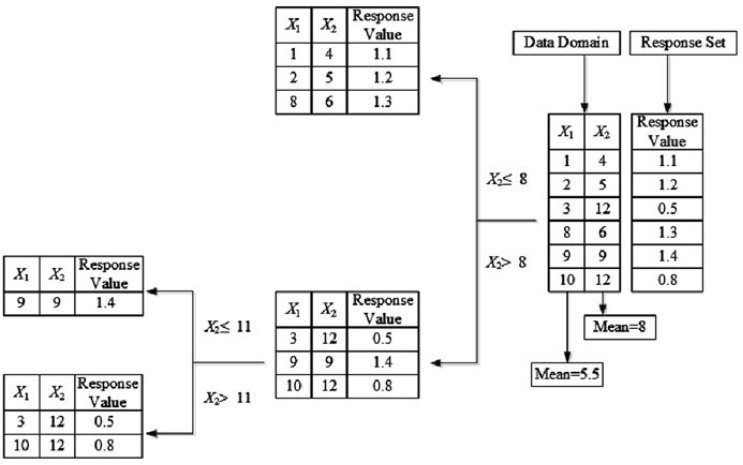

Fig. 10.5 Regression example. A decision tree building with a two-dimensional data domain—output is a real number—result of Listing 10.2

10.1.2 A Coding Example—Regression Tree

The program in this section can be explained by the diagram in Fig. 10.5. The block of code in lines 4–6 defines the two-dimensional data domain and the response set with real numbers for regression. Line 8 defines a single matrix which contains both data domain and response set.

Listing 10.2 A Matlab example—regression tree

```
1  clear all;
2  close all;
3
4  x1=[1 2 3 8 9 10];
5  x2=[4 5 12 6 9 12];
6  yy=[1.1 1.2 0.5 1.3 1.4 0.8];
7
8  xx=[x1' x2' yy'];
9
10 ind1=find(xx(:,3)==1);
11 ind2=find(xx(:,3)==0);
12
13 figure;plot3(xx(:,1),xx(:,2),xx(:,3),'*');grid on;
14
15 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16
17 m1=mean(xx(:,2));
18 indL=find(xx(:,2)<m1);
```

```
19 indR=find(xx(:,2)>=m1);
20
21 xxL=xx(indL,:);
22 xxR=xx(indR,:);
23
24 mean(xxL(:,3))
25
26 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
27
28 ind=2;
29 m2=mean(xxR(:,ind));
30 indRL=find(xxR(:,ind)<m2);
31 indRR=find(xxR(:,ind)>=m2);
32
33 xxRL=xxR(indRL,:);
34 xxRR=xxR(indRR,:);
35
36 mean(xxRL(:,3))
37 mean(xxRR(:,3))
```

The block of code from line 17 to 24 performs the first split using feature 2 and prints the average of the responses of the subdomain in the left side of the tree. The block of code in lines 28–34 splits the subdomain in the right side of the tree further using feature 2. The averages of the responses in the new branches of the tree are printed in lines 36–37. Hence, the final output of this program is:

```
ans =
    1.2000

ans =
    1.4000

ans =
    0.6500
```

This program is hardcoded; therefore, you can modify the program in appropriate places to investigate the effect of the features at different tree nodes. For example, modify the code in line 28 to ind = 1, and study the effect.

10.2 Types of Decision Trees

In supervised learning, the decision tree has been divided into classification trees and regression trees by Breiman et al. [4]. In simple terms, we can say the classification tree helps predict a class label (i.e., discrete) for a response variable whereas the regression tree helps predict a value (i.e., continuous) for the response variable. In Figs. 10.6 and 10.7, the processes of a classification tree and regression tree are illustrated using a hierarchical structure to show the evolution of domain division properties.

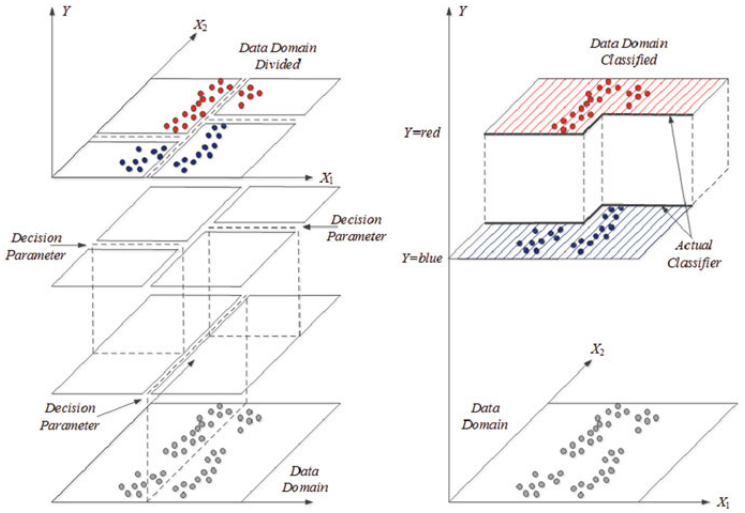

Fig. 10.6 The classification tree is illustrated in 3D using two classes with domain division properties. Response variable Y has two discrete values, *red* or *blue*

10.2.1 Classification Tree

The classification tree helps assign a label to a new data set. For example, it can help us decide if a new observation belongs to a class 1 or a class 0. The concept of the classification tree is explained using the illustration in Fig. 10.6, where two classes (red and blue) are used. Note that the explanation is the construction of a classification tree at the training phase, and it gives a visual example of a decision tree building. It also uses the data domain and its changes during the construction of a decision tree. The left diagram shows the propagation of tree split that divides the data domain and creates subdomains for appropriate classification based on the given class labels. It can be seen as the generation of simple multiple thin layers of data domains with split-regions. The first thin layer shows the given data domain and a split condition which is applied to the first feature. The domain is divided into two subdomains, and it is shown in the second thin layer. These subdomains are further divided into four subdomains based on the second feature, and they are shown in the third thin layer. Finally, the class labels are highlighted and we can see they are classified into different disjointed subdomains.

The diagram on the right side shows the given data domain and the final classified data domain. It illustrates the effect on the response variable (discrete) with two classes, where the related subdomains are combined to show the class separation. The main parameters of the classification tree models are the values of the features

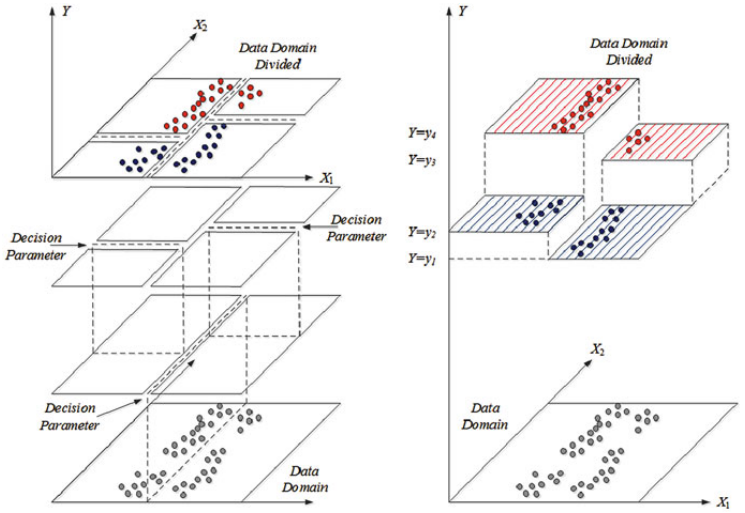

Fig. 10.7 The regression tree is illustrated in 3D using two classes with domain division properties. Response variable Y has two continuous *red* and *blue* values

used to split the tree at a particular node, and these parameters are trained using information gain as a quantitative measure. The combination of parameters selected by the classification tree at training is considered the classifier. The decision parameters and the actual classification with the tree-based classifier are explained later in the chapter.

In classification trees, the split criterion must give an information gain so that the subdomains (i.e., the leaves of the current tree) have class information that help separate the classes. The split criterion needs decision parameters that form the classification model. The decision parameters (a feature and its split location) are calculated based on the maximization of the information gain. That is, the decision parameters shown in Fig. 10.6 can be calculated according to this principle.

10.2.2 Regression Tree

The regression tree helps assign a value for new data. The objective of the regression tree model is to divide the data domain into disjoint, rectangular subdomains by splitting features, and then map the subdomains to nonoverlapping groups of responses that are estimated with the minimum error criterion (e.g., least square

method). See Fig. 10.7. This statement shows four tasks: (1) feature selection for a node, (2) parametrization of the split location, (3) parametrization of the depth of the tree, and (4) estimation of the response variables.

The PhD thesis by Torgo [5] provides a very good explanation, and I encourage readers refer to it for additional information. The full version of the thesis can be found at the following website: http://www.dcc.fc.up.pt/~ltorgo/PhD/. The popular book by Breiman et al. [4] is another best resource for exploring regression trees in depth. The decision parameters are chosen based on the least square criterion (see Leo Breiman's classification and regression trees book [4] and Torgo's thesis [5]). The example in the figure shows the decision parameters that are calculated based on the mean value of the respond variable for each subdomain and the domain.

10.3 Decision Tree Learning Model

We have seen in a previous chapter that modeling means the definition of a function or a mapping between a data domain and a response set followed by the parametrization of the model and the optimization of the parameters. It is clear that the decision tree satisfies this definition and forms a supervised learning model because it can be trained, validated, and tested using the supervised learning algorithms. Let us now study the processes of parametrization and optimization of the decision tree supervised learning model.

10.3.1 Parametrization

As discussed before, a supervised model must be parametrized so that it can be trained, validated, and tested. This is true for a decision tree as well. The question now is how to parametrize a tree-like structure and find the parameters. In building a tree-like structure, the nodes are split and leaves are generated, and this process is recursively done. Therefore, we can parametrize (1) the choice of features for a node, (2) the threshold that splits the node and divides the feature set into two subsets, and (3) the number of levels that the entire tree should have. It is illustrated in Figs. 10.8 and 10.9.

- Parametrization: select the features for the root node and intermediate nodes of the decision tree. Hence, I would say the feature must be considered as one of the parameters for the decision tree. We have two options: (1) We may select a feature for a node randomly, or (2) We may select a feature that can give some information gain (or error reduction) when it is used and subdomains are constructed.

- Parametrization: select a parameter to split criterion (e.g., split location in the feature set). Hence, a threshold for the feature values can form a parameter for the decision tree. We have two options: (1) We may select the statistical tech-

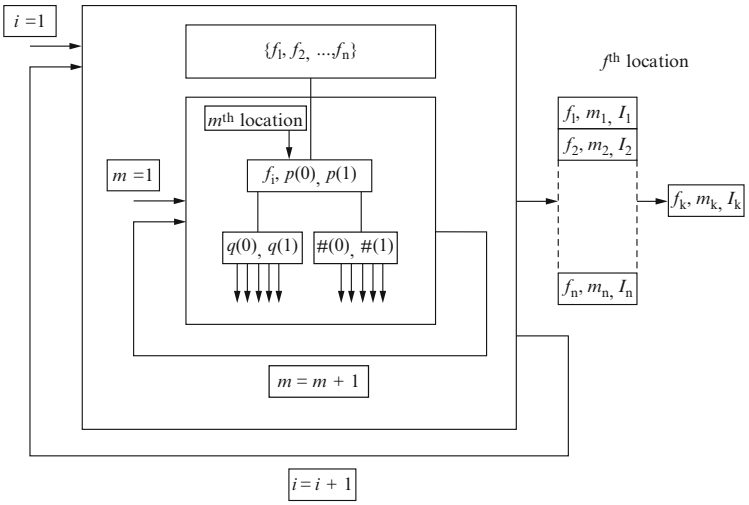

Fig. 10.8 The process of selecting the best features (with its split location) that give the maximum information gain ($i = 1, \ldots, n$)

niques like mean or median to find the split location, or (2) We can find the split location which will give two subdomains that lead to an acceptable information gain (or error reduction) by splitting.

- Parametrization: select a parameter to stop the tree building. It means that the number of levels in the decision tree must be a parameter.

- Optimization: select an algorithm that helps optimize the parameters such that the final decision tree is optimal so that the tree can perform a very good prediction of class labels. This may lead to a computationally expensive process, because all of the possible combinations of features and the split locations must be processed toward obtaining information gain values and selecting the feature and split location combination that gives maximum gain or minimum error.

10.3.2 Optimization

The example in Fig. 10.1 shows the dividing of a tree node (i.e., building a model) and building a decision tree, but it doesn't show how to divide a tree node *efficiently* (i.e., the training). To find an answer to this question, we may need to ask several other questions: Which feature must be selected first to start building the tree? Which features must be selected at the intermediate steps of the tree building? It

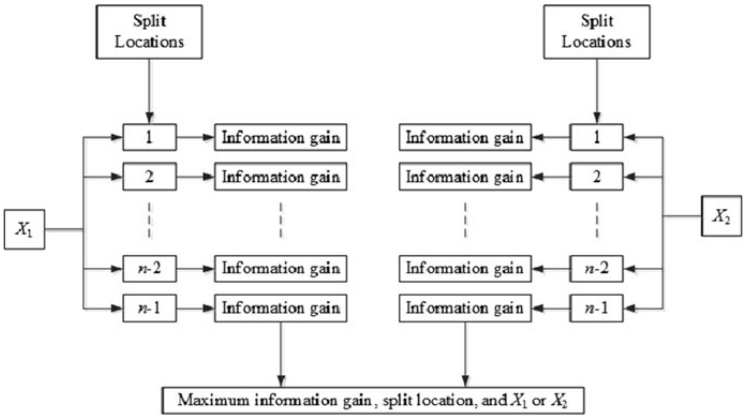

Fig. 10.9 The process of finding the best split location using the maximum information gain—a sub process for Fig. 10.8

means that we need a good quantitative measure which is applicable to tree building structure. Tree building means the generation of leaves (or branches) from a node, and carrying out this process iteratively. Let me explain the standard quantitative measures like entropy [6], Gini impurity [3, 7], information gain [3], and cross-entropy [8].

10.4 Quantitative Measures

In decision tree modeling, the quantitative measures are required in two places: (1) to measure the information gain resulted by a feature split over data domains (or subdomains), and (2) to measure the significance in the class difference at each node to decide further split. Useful measures for the first requirement are the entropy, Gini impurity, and information gain. Useful measures for the second requirement are the class proportions, count differences, and a probability measure (e.g., ratios and percentages).

10.4.1 Entropy and Cross-Entropy

Entropy provides a measure based on the proportionality of the events. For example, if one event occurs more than another event in a place or with an object, then we have good knowledge about that place or the object relative to the majority event. If both events occur the same number of times in that place or with the object, then it is hard to characterize the place or the object. Say, for example, if a football team

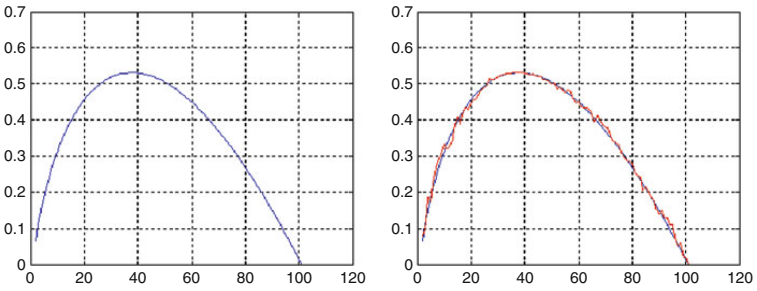

Fig. 10.10 The entropy characteristics with and without an error factor, respectively—horizontal axis represents the probability index and the vertical axis represents the entropy

has 9 wins and 1 loss, then we should be able to characterize the team as a "hard to beat team." Similarly, the rainfall recorded over 365 days in a place shows 360 days of rain, then we may characterize the place possibly as a "rainforest." This is the characteristic of the entropy measure. The entropy is defined as follows [6]:

$$\text{en} = -\sum_i p_i \log_2(p_i), \tag{10.4}$$

where p_i is the probability of the ith event. To understand better, take four different examples. In all examples, the assumption is that two events are occurring in an environment.

Thinking with Example 10.4:

Say, for example, "rain" or "no rain." In this first example, we take the probability for the first event is 1 (i.e., $p_1 = 1$) and the second event is 0 (i.e., $p_1 = 0$), and let us denote the probabilities of the first and the second events by $[1,0]$ matrix. Then the entropy is: $\text{en} = -1 \times \log(1) - 0 \times \log(0) = -1 \times 0 - 0 \times \infty = 0 - 0 = 0$. This result indicates that there is no-error, and it means we can characterize the environment by the first event at 100 %. The characteristics of entropy are illustrated in Fig. 10.10 and the results are generated using the Matlab code in Listing 10.3.

Listing 10.3 A Matlab example—shows entropy noise

```
1 clear all;
2 close all;
3
4 p=0:0.01:1;
5 s=length(p);
6
```

```
randn('seed',13);
r=0.01*randn(1,s);
q=p+r;
q=(q-min(q))/(max(q)-min(q));

x=-p.*log2(p);
y=-q.*log2(q);

figure;plot(x);grid on;
figure;plot(x);grid on;
hold on;plot(y,'color','red');
```

Thinking with Example 10.5:

Now suppose we take the probability for the first and second events as 0.5 (i.e., $p_1 = p_2 = 0.5$). Then we can denote the probabilities of the first and second events by $[0.5, 0.5]$ matrix. In this case, the entropy is: $\text{en} = -0.5 \times \log(0.5) - 0.5 \times \log(0.5) = 1$. This is the maximum error as shown in Fig. 10.10, and it means that we cannot characterize the environment by either of these two events.

Thinking with Example 10.6:

In the next example, suppose the probability for the first and second events are $p_1 = 0.9$ and $p_2 = 0.1$, then we can denote the probabilities of the events by $[0.9, 0.1]$ matrix. If we calculate the entropy for these events, then we have $\text{en} = -0.9 \times \log(0.9) - 0.1 \times \log(0.1) = 0.4690$.

Thinking with Example 10.7:

Similarly, if we take the probabilities as $p_1 = 0.7$ and $p_2 = 0.3$, then the entropy is 0.8813. We can see the probabilities reach $[0.5, 0.5]$ and the entropy (i.e., error) is getting higher.

10.4.2 Gini Impurity

The Gini impurity is another type of measure which can be used to measure incorrect labelling [3, 7] with matching the patterns. The abbreviation Gini stands for generalized inequality index. The Gini impurity may be used in decision tree building, instead of entropy, and it is defined as follows [3]:

$$\text{Gini} = -\sum_i p_i(1 - p_i), \tag{10.5}$$

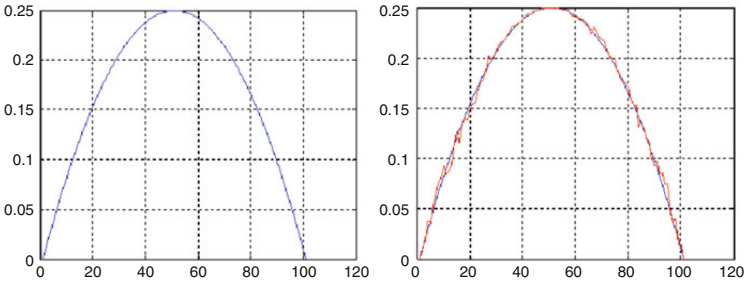

Fig. 10.11 The Gini indexes characteristics with and without an error factor, respectively—horizontal axis represents the probability index and the vertical axis represents the Gini impurity

where p_i is the probability of the ith event. The characteristics of Gini impurity are illustrated in Fig. 10.11. We can see the differences in entropy (in Fig. 10.10) and Gini impurity (in Fig. 10.11) clearly. One difference is the spread and the other is skewness (symmetry).

Listing 10.4 A Matlab example—shows Gini impurity

```
clear all;
close all;

p=0:0.01:1;
s=length(p);

randn('seed',13);
r=0.01*randn(1,s);
q=p+r;
q=(q-min(q))/(max(q)-min(q));

x=p.*(1-p);
y=q.*(1-q);

figure;plot(x);grid on;
figure;plot(x);grid on;
hold on;plot(y,'color','red');
```

We have seen that the entropy measure is useful when multiple events occur at a particular instance or a location. In contrast, the cross-entropy is used to measure the error when multiple events (but the same events) occur at two different instances or locations. We can distinguish these two cases as follows: in the first case, the events come from the same statistical distribution, but in the second case, the events are from two distributions. Therefore, we can say the entropy measures within as the intra-error (i.e., within a distribution) and cross-entropy as the inter-error between two distributions. Suppose two events a and b occur at two different instances with

probabilities $[p_1, p_2]$, and $[q_1, q_2]$, where $p_1 + p_2 = 1$ and $q_1 + q_2 = 1$, then the cross-entropy is defined as follows [6]:

$$x\_\text{en} = -p_1 \log_2(q_1) - p_2 \log_2(q_2). \tag{10.6}$$

In the actual entropy definition, the probabilities p_1 and p_2 occupy the places of q_1 and q_2. Therefore, the difference in the probability q from p is reflected on the cross-entropy. We can generalize this as follows [8]:

$$x\_\text{en} = -\sum_i p_i \log_2(q_i), \tag{10.7}$$

where $\sum p_i = 1$ and $\sum q_i = 1$. Let us now understand the meaning of the cross-entropy through some examples.

Thinking with Example 10.11:

In this example, suppose a container has 10 balls, and the only additional information given to you is that either all of them are red balls or all of them are blue balls. However, I know that the container has only 10 red balls. The game is that I draw a ball without showing it to you, but you must predict its color by guessing it 10 times. Because I know that all the balls are red, the actual probability matrix is $[1, 0]$. Now suppose your predicted probability is $[1, 0]$ (i.e., all 10 guesses you said red), then the cross-entropy is $x\_\text{en} = -1 \times \log(1) - (0) \times \log(0) = 0 - 0 \times \infty = 0$. This indicates that there is no error in actual and predicted values. Suppose your predicted probability is $[0, 1]$. It means all 10 guesses you said blue, then the cross-entropy is: $x\_\text{en} = -1 \times \log(0) - (0) \times \log(1) = -1 \times \infty - 0 \times 0 = -\infty$. It indicates a very large error in your prediction. Suppose your probability matrix is [0.9, 0.1]. It means you guessed 9 times as red and 1 time as blue. Then the cross-entropy is: $x\_\text{en} = -1 \times \log(9/10) - (0) \times \log(1/9)$. The program Listing 10.5 may be used for this purpose and it produces the results in Fig. 10.12.

Listing 10.5 A Matlab example—shows cross-entropy

```
clear all;
close all;

p=0.01:0.01:0.5;
x_en=-p.*log2(p) - (1-p).*log2(1-p);
figure;plot(p,x_en);grid on;

q1=0.01:0.01:0.99;
p1=ones(1,length(q1));

x_en1=-p1.*log2(q1) - (1-p1).*log2(1-q1);
figure;plot(q1,x_en1);
hold on;plot(0.5,1,'o');grid on;
```

Thinking with Example 10.12:

Now suppose the container has 9 red balls and 1 blue ball, then the actual probability matrix is $[0.9, 0.1]$, and your predicted probability is $[0.9, 0.1]$, then the cross-entropy is: $x\_\text{en} = -0.9 \times \log(0.9) - (0.1) \times \log(0.1) = 0.4690$. Suppose your predicted probability is $[0.1, 0.9]$, then the cross-entropy is $x\_\text{en} = -0.9 \times \log(0.1) - (0.1) \times \log(0.9) = 3.0049$. In another example, if the container has 5 red balls and 5 blue balls, and if you predict 5 times red and 5 times blue, then the actual probability matrix is $[0.5, 0.5]$, and the predicted probability measure is also $[0.5, 0.5]$. Therefore, the cross-entropy is: $x\_\text{en} = -0.5 \times \log(0.5) - (0.5) \times \log(0.5) = 1$. It is highlighted in the second figure of Fig. 10.12.

10.4.3 Information Gain

Suppose we have set of binary events, such as success or failure, rain or no rain, head or tail, etc. A set of binary events carries information that characterize the system that generated these events. This is true for any number of events, not just binary events. As we have seen, the entropy measure or the Gini impurity may be used to describe these characteristics. However, if we divide the set into two subsets, these subsets may lead to gain in the information. For example, the original set of events may have an entropy (error), and when the set is divided into subsets then the average of the entropies of these subsets may have a reduced error (entropy) leading to an information gain.

Thinking with Example 10.8:

An example is selected and the steps involved in analyzing the information gain is presented. Consider two events a and b, and a set of these binary events is $S = \{a, b, a, a, b, a, b, b, b, a, a, a\}$. There are seven observations in event a and five observations in event b. Let us write this in a matrix form: $[7a, 5b]$. They can be written as the following probability matrix: $[7/12, 5/12]$, where the first element corresponds to event a and the other event b. Therefore, the entropy of this set of binary events is: $\text{en} = -(7/12) \times \log(7/12) - (5/12) \times \log(5/12) = 0.9799$.

Thinking with Example 10.9:

Let us now split the set into two subsets. The question is, where to split? The different split locations will give different subsets and, in turn, will give a different information gain. The best information gain will help select the best split. Let us first split the set at the middle, and it gives us two subsets S_1 and S_2, where $S_1 = \{a, b, a, a, b, a\}$ and $S_2 = \{b, b, b, a, a, a\}$. We can write their event matrices

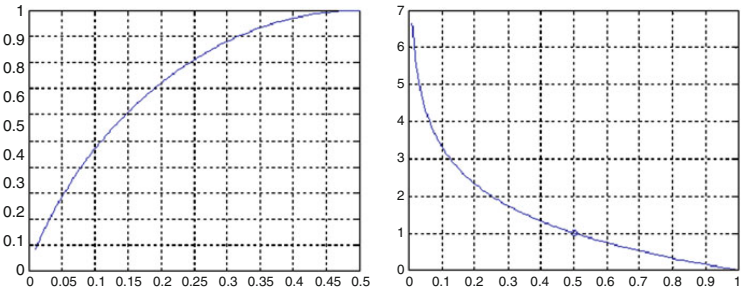

Fig. 10.12 The cross-entropy results of Listing 10.5

as $[4a, 2b]$ and $[3a, 3b]$. Therefore, their probability matrices are: $[4/6, 2/6]$ and $[3/6, 3/6]$. If we calculate their entropies as before, we get en $= 0.9183$ and en $= 1$ for these subsets of events. As we can see, the second subset has an equal number of events, and the entropy is high as illustrated earlier. The fractions of these subsets with respect to the original set are: 6/12 and 6/12. Therefore, the average entropy is: $(6/12) \times 0.9183 + (6/12) \times 1 = 0.9591$, which is smaller than the entropy 0.9799 of the original set. Therefore, there is a gain in the information or the reduction in the error (entropy) from 0.9799 to 0.9591 through the split at the middle of the set.

Thinking with Example 10.10:

What will happen if we split the data set at the 9th location and create the subsets $S_1 = \{a, b, a, a, b, a, b, b, b\}$ and $S = \{a, a, a\}$? The entropies of these subsets can be calculated, and they are 0.9911 and 0, respectively. The fractions of the subsets from the original set are 9/12 and 3/12; therefore, the average entropy of the subsets is: $(9/12) \times 0.9911 + (3/12) \times 0 = 0.7433$. This is a significant reduction in the entropy, and thus it gives a very good information gain. Among these two splits, the split at the 9th location is preferred.

10.5 Decision Tree Learning Algorithm

The goal of the decision tree learning algorithm is to focus on each one-dimensional subspace (i.e., single feature at a time), select the best feature (the best one-dimensional subspace) and best split location, extract the feature value at the best split location, and divide the domain into two subdomains. Then repeat the same process with the subdomains until no domain divisions are required (until a subdomain has a particular class significantly higher than others).

10.5.1 Training Algorithm

The best feature and the best split location of that feature are the first and the most important requirement in the training of a decision tree. As we have discussed earlier, the parametrization and the optimization are performed simultaneously at each node while a decision tree is built (i.e., the data domain and subdomains are divided). The parameters of a decision tree (at each node) are: the best features and the best features' value at the best split location—the value at the split location that gives the maximum information gain. The leafs of a node are the subdomains. Hence, the training algorithm is as follows:

1. Assuming the data is p-dimensional, analyze all the p, one-dimensional subspaces (i.e., a feature at a time) to search for the best features and split locations. If p is high, then the search will be exhaustive, and the process will be computationally expensive.

2. In each one-dimensional subspace, its data domain will be split at each location (i.e., $n-2$ locations, if there are n observations) and the information gain resulted from the split to two subdomains will be calculated.

3. For each feature, the best split location is selected based on the split that gives the highest information gain, and the feature values at that location will be recorded.

4. The best feature and the feature value at the split location will be assigned to the node that is being processed.

5. The data domain is then divided into two subdomains ($SD1$ and $SD2$) at the split locations.

6. Steps 1–5 will be repeated for the subdomains $SD1$ and $SD2$ to get the best feature, best split location, and the best split feature value for the new nodes.

7. The decision tree will be built following these processes until the subdomain that does not need a split because of the subdomain that has a significantly large number of observations from a single class.

The Matlab code, created as a function named *calc_ig_fn* and presented in Listing 10.6, illustrates these tasks. It uses the hardwood floor and carpet floor data sets presented previously and generates a decision tree. It generates only a two-level decision tree as illustrated in Fig. 10.13.

Listing 10.6 A Matlab example—a function to create information gain

```
1  function [sl,mx,ig]=calc_ig_fn(xx,yy)
2
3      ll=length(yy);
4      for ii=2:ll-1
5          %sp is the split location
```

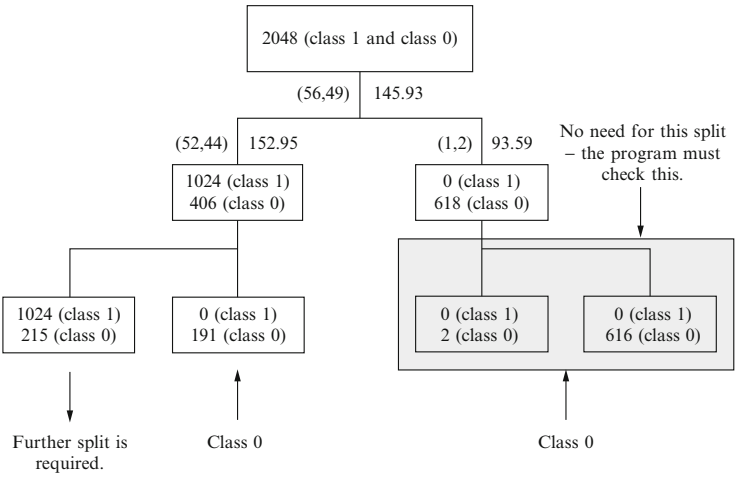

Fig. 10.13 This is the two-level decision tree of the data. The visual effects of these results are presented in Fig. 10.14

```
6         sp=xx(ii);
7
8         %build left and right tree
9         xl=yy(xx<sp);
10        xr=yy(xx>=sp);
11
12        %length of the left tree
13        l1=length(xl);
14        %sum of class 1s
15        n1=sum(xl);
16        %sum of class 0s
17        n2=l1-n1;
18
19        %probabilities
20        p1=n1/(n1+n2)+0.0001; %1.0e-14; %0.000001;
21        p2=n2/(n1+n2)+0.0001; %1.0e-14; %0.000001;
22
23        %entropy of the left tree
24        en1 = -p1*log(p1)-p2*log(p2);
25
26        %length of the right tree
27        l2=length(xr);
28        %sum of class 1s
29        n1=sum(xr);
30        %sum of class 0s
31        n2=l2-n1;
32
```

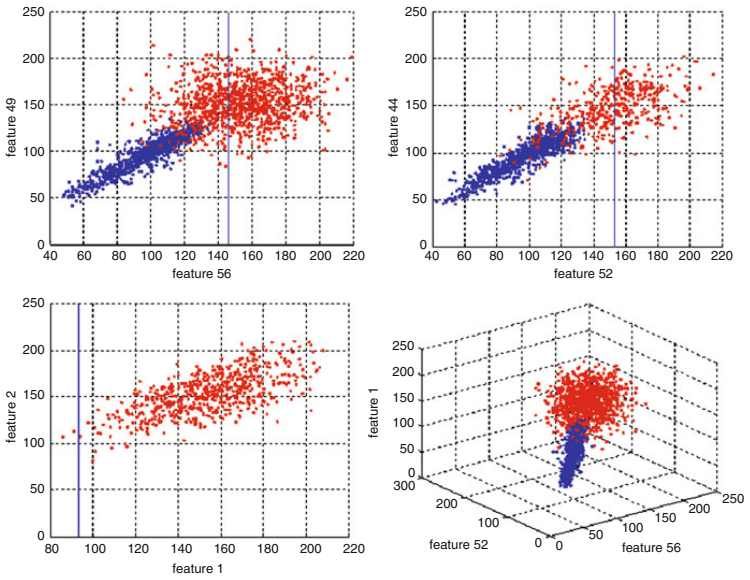

Fig. 10.14 The visual effects of the results obtained in the two-level decision tree presented in the previous figure

```
        %probabilities
        p1=n1/(n1+n2)+0.0001; %1.0e-14; %0.000001;
        p2=n2/(n1+n2)+0.0001; %1.0e-14; %0.000001;

        %entropy of the right tree
        en2 = -p1*log(p1)-p2*log(p2);

        %Calculates information gain
        ig(ii-1)=1-((l1/ll)*en1 + (l2/ll)*en2);
    end
    %find the split location with the maximum information gain
    tmp=find(ig==max(ig));
    sl=tmp(1);
    mx=max(ig);
end
```

This function accepts the data domain stored in a variable *xx* together with their corresponding class labels (only two classes), and then gives the best feature *mx* and best feature split *sl*. It also gives information gain values *ig* for each feature at each split. The second task is to customize the data, pass it through the function, obtain the best feature and best split location, find the feature value at that location, split the data domain (i.e., the data table) at that location, and analyze if the subdomains

(left and right) require further split. The Matlab code in Listing 10.7 perform these tasks only at two nodes sequentially. The program has been written to simplify the explanation, and thus it requires programming enhancement. One way to do it is to write this program with recursive, iterative statements or functions.

Listing 10.7 A Matlab example—decision tree building algorithm

```
function best_subspaces=dt_build_fn(hw,cp)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Building the root node
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

yy=[ones(1,size(hw,1)) zeros(1,size(cp,1))];
xx=[hw;cp]';

for ii=1:size(xx,1)
    [slx(ii),fmx(ii),igx{ii}]=calc_ig_fn(xx(ii,:),yy);
end

fmxsort=sort(fmx,'descend');
tmp1=find(fmx==fmxsort(1));
f1=tmp1(1);
fmx(f1)=0;
tmp2=find(fmx==fmxsort(2));
f2=tmp2(1);
fval=xx(f1,slx(f1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Splitting the root node
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

idxL=find(xx(f1,:)<fval);
idxR=find(xx(f1,:)>=fval);

xxL=xx(:,idxL);
xxR=xx(:,idxR);

yyL=yy(idxL);
yyR=yy(idxR);

fprintf('\n');
fprintf('Left␣side␣class␣1␣=␣%d\n',sum(yyL));
fprintf('Left␣side␣class␣0␣=␣%d\n\n',length(yyL)-sum(yyL));

fprintf('Right␣side␣class␣1␣=␣%d\n',sum(yyR));
fprintf('Right␣side␣class␣0␣=␣%d\n',length(yyR)-sum(yyR));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Building the left node
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for ii=1:size(xxL,1)
```

```
47      [slxL(ii),fmxL(ii),igxL{ii}]=calc_ig(xxL(ii,:),yyL);
48  end
49
50  fmxsortL=sort(fmxL,'descend');
51  tmp3=find(fmxL==fmxsortL(1));
52  f1L=tmp3(1);
53  fmxL(f1L)=0;
54  tmp4=find(fmxL==fmxsortL(2));
55  f2L=tmp4(1);
56  fvalL=xx(f1L,slxL(f1L));
57
58  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
59  % Splitting the left node
60  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
61
62  idxLL=find(xxL(f1L,:)<fvalL);
63  idxRL=find(xxL(f1L,:)>=fvalL);
64
65  xxLL=xxL(:,idxLL);
66  xxRL=xxL(:,idxRL);
67
68  yyLL=yyL(idxLL);
69  yyRL=yyL(idxRL);
70
71  fprintf('\n');
72  fprintf('Left␣side␣class␣1␣=␣%d\n',sum(yyLL));
73  fprintf('Left␣side␣class␣0␣=␣%d\n\n',length(yyLL)-sum(yyLL));
74
75  fprintf('Right␣side␣class␣1␣=␣%d\n',sum(yyRL));
76  fprintf('Right␣side␣class␣0␣=␣%d\n',length(yyRL)-sum(yyRL));
77
78  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
79  % Building the right node
80  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
81
82  for ii=1:size(hw,2)
83      [slxR(ii),fmxR(ii),igxR{ii}]=calc_ig(xxR(ii,:),yyR);
84  end
85
86  fmxsortR=sort(fmxR,'descend');
87  tmp5=find(fmxR==fmxsortR(1));
88  f1R=tmp5(1);
89  fmxR(f1R)=0;
90  tmp6=find(fmxR==fmxsortR(2));
91  f2R=tmp6(1);
92  fvalR=xx(f1R,slxR(f1R));%%%%%%%%%%%%%%%%%%%%%%%%%
93
94  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
95  % Splitting the right node
96  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
97
98  idxLR=find(xxR(f1R,:)<fvalR);
99  idxRR=find(xxR(f1R,:)>=fvalR);
100
```

```
xxLR=xxR(:,idxLR);
xxRR=xxR(:,idxRR);

yyLR=yyR(idxLR);
yyRR=yyR(idxRR);

fprintf('\n');
fprintf('Left␣side␣class␣1␣=␣%d\n',sum(yyLR));
fprintf('Left␣side␣class␣0␣=␣%d\n\n',length(yyLR)-sum(yyLR));

fprintf('Right␣side␣class␣1␣=␣%d\n',sum(yyRR));
fprintf('Right␣side␣class␣0␣=␣%d\n',length(yyRR)-sum(yyRR));
fprintf('================================================\n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Printing root node and split
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure; grid on;
for ii=1:size(xx,2)
   if(yy(ii)==1)
      hold on;plot(xx(f1,ii),xx(f2,ii),'.');
   else
      hold on;plot(xx(f1,ii),xx(f2,ii),'r.');
   end
end
hold on;line([fval fval],[0 250]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Printing left node and split
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure; grid on;
for ii=1:size(xxL,2)
   if(yyL(ii)==1)
      hold on;plot(xxL(f1L,ii),xxL(f2L,ii),'.');
   else
      hold on;plot(xxL(f1L,ii),xxL(f2L,ii),'r.');
   end
end
hold on;line([fvalL fvalL],[0 250]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Printing right node and split
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure; grid on;
for ii=1:size(xxR,2)
   if(yyR(ii)==1)
      hold on;plot(xxR(f1R,ii),xxR(f2R,ii),'.');
   else
      hold on;plot(xxR(f1R,ii),xxR(f2R,ii),'r.');
   end
end
hold on;line([fvalR fvalR],[0 250]);

```

```
155 best_subspaces=[f1 f2; f1L f2L; f1R f2R];
156
157 end
```

In this program the first two blocks of codes from line 3 to line 20 and from line 22 to 40 perform the building and splitting the root node, respectively. After splitting, it builds the left node which is provided in the lines 42–56 and then split the left node. The block of code from line 58 to 76 performs this left node split and then the code from line 78 to 113 perform the right node split. The rest of the code prints the results as commented in the program.

Listing 10.8 A Matlab example—initiates decision tree training

```
1 clear all;
2 close all;
3
4 hw=csvread('hardwood.csv');
5 cp=csvread('carpet.csv');
6
7 best_subspaces=dt_build_fn(hw,cp);
```

This program simply calls the function presented in the Listing 10.7 and performs the two-level decision tree construction for the input data the hardwood floor (Fig. 10.15) and the carpet floor (previously used). The results will be the best sub spaces.

10.5.2 Validation Algorithm

The cross-validation may be conducted to determine the depth of the tree, and it can allow a tree pruning mechanism to speed up the testing when the new data arrives to be classified. Hence, it can bring computational advantages.

10.5.3 Testing Algorithm

The testing algorithm is simple in decision tree learning. It takes one observation at a time from the test data (Fig. 10.16) and puts it through the decision tree constructed using the training algorithm. The feature and the feature value at the root node of the decision tree classifier are observed, and they are used to assign the observation to the left or the right tree. It is pushed into the tree until it reaches one of the leaves. Then the corresponding class label is assigned to the observation if it is a classification tree, and if it is a regression tree then the actual predicted value is assigned.

Fig. 10.15 Training Image Hardwood Floor

Listing 10.9 A Matlab example—a simple decision tree testing

```
clear all;
close all;

hw=csvread('hardwood.csv');
cp=csvread('carpet.csv');

oo=size(hw,1);
ff=size(hw,2);
rand('seed',131);
rr=1:2048; %randperm(2048);

%1 represents hardwood floor and 0 represents carpet floor
ty=[ones(1,oo) zeros(1,oo)];
tx=[hw;cp]';
yy=ty(rr);
xx=tx(:,rr);

%For each feature xx{ii}, it provides information gains igx{ii}
%For each feature xxii}, it also provides split locations slx{ii}
for ii=1:ff
    [slx(ii),fmx(ii),igx{ii}]=calc_ig(xx(ii,:),yy);
end

fnum=find(fmx==max(fmx))
w1=slx(fnum)+1;
fmx(fnum);
```

Fig. 10.16 Test Image Hardwood Floor

```
27
28 in=xx(fnum,w1)
29
30 tt1=csvread('test_hw3.csv');
31 hw=tt1';
32
33 f1=find(hw(fnum,:)<in);
34 f2=find(hw(fnum,:)>=in);
35
36 %total on the left side is 1430
37 length(yy(f1))
38
39 %left hand side all 1024 hardwood and 1430-1024=406 carpet
40 sum(yy(f1))
41
42 %right hand side ALL carpets 618
43 sum(yy(f2))
44
45 figure; grid on;
46 for ii=1:1024
47    if(yy(ii)==1)
48       hold on;plot(hw(56,ii),hw(52,ii),'.');
49    else
50       hold on;plot(hw(56,ii),hw(52,ii),'r.');
51    end
52 end
53 hold on; line([in in],[0 250]);
```

10.6 Decision Tree and Big Data

In this section, a toy example to help implement decision tree modeling and algorithm is presented. It uses the sorting and parallelization features of the big data processing platform, a single node RHadoop with R programming environment, and the MapReduce framework presented in Chaps. 4 and 5. The structure of the toy example may be used to write the full implementation of the decision tree and process real data sets to illustrate the performance of the decision tree classification on a Hadoop platform.

10.6.1 Toy Example

This example does not provide a program for building a decision tree; instead, it brings the features of MapReduce framework that can help you write a program to implement decision tree supervised learning. The content of the data file used for this purpose is presented below:

```
1 4 1
2 5 1
3 12 0
8 6 1
9 9 1
1 12 0
```

It shows a three-column table with six observations. The first two columns show the data points (or the data domain), and the last column shows the class labels (or the response set). The goal of this program is to divide the data domain based on the mean value of feature 2 (second column) first, then map them to the labels. This will be the split at the root node of the tree. Then repeat the process on the left side (left subdomain) of the tree and the right side (right subdomain) of the tree until the split is no longer necessary.

Listing 10.10 An RHadoop example—it can help you write a decision tree technique for big data applications

```
1  Sys.setenv(HADOOP_HOME='/usr/lib/hadoop-0.20-mapreduce')
2  Sys.setenv(HADOOP_CMD='/usr/bin/hadoop')
3  Sys.setenv(HADOOP_STREAMING='/usr/lib/hadoop-0.20-mapreduce/
       contrib/streaming/hadoop-streaming-2.0.0-mr1-cdh4.7.0.jar')
4
5  data <- read.table("tree1.txt", sep="")
6
7  library(rmr2)
8  library(rhdfs)
9
10 hdfs.init()
11
```

```
12 gauss.data = to.dfs(data)
13
14 gauss.map.fn = function(k, v) {
15
16  #Split at the root
17  m1=mean(v[,2])
18  k=ifelse(v[,2]<m1,1,0)
19  v[,2]=k
20  keyval(k,v)
21
22  #divide the tree
23  l=which(v[,2]==0)
24  r=which(v[,2]==1)
25
26  vl=v[l,]
27  vr=v[r,]
28
29  #Split needed on the left side
30  m2=mean(vl[,1])
31  kl=ifelse(vl[,1]<m2,0,1)
32  vl[,1]=kl
33  keyval(kl,vl)
34
35  #Split not needed on the right side
36  kr=vr[,3]
37  vr[,1]=kr
38  keyval(kr,vr)
39
40  #Concatenate (key, value) pair
41  c.keyval(keyval(kl,vl),keyval(kr,vr))
42 }
43
44 gauss.reduce.fn = function(k, v) {
45
46   keyval(k, v)
47 }
48
49 mr.gauss = mapreduce(input = gauss.data, map = gauss.map.fn,
       reduce = gauss.reduce.fn)
50
51 mr.results = from.dfs(mr.gauss)
52 mr.results
```

We have already seen some of the statements required for performing MapReduce using rmr2 package [9]. The codes needed for this specific example are described below. The block of code in lines 16–20 splits the data domain based on the mean of feature 2 (line 17) and labels them with 1 and 0 (line 18). Then it generates (key, value) at lines 19 and 20—this will help the mapreduce() function to sort them with respect to the key value, which is the label. In the block of code in lines 23–27, the subdomains (or the left and right children) are created. The block of code from lines 29–33 performs the split using feature 1 as in lines 16–20. This should be done either iteratively or recursively as, this being a toy example, these

tasks are performed sequentially. The right split is not required, as we can see the output after labeling the data based on the split condition of feature 2. However, a (key, value) pair is generated for the right child of the tree. The (key, value) pairs are then concatenated in line 41. The output of the program is given below, which gives the sorted (key, value) pair:

```
$key
[1] 0 0 1 1 1 1

$val
  V1 V2 V3
3  0  0  0
6  0  0  0
5  1  0  1
1  1  1  1
2  1  1  1
4  1  1  1
```

As we can see, the data is sorted with respect to the key, which is the label of the classes in column three of the data set. Based on the mean value of feature 2, the data domain is split at the third row, and class 0 is assigned to the top half of feature 2, and class 1 is assigned to the bottom half of feature 2. If we compare the feature 2 column (V2) and the labels column (V3), then we can see the majority of the actual labels matches, except the third row. If we now extend this to feature 1, then its label correctly matches with the actual label. This output is similar to the results presented in Fig. 10.2.

Problems

10.1. Code Revision
Revise the MapReduce programs presented in this chapter using the coding principles taught in Chap. 5.

10.2. Building a Decision Tree
The decision tree illustrated in Fig. 10.1 used the tree split using X_3, X_2, and X_1 order. Reproduce the decision tree with the order X_2, X_1, and X_3. Then compare the classifiers and the results.

10.3. Real Example

(a) Complete the toy example using iteration (or recursion) together with the actual method of calculating split location and selection of features using the Gini index and the information gain approaches.
(b) Use this implementation to the real data sets like the hardwood floor and carpet floor data sets. Make sure the programs follow the coding principles presented in a previous chapter.

References

1. S. B. Kotsiantis. “Supervised machine learning: A review of classification techniques,” Informatica 31, pp. 249–268, 2007.
2. S.K. Murthy. “Automatic construction of decision trees from data: A multi-disciplinary survey,” Data Mining and Knowledge Discovery, Kluwer Academic Publishers, vol. 2, no. 4, pp. 345–389, 1998.
3. http://en.wikipedia.org/wiki/Decision_tree_learning
4. L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. “Classification and Regression Trees,” CRC Press, 1984.
5. L. Torgo. “Inductive learning of tree-based regression models,” PhD Thesis, Department of Computer Science, Faculty of Science, University of Porto, Porto, Portugal, pp. 57–104, 1999.
6. T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning. New York: Springer, 2009.
7. https://www.stat.berkeley.edu/∼breiman/RandomForests/cc_home.htm
8. L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. “Regularization of neural networks using dropconnect.” In Proceedings of the 30th International Conference on Machine Learning (ICML-13), pp. 1058–1066, 2013.
9. http://www.rdocumentation.org/packages/rmr2.