CHAPTER 7
OPTIMAL DECISION TREE ALGORITHMS

7.1 Introduction

Decision trees are developed to represent a system of sequential decision making process, where information is used in pieces in order to arrive at a unique decision. In addition to being unique, this decision is also required to be ‘the best’. Since there are two or more stakeholders involved within the course of decision making, the concept of the best solution may be subjective and may change from one stakeholder to another. It is therefore necessary to take a stand and search for the best solution from that stand point. Even then, the best solution requires a criterion for comparing one solution with another to decide which solution is better among the two. This comparison can then be extended to a multiple of solutions and the best solution can be found. However, this procedure can be highly complex, requiring a large amount of resources like time and space. It is therefore preferred to make decisions in a sequence, so that every decision can be optimized under the prevailing conditions. It may be easy to understand this principle with help of one concrete example. Consider the Play Tennis example that can be found abundantly in the literature. There are four attributes as inputs. These attributes with their possible values are as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook</td>
<td>Sunny, Overcast, Rain</td>
</tr>
<tr>
<td>Humidity</td>
<td>High, Normal</td>
</tr>
<tr>
<td>Wind</td>
<td>Strong, Weak</td>
</tr>
<tr>
<td>Temperature</td>
<td>Hot, Mild, Cool</td>
</tr>
</tbody>
</table>

The objective is to see whether to play or not, while the values of all the four attributes are known.

Theoretically speaking, two of the attributes, namely, outlook and temperature, have three distinct values each, while the remaining two attributes, namely, humidity and wind, have two distinct values each. As a result, there can be a total of 36 distinct combinations of the four attributes. A rule-based system would specify the output for each of these 36 combinations, so that on any given day, it would be very to decide whether or not to play tennis simply by looking at the appropriate entry in the decision table.
The problem is not so simple because the dataset contains only 14 of the 36 possible combinations of the four input attributes. This situation poses two problems. First, how to make decision when an attribute value combination occurs that is not in the given dataset. Second, whether it is necessary to take into consideration values of all the four attributes for deciding whether or not to play. What is required for a decision to be rational is that it is consistent. It can be easily verified in this regard that there is always a consistent decision for any dataset when a separate decision is made for every instance in the dataset. However, such a decision would not generalize to new instances. What is therefore required is to develop more compact decision rules. The trivial decision tree would have a path to leaf for every individual data point in the given dataset. For example, in the Play Tennis example, it is easy to draw a tree having 14 leaves, but without any clue as to whether to play or not as soon as the four attributes take such values that their combination does not occur in the given dataset. It is therefore required to ‘learn’ from the given dataset, so that a decision can be made even if there is an invisible situation in terms of a particular sequence of values of the input attributes.

We consider the given dataset that has 14 instances as follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>
The given dataset can be re-written by sorting values of one input attribute at a time. The following four tables are obtained in this way.

**Data sorted by outlook**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>

**Data sorted by Temperature**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Weather</td>
<td>Temperature</td>
<td>Humidity</td>
<td>Wind</td>
<td>Label</td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
<td>-------------</td>
<td>----------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>X9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Data sorted by Humidity**

<table>
<thead>
<tr>
<th></th>
<th>Weather</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>X13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Data sorted by wind**

<table>
<thead>
<tr>
<th></th>
<th>Weather</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>X9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>X2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
</tbody>
</table>
X11  Sunny  Mild  Normal  Strong  Yes  
X12  Overcast  Mild  High  Strong  Yes  
X14  Rain  Mild  High  Strong  No  

Even after sorted the rows by values of one attributes at a time, there is no clear generalization of the decisions that appear in the last column of every table. This happens because the tables contain more data than is necessary to arrive at a decision. The following compact presentation of data may make it easy to perceive the relations between every input attribute and the output attribute.

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Sunny</th>
<th>Overcast</th>
<th>Rain</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Hot</th>
<th>Mild</th>
<th>Cool</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Humidity</th>
<th>Sunny</th>
<th>Overcast</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Wind</td>
<td>Wind</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Weak</td>
<td>Strong</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

The purpose of preparing the compact tables is to measure utility of every attribute and find out which attributes are good and which attributes are poor. An attribute is good when, for at least one of its values, all instances have a common output values.

On the other hand, an attribute is likely to be poor when

- it does not provide any discrimination.
- the attribute is immaterial to the output concerned.
- The output has the same number or positive and negative instances for each value of the attributes.

These two descriptions of a good attribute and a poor attribute are rather simplistic. What is required at this stage is a rigorous measure of the quality of an attributes regarding its role in influencing the output. The concept of entropy must be introduced at this stage.

### 7.2 Entropy, Information Gain, and Gain Ratio

When the challenge is to predict a random event, entropy is the measure of its unpredictability. Information gain heuristic attempts to reduce the entropy with help of a set of examples by grouping them in such a way that their classifications are more in agreement than in the set of all examples taken together.

Let $X$ is a discrete random variable, having $n$ possible values $x_1, x_2, \ldots, x_n$ with corresponding probabilities $P(x_1), P(x_2), \ldots, P(x_n)$. The entropy, $H$, of $X$ is then defined by

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

As $P(x_i)$ can take the value 0 or 1 for some $x_i$, it is assumed that

$$P(x_i) \log_2 P(x_i) = 0$$
When $P(x_i) = 0$ or $1$ for some $x_i$.

The predictability of a classification is made higher by reducing entropy. When entropy is reduced to zero, it is meant to indicate that the group of examples represents their common classification.

Information gain is the change in entropy when a node in a decision tree is split into its child nodes. The entropy of the parent node is then compared to the weighted sum of the entropies of its child nodes, when a splitting attribute is specified.

In a binary classification, the entropy of a collection $S$ of positive and negative cases, the entropy of $S$ is,

$$H(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-),$$

In the above expression, $p_+$ indicates the proportion for positive cases and $p_-$ indicates the proportion for negative cases in $S$.

Returning to the PlayTennis example, $S$ is a collection of fourteen cases with nine positive cases and five negative ones. Therefore, the entropy of $S$ relative to the Boolean classification is,

$$H(+9, -5) = -\frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) = 0.940$$

It may be noted that the entropy would be zero, if each and every members of $S$ belong to the same unit.

Now, let’s consider the partitions of the set of 14 cases in the Play Tennis example induced by each of the four attributes, one after another.

First, consider the partition induced by the attribute “Outlook”. Let $S_i$ be the set of cases where

Outlook = Sunny,

$S_2$ be the set of cases where

Outlook = Overcast and

$S_3$ be the set of cases where

Outlook = Rain.
Then, $S_1$ has two positive cases and three negative ones; $S_2$ has four positive cases and no negative cases, while $S_3$ has three positive cases and two negative cases. As a consequence, it can be easily shown that

$$H(S_1) = -\frac{2}{5} \log_2 \left( \frac{2}{5} \right) - \frac{3}{5} \log_2 \left( \frac{3}{5} \right) = 0.97,$$

$$H(S_2) = -\frac{4}{4} \log_2 \left( \frac{4}{4} \right) - 0 \log_2 \left( \frac{0}{4} \right) = 0,$$

$$H(S_3) = -\frac{3}{5} \log_2 \left( \frac{3}{5} \right) - \frac{2}{5} \log_2 \left( \frac{2}{5} \right) = 0.97,$$

so that their weighted average is

$$H(Outlook) = \frac{5}{14} \times H(S_1) + \frac{4}{14} \times H(S_2) + \frac{5}{14} \times H(S_3)$$

$$= \frac{5}{14} \times 0.97 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.97$$

$$= 0.693.$$

Taking the difference between $H(S)$ and $H(Outlook)$, one obtains

$$gain(Outlook) = H(S) - H(Outlook)$$

$$= 0.94 - 0.693$$

$$= 0.247.$$

Next, consider the partition of $S$ induced by the attribute “Temperature”. In this case, let $S_1$ be the set where Temperature = Hot, $S_2$ be the set where Temperature = Mild, $S_3$ be the set where Temperature = Cool.

Then, $S_1$ has 2 positive cases and 2 negative cases, $S_2$ has 4 positive cases and 2 negative cases, while $S_3$ has 3 positive cases and 1 negative cases. The entropy of this partition is therefore obtained as follows.
\[ H(S_1) = -\frac{2}{4} \log_2 \left( \frac{2}{4} \right) - \frac{2}{4} \log_2 \left( \frac{2}{4} \right) \]
\[ = \frac{1}{2}(-1) \cdot \frac{1}{2}(-1) \]
\[ = 1 \]
\[ H(S_2) = -\frac{4}{6} \log_2 \left( \frac{4}{6} \right) - \frac{2}{6} \log_2 \left( \frac{2}{6} \right) \]
\[ = 0.9183 \]
\[ H(S_3) = -\frac{3}{4} \log_2 \left( \frac{3}{4} \right) - \frac{1}{4} \log_2 \left( \frac{1}{4} \right) \]
\[ = 0.8113. \]

These three together give

\[ H(Temperature) = \frac{4}{14} \times H(S_1) + \frac{6}{14} \times H(S_2) + \frac{4}{14} \times H(S_3) \]
\[ = \frac{4}{14} \times 1 + \frac{6}{14} \times 0.9183 + \frac{4}{14} \times 0.8113 \]
\[ = 0.911. \]

Similarly, it is easy to find that

\[ H(Humidity) = 0.788, \quad and \]
\[ H(Wind) = 0.892 \]

As a consequence of the values of entropy as obtained above, the gains in the four cases are given below:

\[ gain(Outlook) = 0.94 - 0.693 \]
\[ = 0.247, \]
\[ gain(Temperature) = 0.94 - 0.911 \]
\[ = 0.029, \]
\[ gain(Humidity) = 0.94 - 0.788 \]
\[ = 0.152 \quad and \]
\[ gain(Wind) = 0.94 - 0.892 \]
\[ = 0.048 \]

These four computations reported above show that ‘Outlook’ has the maximum information gain and hence ‘Outlook’ should be selected as the first splitting attribute for constructing the tree.
Considering the partition generated by the attribute ‘Outlook’, it is easy to note that $S_2$ (Outlook = Overcast) has all positive cases, and hence need not be split any more. However, since $S_1$ and $S_3$ must be split further. Hence, consider $S_1$ first, which defined by ‘Outlook’ = Sunny.

Partitioning $S_1$ by ‘Temperature’ produces the following partition.

\[
\begin{array}{ccc}
\text{Outlook = Sunny} & (S_1) \\
\text{Temperature} & \\
\text{Hot (S}_{11}\text{)} & \text{Mild (S}_{12}\text{)} & \text{Cool (S}_{13}\text{)} \\
\text{no} & \text{yes} & \text{yes} \\
\text{no} & \text{no} & \\
\end{array}
\]

Here, $H(S_1) = 0.97, \ H(S_{11}) = H(S_{13}) = 0, H(S_{12}) = 1$

and hence

\[
H(Temperature|Outlook = Sunny) = \frac{2}{5} \times 0 + \frac{2}{5} \times 1 + \frac{1}{5} \times 0
\]

= 0.40,

giving

\[
gain(Temperature|Outlook = Sunny) = 0.97 - 0.40 = 0.57
\]

Similarly, partitioning $S_1$ by ‘Humidity’, the following partition is obtained.

\[
\begin{array}{ccc}
\text{Outlook = Sunny} & (S_1) \\
\text{Humidity} & \\
\text{High (S}_{11}\text{)} & \text{Normal (S}_{12}\text{)} \\
\text{no} & \text{yes} \\
\text{no} & \text{yes} \\
\text{no} & \\
\end{array}
\]

$H(S_{11}) = H(S_{12}) = 0$, so that
\[ H(\text{Humidity} \mid \text{Outlook} = \text{Sunny}) \]
\[ = 0 \]

and hence

\[ \text{gain}(\text{Humidity} \mid \text{Outlook} = \text{Sunny}) \]
\[ = 0.97 - 0 \]
\[ = 0.97 \]

Finally, partitioning \( S_1 \) according to wind to conditions generates the following partition

\[
\begin{align*}
\text{Outlook} &= \text{Sunny} \quad (S_1) \\
\text{Wind} \\
\text{Weak} (S_{11}) &\quad \text{Strong} (S_{12}) \\
\text{yes} &\quad \text{yes} \\
\text{no} &\quad \text{no} \\
\text{no} &
\end{align*}
\]

\[ H(S_1) = 0.97 \]
\[ H(S_{11}) = 0.9183, \]
\[ H(S_{12}) = 1, \text{ so that} \]

Therefore,

\[ \text{gain}(\text{wind} \mid \text{Outlook} = \text{Sunny}) \]
\[ = 0.97 - 0.951 \]
\[ = 0.019 \]

Obviously, as gain \((\text{Humidity} \mid \text{Outlook}=\text{Sunny})\) is the highest, select ‘Humidity’ as the splitting attribute. Also since the entropy has been reduced to zero, no further splitting is necessary.

Finally, consider the case where \( \text{Outlook} = \text{Rain} \). Partitioning \( S_3 \) according to each of the other three attributes gives the following results.

\[
\begin{align*}
\text{Outlook} &= \text{Rain} \quad (S_3) \\
\text{Temperature} \\
\text{Hot} (S_{31}) &\quad \text{Mild} (S_{32}) & \quad \text{Cool} (S_{33}) \\
\text{yes} &\quad \text{yes} &
\end{align*}
\]

\[ H(S_3) = 0.97 \]
H(S₃₁) = 0
H(S₃₂) = 0.9183
H(S₃₃) = 1

\[
H(\text{Temperature } e|\text{Outlook } = \text{Rain}) \\
= 0 \times 0 + \frac{3}{5} \times 0.9183 + \frac{2}{5} \times 1 \\
= 0.951
\]

\[
gain(\text{Temperature } e|\text{Outlook } = \text{Rain}) \\
= 0.97 - 0.951 \\
= 0.019
\]

Outlook = Rain (S₃)

Humidity

<table>
<thead>
<tr>
<th>High (S₃₁)</th>
<th>Normal (S₃₂)</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

H(S₃) = 0.97
H(S₃₁) = 1
H(S₃₂) = 0.9183

\[
H(\text{Humidity } e|\text{Outlook } = \text{Rain}) \\
= \frac{2}{5} \times 1 + \frac{3}{5} \times 0.9183 \\
= 0.951
\]

\[
gain(\text{Humidity } e|\text{Outlook } = \text{Rain}) \\
= 0.97 - 0.951 \\
= 0.019
\]

Outlook = Rain (S₃)

Wind

<table>
<thead>
<tr>
<th>Weak (S₃₁)</th>
<th>Strong (S₃₂)</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

H(S₃) = 0.97
\[ H(S_{31}) = 0 \]
\[ H(S_{32}) = 0 \]

\[
H(W|Outlook = Rain) = 0
\]

\[
gain(W|Outlook = Rain) = 0.97
\]

Again, gain \((Wind \mid Outlook = Rain)\) being the highest, “Wind” is selected for splitting the node “Outlook = Rain”. Since the entropy is reducing to zero, there is no need for any more split.

Collecting all the results obtained so far, the following 5 rules emerge.

Rule 1 If Outlook = Overcast, then Play = yes

Rule 2 If Outlook = Sunny, and if Humidity = High, then Play = no

Rule 3 If Outlook = Sunny, and if Humidity = Normal, then Play = yes

Rule 4 If Outlook = Rain, and if Wind = Weak, then Play = yes

Rule 5 If Outlook = Rain, and Wind = Strong, then Play = no

These five rules are given in the structure of a decision tree as follows:

**Figure 7.1 Decision Tree 1**
The Play Tennis example discussed above used the entropy as the measure of impurity, and information gain (which is same as reduction in the value of entropy) as the criterion for selecting the splitting attribute. The approach used in this example is a top-down approach since there is no back tracking. The algorithm begins by asking the question as, which attribute supposed to be used at the root of the tree? and the reply to this question is to evaluate every attribute to find how it classifies the cases in the training dataset alone. The best attribute is then selected as the root node. It’s values define the branches and the subsets defined by these values are the child nodes. The same procedure is recursively employed on all nodes until nodes with zero entropy are obtained. Even though the example used here makes use of the information gain as a quantitative measure of an attribute, it is possible to use some other measure in its place.

One more example is mentioned here, but not solved completely. It may a good practice exercise for an interested reader. The example is based on a simple question “Will the customer wait for a table?” There are 10 attributes in this example, and 12 cases are available as the training dataset. The ten attributes are listed as follows:

- **Alternate**: Is there any appropriate alternative restaurant close by
- **Bar**: Is there comfortable bar area to wait in the restaurant
- **Fri/Sat**: true on Fridays and Saturdays
- **Hungry**: whether the customer is hungry
- **Patrons**: How many persons are before now in the restaurant? (possible values are: None, Some and Full)
- **Price**: price range of the restaurant($, $$, $$$)
- **Raining**: whether it is raining outside
- **Reservation**: whether customer has made a reservation
- **Type**: the type of restaurant (French, Italian, Tahí or Burger)
- **Wait Estimate**: the waiting time predicted by the host (0-10 minutes, 10-30 minutes, 30-60 minutes, more than 60 minutes)

The question is as to which attribute should be chosen to split at the top of the tree. In other words, the question is as to which attribute provides the most information. In order to reply, this question it is obvious that some information theory is to be understood. Suppose an
event e has probability p of occurring. It is then customary to define a function I(p), called the information function that satisfies the following conditions.

- \( I(p) \geq 0 \), \( I(1) = 0 \). That is the information of any event is non-negative, and the certain event has no information.
- \( I(p_1, p_2) = I(p_1) + I(p_2) \). That is, the information from two independent events must be given the sum of their individual information.
- \( I(p) \) is continuous in p. That is, small changes in values p correspond to small change in values of I(p).

These two can be put together to obtain
\[
I(p^2) = 2I(p),
\]
and more generally
\[
I(p^n) = nI(p).
\]
This may also be expressed as
\[
I(p) = I\left(p^{1/m}\right)^m = mI(p^{1/m})
\]
hence
\[
\frac{1}{m} I(p) = I(p^{1/m})
\]
More generally, positive integers m and n,
\[
I(p^{n/m}) = \frac{n}{m} I(p)
\]
This property is true for all faction \( n/m \), which include all rational numbers. This can be extend to all positive real numbers to obtain
\[
I(p^a) = aI(p), a > 0
\]
The function that possesses property are of the form
\[
I(p) = -\log_b(p) \text{ for some base } b. \text{ Information theory uses } b = 2 \text{ and measures the amount of information in “bits” as units of measurement.}
\]
The question now is about measuring the average information when there are several events. Suppose there are J events, denoted by \( v_1, v_2, \ldots, v_J \), occurring with respective probabilities \( p_1, p_2, \ldots, p_J \), where \( (p_1, p_2, \ldots, p_J) \) is a discrete probability distribution. The expected (that is average) information of all J events taken together is
\[ E_{p \sim (p_1, \ldots, p_J)} I(p) = \sum_{j=1}^{J} p_j I(p_j) = -\sum_{j=1}^{J} p_j \log_2(p_j) \]

where \( p \) is the vector \((p_1, p_2, \ldots, p_J)\), and \( H(p) \) is known as the entropy of the discrete distribution \( p \). As one of the probabilities in the vector goes to 1, all the other probabilities go to zero, and, as a result \( H(p) \to 0 \), and this is precisely what is desired.

The algorithm C4.5, when is obtained by modifying the algorithm ID3, uses information gain as the splitting criterion. Information gain is defined as the expected reduction in entropy occurring owing to splitting of a node into its child nodes according to an attribute. In the example mentioned above, the information gain due the attribute ‘Patrons’ is 0.541, while there is no (that is, zero) information gain due to the attribute ‘Type’. As a matter of fact, in this example, ‘Patrons’ have the highest information gain of all the attributes and is therefore considered to be the root of the tree. In general, the attribute that maximizes the information gain is selected at every stage of split.

One of the problems with information gain is that it tends to partition too much, because it favors maximal splits in the sense that it likes every leaf to contain one example so that the entropy at every leaf node is zero, the best that can be achieved. This however, does not achieve any simplification and does not provide a compact classification scheme.

An alternative to information gain is the gain ratio. The interest is in large information gain, but at the same time, the interest is also in fewer partition sets. Suppose the attribute \( A \) is used to split the node \( S \) into \( J \) child nodes, \( S_j, j = 1, 2, \ldots, J \). then \( SplitInfo \) is defined as follows:

\[
SplitInfo(S, A) = -\sum_{j=1}^{J} \frac{|S_j|}{|S|} \log_2 \left( \frac{|S_j|}{|S|} \right)
\]

where \(|S_j|\) is the number of cases in branch \( j, j = 1, 2, \ldots, J \). It is desired that each term in the sum be large, which in turn means that \(|S_j| / |S|\) is desired to be large. That is, it is desired that each branch have a lot of cases. The gain ratio is then defined as follows:

\[
GainRatio(A) = \frac{Gain(S, A)}{SplitInfo(S, A)}
\]
The objective then is to maximize the gain ratio rather than information gain in order to obtain an optimal tree.

The above discussion shows that one should keep splitting dataset until one of the following events occurs.
1. No more examples are left to classify, and hence there is no point in attempting to split.
2. All cases belong to the same class, and therefore there is no need for further splitting.
3. No more attributes are available for further splitting.

In the restaurant example, for instance, the following tree is obtained.

**Figure 7.2 Decision Tree 2**

```
Patrons?
  /   |
None  Some  Full
  |     |
No    Yes

Hungry?
  /   |
No   Yes
    |
Type?
  /   |
No   No

French  Italian  Thai  Burger
  /    /    /    /
Yes  No  Fri/Sat?  Yes
       /   /
No   Yes

By the way, the information gain criterion produces the following tree for the same dataset.

**Figure 7.3 Decision Tree 3**

```
Patrons?
  /   |
None  Some  Full
  |     |
No    YES

WaitEstimate?
  /   |
>60   30-60  10-30  0-10
    |
No   Alternate?

Hungry?
  /   |
No   Yes

Reservation?
  /   |
No   Yes

Fri/Sat?
  /   |
No   Yes

Alternate?
  /   |
No   Yes

Bar?
  /   |
No   Yes

Raining?
  /   |
No   Yes
```
It may be interesting to revisit the Play Tennis example with the new selection criterion. Note that information gains for the four input attributes are as follows:

- gain (outlook) = 0.246
- gain (temperature) = 0.029
- gain (humidity) = 0.151
- gain (wind) = 0.048

Among the fore input attributes outlook and humidity have the information gain that is above the average, and hence are worth further consideration. The outlook attribute has the value sunny in five cases, overcast in four cases, and rain in five cases. The entropy of the attribute outlook, therefore, is given as

\[ H(\text{outlook}) = -\frac{5}{14} \log_2 \left( \frac{5}{14} \right) - \frac{4}{14} \log_2 \left( \frac{4}{14} \right) - \frac{5}{14} \log_2 \left( \frac{5}{14} \right) \]
\[ = 1.578 \]

Similarly,

\[ H(\text{humidity}) = -\frac{7}{14} \log_2 \left( \frac{7}{14} \right) - \frac{7}{14} \log_2 \left( \frac{7}{14} \right) \]
\[ = 1 \]

Therefore,

- gain ratio (outlook) = 0.246 / 1.578 = 0.156
- gain ratio (humidity) = 0.151 / 1.000 = 0.151

The attribute outlook would still be selected by the gain ratio criterion, but its superiority to the attribute humidity is much reduced in comparison with the same in terms of information gain.

What is interesting to note is that the gain ratio criterion has produced a simpler tree than the information gain criterion. Does this mean that information gain is not a good enough
criterion? Obviously, one needs to look for more splitting criteria before arriving at any conclusion in this regard. Gini index is available in cases where every split is only binary because every attribute has only two possible values. When the probabilities of the two possible values, usually denoted by 1 and 0, are \( p \) and \( 1 - p \), respectively, the Gini index of the attribute is defined as follows

\[
Gini(p) = 2p(1-p), \quad 0 \leq p \leq 1
\]

Some of the researchers have proposed to use \( 1 - \max(p, 1-p) \) as the splitting criterion because it can be attached an interpretation. The value \( 1 - \max(p, 1-p) \) indicates the proportion of incorrect classification if you classify, the event to happen when \( p > \frac{1}{2} \) and if you classify the event not to happen when \( p < \frac{1}{2} \). In other words, this is minimum of the larger proportion of possible misclassification. This is the reason this criterion is often described as a pessimistic or conservative criterion.

In a comparison among several selection criteria, what has been found is that the gain ration criterion produces much smaller trees when all attributes are binary. When some attributes have huge records of values, the subset criterion gives a large amount of smaller trees with an improved predictive performance and that contains much more calculation effort. The three criteria mentioned here are all based on information theory, but information theory is not the only possible basis for such criteria.

7.3. Genetic Algorithms

Genetic Algorithms is a class of adaptive heuristic algorithms which depends upon the evolutionary notions of usual choice and genetics. They belong to a rapidly growing area of artificial intelligence. They are basically inspired by Darwin’s theory about evolution that can be summarized by the famous principle of “survival of the fittest”. Genetic algorithms are based on randomization, but they also have the ability to exploit historical information and direct the future search into that region in the search space that is more likely to contain the optimal solution. Genetic algorithms are used in solving search problems with the idea that they navigate the state-space more efficiently, especially when the state-space is large, in finding near-optimal solutions that could not have been reached otherwise.

Genetic algorithms usually start with a population of candidate solutions, also called individuals in the population. These individuals are evolved so as to move towards better solutions by exploring the state-space of the given problem. Genomic encoding is used to
represent every individual. The fitness function takes an individual as its input and evaluates its fitness value. It then becomes important to pay attention to encoding and fitness function when developing a genetic algorithm. The fitness function must assign a fitness score to every individual even if all individuals are not performing satisfactorily. This is required to be done because all individuals perform poorly at the beginning of the search and even then the fitness function must drive the search in the direction of the part of state-space that contains better solutions. Another important decision is to choose the population size. The population size should not be too small because it may not be sufficient to cover the state-space for finding the solution. On the other hand, increasing the population size also increases the search time of genetic algorithms. It is therefore a common practice to run the algorithm on different population sizes so as to determine a good population size for the given problem.

After the initial population size is decided, selection schemes are used for selecting the fittest individuals from this population. Different techniques are available for selecting the fittest individual. Fitness proportionate and rank based strategy is an example of such techniques. After selecting the fittest individuals form the initial population, genetic operators like mutation and crossover are applied on them and new generation is generated. Fitness of the new individuals is then calculated. Genetic operators are to be defined in such a way that, when they generate new individuals after mutation or crossover, the fitness function should still be able to calculate their fitness values. Search cannot be continued if this was not possible. After generating new individuals, some replacement strategy is used so as to choose the initial population for the next interaction. These steps are performed repeatedly until convergence of fitness is achieved. That is, until fitness values of two successive generations do not differ, or the specified number of generations has been reached.

If old individuals are completely replaced with new ones, it is possible that the best performing individuals also would be lost. This can be overcome by making the algorithm remembers the best individuals generated so far.

It must be clear now that decision tree induction using genetic algorithms would require an encoding, initial population size, fitness function, genetic operators like mutation and crossover, selection strategy, and a replacement strategy. A brief discussion would be useful to understand how encoding is done, what are the selection and replacement strategies that can be
used, how genetic operators are implemented, and how the fitness function is selected if genetic algorithms are to be used for generating decision trees.

1. Encoding: The population is encoded using a tree-based encoding scheme. The main reason for using a tree-based encoding scheme is that it is possible to encode and manipulate trees directly. The tree comprises a root node with its set of child nodes. The additional nodes can either be decision nodes or leaf nodes. The decision nodes have associated decision variables, whereas leaf nodes contain their class labels. Each possible outcome of a decision leads to a child node. This encoding is therefore a variable length encoding since the length of encoding is seen to depend upon the depth or branching factor of a tree.

2. Crossover: A crossover is straightforward in this encoding scheme. The trees are traversed and one node is chosen randomly for each. These two nodes are swapped to get two different trees. As a result of swapping the two sub-trees, a large change will occur in the behavior of the tree. The good behavior generated by a sub-tree can then be preserved.

3. Mutation: A node in the tree is chosen randomly and turned into a leaf node for mutation. Mutation is also achieved by converting a leaf node into a randomly selected decision node. The purpose of mutation is to generate an alternative valid solution, and not necessarily a good solution. For example, a solution is still valid even if some attribute is used twice as the decision criterion in a certain path through the tree.

4. Selection Strategy: Selection strategy is required for selecting fittest individuals from the collection of individuals that are fit for crossover and mutation. Two possible selection strategies are fitness proportionate and rank based.

In the fitness proportionate strategy, function is used for assigning fitness values to all the individuals in the population and probabilities of selection of individuals are associated with these fitness values. Suppose $f_i$ denote the fitness value of individual $i$ in the population, then the probability of its selection is given by

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

where $N$ is the entire number of items in the population.

The rank order based selection strategy assigns a rank to every individual according to its fitness value. Then the fittest individuals are selected and are assigned to mutation having
probability 'p' and to crossover with probability '1– p'. The usual practice is to keep the value of p below 0.2 because mutation is less frequent than crossover.

5. Replacement Strategy: Replacement strategy is required in deciding how many individuals in the populations are to be kept for the next generation. There are two popular replacement strategies, and they are complete replacement and elitism. The complete replacement strategy, as the name suggests, involves replacing the entire population with new individuals that are generated randomly. Elitism replacement strategy, by contrast, keeps only a number of individuals in the population to be used as the subsequently generation of population.

6. Fitness Function: Genetic Algorithms make use of the fitness function to measure the fitness value of items. Three commonly used fitness functions measure the accuracy, the precision, and the recall of an individual decision tree, when predictions are made for the training data. An individual is used as the input to the fitness function that returns its fitness value as the output. In case of decision tree analysis, the decision tree is the input. The input decision tree is used for predicting labels for all cases in the training data. From the predicted labels, the accuracy, precision, or recall is calculated, depending on the fitness function that is being used. This is the fitness value of the individual decision tree.

7.4. Decision Trees with Modified Attributes

This section proposes a modification in the standard decision tree induction method when multi-value attributes are present in the data. The proposal is to convert every attribute to binary attributes. The number of binary attributes in the set will be equal to the number of distinct values that the original attribute can take. More specifically, if an attribute A having 'v' possible values, i.e. $A_1, A_2, \ldots, A_v$, it is said that the attribute A replaced with v binary attributes and these be called $A_1, A_2, \ldots, A_v$. For instance, in the Play Tennis, the attribute ‘outlook’ is having three values - sunny, overcast, and rainy. The proposed method would remove the attribute ‘outlook’ and replace it with three binary attribute ‘Sunny’, ‘Overcast’, and ‘Rainy’. Similarly, the attribute ‘Temperature’ would be removed and be replaced with three binary attributes ‘Hot’, ‘Mild’ and ‘Cool’. In this way, the proposed modification would transform the data in the Play Tennis example from having five attributes to having nine attributes.

The decision rule derived in the Play Tennis example can be written as follows:

“If outlook = sunny and humidity = high, then play = no”
“If outlook = sunny and humidity = normal, then play = yes”
“If outlook = overcast, then play = yes”
“If outlook = rainy and wind = weak, then play = yes”
“If outlook = rainy and wind = strong, then play = no”

This decision rule, which has five examples of decisions, may be indicated by a decision tree that has five leaf nodes. The five conditions stated above indicate paths to five leaf nodes beginning from the root node. The decision tree, thus, represents the above five decisions as follows:

Figure 7.4 Decision Tree 4

It is very interesting to make a note that the five rules stated above are independent of each other in the sense that they can be written in any order and still achieve the same decision in any given situation. Another interesting point is that the attribute ‘Temperature’ has no influence on the decision regarding whether to play or not to play.

Now, let the same example be considered with modified attributes, so that the attributes in the new example are all binary. In the modified structure, the dataset has nine attributes, namely, overcast, sunny, rainy, hot, mild, cool, high as well as normal humidity, weak as well as strong wind. With regard to new attributes, the decision rules can be stated as follows:

"If outlook=overcast, then play = yes", likewise, "if temperature = hot, then play = no", likewise, "if wind = strong, then play = no", likewise, "if outlook = rainy, then play = yes", likewise, "if humidity = normal, then play = yes", and likewise, play = no.

The sequential decision rule can be represented by a binary decisions tree. It may be noted that the optimal binary decision tree is the smallest decision tree, which is consistent with the training data. The binary decision tree generated by the above decision rule is as follows:

Figure 7.5 Decision Tree 5
It may be noted that this decision tree has six leaf nodes, as opposed to the decision tree generated by the classical algorithm that has five leaf nodes. It is interesting to note that every branching in the binary decision tree has one leaf node and one decision node.

It should be noted that the decision rule stated above is a genuinely sequential decision rule because it is taking out a subset of the given data and specifying the output for the relevant subset. There is no independence among the different parts of the decision rule and the different parts of the rule are not interchangeable. The rule must be applied in the same order in which it is presented. This is a genuinely recursive algorithm. Also, since all the attributes are binary, any measure of information can be used to measure the information gain, or the gain ratio at every split. It is interesting to note that the attribute ‘Temperature’ is involved in the new decision rule and is, therefore, not redundant, as was the case in the given problem.

The second example of the restaurant data, where the problem is whether or not the customer will wait for a table will also be presented to illustrate the proposed method. In this case, the final sequential decision rule is as follows:

If patrons = none, then will wait = no;
else if patrons = some, then will wait = yes;
else if Est > 60, the will wait = no.
else if Type = Italian, then will wait = no,
else if Type = Berger, then will wait = yes,
else if Fri = yes, then will wait = yes,
else will wait = no.

The decision tree for the will wait problem according to the conventional decision making processes is given below:

Figure 7.6 Decision Tree 6

Figure 7.7 Decision Tree 7

Note that the decision tree depicted above has 131 nodes. Now consider the decision tree induced by the modified algorithm, where multi-value attributes are converted to binary attributes. This decision tree is as follows:
It may be noted at this stage that the decision tree depicted above has only seven leaf nodes, showing that the modified algorithm can generate a more compact tree than the standard algorithm.

7.5 Decision Table in Canonical Form

Decision tables are usually written in the same order that the original dataset has. In this section, it is proposed that the decision table for a sequential decision rule can be written in a canonical form. It is further claimed that the decision table in the canonical form is simple to develop, easy to understand, and straightforward for use in future instances where the output is to be predicted for the specified input attribute values.

The canonical form of the decision table is obtained as follows. Suppose $A_1$ is the attribute at the root that defines the first leaf node. Let $n_1$ be the number of cases in the first leaf node. Then, the first column specifies the attribute $A_1$ and the first $n_1$ rows of the table consist of the $n_1$ cases where the attribute $A_1$ is present. The common value of the output attribute corresponding to all these cases is written in the first $n_1$ rows of the last column of the table, which corresponds to the output attribute. Let $A_2$ be the second attribute that defines the second leaf node and let $n_2$ be the number of cases in this node. Then the second column of the decision table specifies the attributes $A_2$ and the next $n_2$ rows contain the $n_2$ cases corresponding to the attribute $A_2$. The last column contains the common value of the output attribute in the same $n_2$ rows. This structure is repeated till all the cases of the given dataset are classified. The table can be read easily. The decision for the first $n_1$ cases is unconditional, while the decision for the next $n_2$ cases conditional on the event that no decision has been taken for these cases, so far. Similarly it happens for next $n_3$ cases and so on.
Consider the Play Tennis example. The six leaf nodes correspond to the attributes “Overcast”, “Hot (temperature)”, “Strong(wind)”, “Rainy”, “Normal Humidity” and “High Humidity”. The decision table in the canonical form is written as follows:

<table>
<thead>
<tr>
<th>X3</th>
<th>overcast</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>X7</td>
<td>overcast</td>
<td>Yes</td>
</tr>
<tr>
<td>X12</td>
<td>overcast</td>
<td>Yes</td>
</tr>
<tr>
<td>X13</td>
<td>overcast</td>
<td>Yes</td>
</tr>
<tr>
<td>X1</td>
<td>Hot</td>
<td>No</td>
</tr>
<tr>
<td>X2</td>
<td>Hot</td>
<td>No</td>
</tr>
<tr>
<td>X6</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X14</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>X4</td>
<td>Rainy</td>
<td>Yes</td>
</tr>
<tr>
<td>X5</td>
<td>Rainy</td>
<td>Yes</td>
</tr>
<tr>
<td>X10</td>
<td>Rainy</td>
<td>Yes</td>
</tr>
<tr>
<td>X9</td>
<td>Normal</td>
<td>Yes</td>
</tr>
<tr>
<td>X11</td>
<td>Normal</td>
<td>Yes</td>
</tr>
<tr>
<td>X8</td>
<td>High</td>
<td>No</td>
</tr>
</tbody>
</table>

Similarly, the decision table for will wait problem can be written the canonical form as follows:

<table>
<thead>
<tr>
<th>X7</th>
<th>None</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>X11</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>X1</td>
<td>Some</td>
<td>Yes</td>
</tr>
<tr>
<td>X3</td>
<td>Some</td>
<td>Yes</td>
</tr>
<tr>
<td>X6</td>
<td>Some</td>
<td>Yes</td>
</tr>
<tr>
<td>X8</td>
<td>Some</td>
<td>Yes</td>
</tr>
<tr>
<td>X5</td>
<td>Est &gt; 60</td>
<td>No</td>
</tr>
<tr>
<td>X9</td>
<td>Est &gt; 60</td>
<td>No</td>
</tr>
<tr>
<td>X10</td>
<td>Italian</td>
<td>No</td>
</tr>
<tr>
<td>X12</td>
<td>Berger</td>
<td>Yes</td>
</tr>
<tr>
<td>X4</td>
<td>Fri</td>
<td>Yes</td>
</tr>
</tbody>
</table>
The two example given above should be enough to show how converting all attributes to binary forms and then developing a sequential decision rule makes it easy to develop a decision tree and also allows the corresponding decision table to be presented in the canonical form.

The main reason why the modified algorithm is presented through two examples is that it is very difficult to describe it only theoretically and it is also equally difficult to understand in its abstract form. It is rather unfortunate that the literature does not provide large number of examples that can be used for making comparisons between the standard algorithm and modified algorithm. Another shortcoming of the use of the available examples is a small size of the training data. If the training dataset is very small, the induction algorithm is more likely to generate an extremely simple decision tree. Such a tree, however, is not sufficiently strong to generalize well to accurately predict the output attribute for unknown instances in new datasets.

7.6 Popular Decision Tree Algorithms: A Survey

1. Classification and Regression Trees (CART):

   It is one of the most efficient classification algorithms that have revolutionized the field of analytics and opened doors of the new age of data mining. CART is equally popular among technical and non-technical users due to its elegance, accuracy and ease of use. CART very efficiently brings out important data relationships that could remain undetected using other analytical tools. CART was developed by Breiman, Friedaman, Olshen, and Stone in 1984. The representation for the CART model is a binary tree. CART can handle categorical and numerical data equally efficiently. CART uses Gini index as the splitting Criterion.

2. Iterative Dichotomiser 3(ID3):

   In decision tree learning, ID3 is an algorithm that was invented by Ross Quinlan in 1986. It is used for generating a decision tree from a dataset. ID3 is typically used in the domains of machine learning and natural language processing. ID3 has the drawback that it cannot always generate an optimal solution in the sense that it can get stuck at local optima. ID3 can result in an over fitted model and is harder to use on continuous data.

3. C4.5 Algorithm:

   This algorithm is an extension of ID3 and was proposed by Quinlan in 1993. It is often called a statistical classifier because it can be used for classification. C4.5 uses information gain as the splitting criterion. C4.5 can easily handle missing values because the calculation of
information gain does not use missing values. In comparison with CART that always generates binary trees, C4.5 allows two or more outcomes. C4.5 uses singlepass confidence intervals for the binomial distribution and as a result, does not prune trees after growing them.

4. See5/C5.0:

C5.0 is an implementation of decision tree algorithm that can analyze large datasets containing thousands to millions of records. It can handle up to hundreds of attributes that can be in categorical, numerical, time or date formats. It is capable of parallel computing.

5. FACT, QUEST, and CRUISE:

A Group of researchers led by Prof. Wei-yin Loh has created a family of some interesting decision tree algorithms. The three prominent members of the family are Fast Algorithms for Classification Trees (FACT) Loh and Vanichsetakul (1988), Quick, Unbiased, Efficient, Statistical Tree (QUEST) Loh and Shin (1997), and Classification Rule with Unbiased Interaction Selection and Estimation(CRUISE) Kim and Loh (2001 and 2003). These methods are known as statistical trees as they are strongly depend upon the statistical tools used in the building and refinement of trees. These algorithms have univariate as well as multivariate version, though their univariate forms are used more often.

FACT has been designed to work with datasets that contain numerical features, but the authors have described the method for converting representative attributes into continuous variables before the implementation of the core algorithm.

QUEST was developed as a prominent development over FACT. A general outline and form of the algorithm has been the identical in the sense that feature selection is isolated from determining the split level, and symbolic features are converted to numerical variables in a similar manner. The major point of differences is that the split is made using a quadratic discriminant functions instead of a linear discriminant function (as in the case of FACT) and that the resulting tree is binary (unlike FACT, where multiple classifications are allowed).

CRUISE is a continuation of FACT and QUEST. It has attained a prominent improvement in the direction of making unbiased feature selection. The general strategy of CRUISE is the same as that of its two predecessors, but some major differences exist. CRUISE generates multi-split trees, and it uses pairs of features to select the features for splitting.

6. C Tree:
Many solutions have been obtained through the unbiased feature selection in constructing decision trees. The line of approach of Hothorn et al. (2004) and Zeileis et al. (2008) towards a conditional inference construction is one of the most interesting results of this pursuit. This approach has led to the development of the C Tree algorithm. C Tree has the same separation between feature selection and split finding as FACT, but uses non-parametric permutation tests for determining the best split level for numerical features.

7. LMDT:

The LMDT algorithm (Utgoff and Brodley, 1991) is one of the first and popular approaches to developing decision trees using linear combinations of features. LMDT does not approach the problem of feature selection by considering one feature after another, but constructs a linear machine for each node which is necessary to break. A linear machine is a collection of k, linear discriminating functions used to categorize data objects in the k objective classes.

8. Oblique Classifier 1 (OC 1):

Oblique Classifier 1 (OC 1), Murthy et al. (1993) is a method of constructing decision trees through a search for optimal hyper plane that separates various classes of items. This investigation depends on a heuristic in order to obtain local minima and non-deterministic approaches to pick up the local minima to get better solutions. A single separating hyper plane is determined at each and every node of a decision tree, thus resulting is a binary tree. In a way similar to LMDT, the hyper plane is defined by the feature vector of dimension m+1, where m is the dimension of the feature space. OC 1 uses what is known as the perturbation method that adjusts a selected coefficient in the hyper plane so as to enlarge a measure of contamination of the break up by the hyper plane.

9. L Tree, Q Tree and Lg Tree:

A mixture of three concepts: 1) the divide and conquer methodology of decision trees, 2) linear discriminant analysis of multivariate statistics, and 3) constructive induction have resulted in probabilistic linear trees L Tree (Gama, 1997). An L Tree performs a split at each node in the independent steps as given below:

- Linear combinations of features to construct new attributes.
- A technique of univariate decision tree construction to determine the best split.
Some significant differences between the new approach and other methods were caused by the scheme described above and other ideas introduced in L Tree induction. The following are examples of such significant differences.

- The same classification tree can have different numbers of features describing data between nodes.
- When new attributes are created, they can be used in the same branch at descendant nodes.
- An object is classified using probabilities estimated through an examination of data distributions in excess of the entire route from the root node leading to the proper leaf node.

10. DT-SE, DT-SEP and DT-SEPIR:

John (1995) has developed the plan of by means of smooth criteria for prediction the split quality and iterative refiltering of regularization of decision trees. Since soft multidimensional criteria are incompatible with symbolic characteristics, methods in the DT-SE family require preprocessing of the data to remove symbols that do not have a sensible order. All scattered categorical attributes are changed into appropriate 0-1 indicator variables in order to avoid accidental data which is being introduced or existing information implied by arbitrary sequence of images. Binary splits, therefore, are natural for these family trees, even though it is easily possible for applying the same optimization procedures to analyzing multi way splits.

11. Linear Decision Trees (LDT):

Yildiz and Alpaydin (2000) has developed an algorithm called linear decision trees (LDT) after performing an analysis of the following aspects of decision tree induction algorithms:

- Univariate or multivariate : Types of node
- Splitting a node into two or more than two subnodes: Branching factor
- Grouping classes into two super classes for twofold trees
- Measures undertaken for split characteristic
- Minimization procedures used to find most excellent splits

The linear decision trees algorithm that results performs linear discriminant analysis at every node and creates binary trees.

12. Multivariate Adaptive Regression Splines (MARS):

MARS, developed and introduced by Friedman in 1991, MARS works by finding optimal
transformation and interactions of variables. Non-linear relationships are described in two way is MARS. First, every variable has a natural cut-off because the influence of the variable may change over the range of its values. Second, different variables may have interaction effects, where the effects of two individual variables may not be significant, but the product of the same two variables may have significant effect.

MARS achieves the best prediction model in two steps. First, a collection a Basis Functions (BFs) is built by transforming independent variables, taking into account any nonlinearities and interactions that may be present in the data. Second, a least-squares model is estimated using the Basis Functions as independent variables, which explores interactions between different independent variables.

13. SMART:

SMART is a classification and regression algorithm and is used most commonly in batch mode. It is very slow in the training phase, but quick in the classification phase. A non-statistician would find the output difficult to comprehend, but a graphical front-end can improve its interpretability. There are some difficulties in deciding how many terms should be included in the model. SMART has a major advantage that it’s accepts a cost matrix and hence is more suitable when costs are important.

14. CASTLE:

CASTLE builds a probabilistic model for the attributes in the empirical data, and is a Bayesian algorithm. The polytree algorithm builds a model for entire data without any regard for the class of the object and hence CASTLE is used. CASTLE may be used in interactive and batch modes. The outcome of CASTLE is in the form of a polytree and provides a graphical representation of the possible relationships among attributes and classes. A standard test can be used to develop a goodness-of-fit measure into CASTLE.

15. AC$^2$ and New ID:

AC$^2$ and New ID are descendants of ID3 and their performance has been empirically observed to be similar. The AC$^2$ package has interactive graphical interface and incorporates certain forms of hierarchical structures in the dataset. It can handle logical and numerical attributes. Interactive mode and facility of editing data under its uses interface are two main
features of \( AC^2 \). In order to express hierarchical attributes, when present, \( AC^2 \) uses a format different from the usual format. \( AC^2 \) has the ability to deal with unknown values as well as multi-valued attributes. Decision trees produced by \( AC^2 \) can be very large in comparison with other tree algorithms.

New ID is based on the original ID3. However, it is identical to CN2 in the interface and command system. It can be used in interactive as well as in batch mode, even though its native mode is interactive. New ID can handle logical and numerical attributes. It possesses a post-pruning facility which deals with noise. It can also deal with unknown value. New ID does not classify all data elements and, as a result, some records may not be classified. New ID outputs a chaotic matrix which has an additional row and column for unclassified data elements. It does not involve a cost matrix.

16. Cal5:

Cal5 uses statistical methods to build decision trees for numerical attributes. Symbolic values have to be converted to numerical values to be converted to numerical values. Cal5 is menu-driven and has numerous menus to guide the user in completing operation, and is therefore easy set up and run. However, it has several parameters, whose interpretation is not easy for a new user. Also, different parameter settings lead to very different results, and hence tuning of parameters in very important.

17. CN2:

CN2 is a rule-based algorithm for recursive partitioning. It has two variants, known as ordered and unordered. Among the two variants, the unordered one appears to produce better results. Its performance is almost equivalent to that of decision trees. However, CN2 performs badly when costs are involved. Though interactive mode is its natural mode, CN2 may be used in interactive and batch modes. CN2 differs from other algorithms in that it requires declarations that define the type of ranges of values for every attribute. CN2 is very easy to develop and to run in the interactive mode as its operations are totally menu-driven. The only important parameter that may have a significant effect on the results is rule types.

18. IT rule:

IT rule is really an exploratory tool, designed for extracting isolated interesting rules (facts) from a dataset. In this sense, it is not a classification algorithm. Also, the rules extracted
from a dataset are not expected to cover all data elements. In a way, it can be said that IT rule does not look for the ‘best set of rules’. On the contrary, it looks for a set of ‘best rules’, where every rule is simple in structure in the sense that the rules have high information content. The search for rules is time-consuming because it is exhaustive, subjective to the limitations mentioned above. The algorithm has no provision of using a cost matrix, but it is not difficult to incorporate costs.

19. ALLOC 80:

This algorithm was not intended for large datasets because it often fails on large datasets, and does not have adequate diagnostics. It can work with numerical and logical attributes, and appears to be superior to other statistical algorithms in this regard. The cross-validation methods used for selection of parameters are cumbersome, although they should work in principle. Choosing good smoothing parameters is an outstanding problem. The program uses multiplicative Kernel and when some of the attributes are highly correlated, it is no more flexible.

ALLOC80 has, to some extent, a lower error-rate than that of k-NN. It also uses less storage but takes almost double time in training and testing.

20. Genetically Evolved Decision Trees (GATree):

GATree uses genetic algorithms to evolve decision trees. It does not use binary string, but uses a binary tree structure as a natural representation of the problem. Various parameters like the number of generations, the size of population, and probabilities of crossover and mutation can be set by the user. Genetic algorithms are most often related with decision trees as a preprocessor for selecting features.

21. ALICE d’ISoft:

ALICE d’ISoft, a software for building decision trees, is a powerful and attractive tool that facilitates creation of segmentation models. It allows the user to explore data interactively and directly online. It is commercial software and works on the Microsoft windows platform.

22. CHAID, AID, MAID and THAID:

Researchers in the field of applied statistics have developed various procedures to generate decision trees based on the principle of automatic-interaction-detection (AID). Chronologically and developmentally, the sequence goes as follows:

AID (1971)
Chi-square Automatic Interaction Detection (CHAID) was originally designed and developed only to carry out nominal attributes. It finds the pair of values of each and every attribute which differs, to some extent, with respect to the target variable. Importance of the difference is obtained by means of the p-value from a statistical test. A statistical test designed depends on the type of a target attribute. If the selected pair does not differ significantly according to a certain level of significance, the pair is merged and a new search is initiated to find another couple to be combined. The said process is continued until new pairs are obtained that can be merged.

After the merging operation is completed, the attribute used for splitting the node currently in hand is chosen so that each child node is homogeneous with respect to the selected attributes. It should be noted that the current node is not split if the accepted input attribute is insignificant at a specified level of significance. The procedure stops as soon as any one of the conditions given below is satisfied:

1. A tree reaches its maximum depth.
2. No node splits further as a result of the minimum size of a node to be a parent node is reached.
3. The minimum size for a node to be a child is reached.

CHAID does not provide for pruning.

23. T1:

T1 is a one-level decision tree algorithm which makes use of one attribute only to classify instances (Holte, 1993). Missing values are treated as a special value, and become one more value of the concerned attribute.

24. PUBLIC (Rastogi and Shim, 2000):

PUBLIC integrates the increasing and pruning operations by using machine design learning cost to reduce the computational difficulty.