A decision tree classifier is a hierarchical structure where at each level a test is applied to one or more attribute values that may have one of two outcomes as shown in figure 3.1.

The outcome may be a leaf, which allocates a class, or a decision node, which specifies a further test on the attribute values and forms a branch or sub-tree of the tree. Classification is performed by moving down the tree until a leaf is reached [16]. The method for constructing a decision tree as summarized by Quinlan (1993) is as follows:

- If there are $k$ classes denoted $\{C_1, C_2, ..., C_k\}$, and a training set, $T$, then
- If $T$ contains one or more objects which all belong to a single class $C_j$, then the decision tree is a leaf identifying class $C_j$.
- If $T$ contains no objects, the decision tree is a leaf determined from information other than $T$. If $T$ contains objects that belong to a mixture of classes, then a test is chosen, based on a single attribute that has one or more mutually exclusive outcomes $\{O_1, O_2, ..., O_n\}$. $T$ is partitioned into subsets $T_1, T_2, ..., T_n$, where $T_i$ contains all the objects in $T$ that have outcome $O_i$ of the chosen test. The same method is applied recursively to each subset of training objects to build the decision tree.
Decision tree classifiers differ in the ways they partition the training sample into subsets and thus form sub-trees. That is, they differ in their criteria for evaluating splits into subsets. The See5 or C4.5 induction algorithm uses information theory (Shannon, 1949) to evaluate splits. CART uses Gini Index to split the training samples (Breiman, 1984) and some methods use Chi-Square measure.

Many studies have been done comparing See5 decision tree algorithm with other classifiers and found that See5 based on the Information theory is more accurate and gives reliable results. The other advantage of See5 algorithm is that it can convert decision tree into corresponding classification rules. Rules are more comprehensive, easy to understand and easy to implement. Therefore the working of the decision tree explained in this chapter is in context to See5 algorithm.

### 3.1 Strengths of Decision Tree Methods

- **Ability to Generate Understandable Rules:** The ability of decision trees to generate rules that can be translated into comprehensible English or SQL is the greatest strength of this technique.

- **Ease of Calculation at Classification Time:** A decision tree can take many forms, in practice, the algorithms used to produce decision trees generally yield trees with a low branching factor and simple tests at each node. Typical tests include numeric comparisons, set membership, and simple conjunctions. When implemented on a computer, these tests translate into simple Boolean and integer operations that are fast and computationally inexpensive.

- **Ability to Handle Both Continuous and Categorical Variables:** Decision-tree methods are equally adept at handling continuous and categorical variables. Categorical variables, which pose problems for neural networks and statistical techniques, come ready-made with their own splitting criteria: one branch for each category. Continuous variables are equally easy to split by picking a number somewhere in their range of values.
✓ **Ability to Clearly Indicate Best Fields:** Decision-tree building algorithms put the field that does the best job of splitting the training records at the root node of the tree.

### 3.2 Weaknesses of Decision Tree Methods

Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous variable such as remote sensing data. Decision trees are also problematic for time-series data unless a lot of effort is put into presenting the data in such a way that trends and sequential patterns are made visible.

- **Computationally Expensive to Train:** The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be computationally expensive since many candidate sub-trees must be formed and compared.

- **Trouble with Non-Rectangular Regions:** Most decision-tree algorithms only examine a single field at a time. This leads to rectangular classification boxes that may not correspond well with the actual distribution of records in the decision space.
3.3 Criteria for Evaluating Splits – Information Theory

As mentioned above that different decision tree algorithms have different criteria for splitting the training samples. See5 uses criteria, which is based on the information theory. It defines a statistical property called information gain that measures how well a given attribute separates the training samples according to their target classification. See5 uses this information gain measure to select among the candidate attributes at each step while growing the tree.

For any subset $S$ of $X$, where $X$ is the population, let $freq(j_i, S)$ be the number of objects in $S$, which belongs to class $i$. Then consider the ‘message’ that a randomly selected object belongs to class $j_i$. The ‘message’ has probability $freq (j_i, S) / |S|$, where $|S|$ is the total number of objects in subset $S$. The information conveyed by the message (in bits) is given by $-\log_2 freq (j_i, S) / |S|$. Summing over the classes gives the expected information (in bits) from such a message:

$$\text{Info}(s) = -\log_2 \left(\frac{\text{Freq} (C_j, S)}{|s|}\right) \quad 3.1$$

When applied to a set of training objects, $\text{Info}(T)$ gives the average amount of information needed to identify the object of a class in $T$. This amount is also known as the entropy of the set $T$. Consider a similar measurement after $T$ has been partitioned in accordance with the $n$ outcomes of a test $X$. The expected information requirement can be found as a weighted sum over the subsets $\{T_i\}$:

$$\text{Info}_X(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \text{Info}(T_i) \quad 3.2$$

The quantity

$$\text{gain}(X) = \text{info}(T) - \text{info} X(T) \quad 3.3$$
Measures the information that is gained by partitioning \( T \) in accordance with the test \( X \). The gain criterion selects a test to maximize this information gain. The gain criterion has one significant disadvantage in that it is biased towards tests with many outcomes. The gain ratio criterion (Quinlan, 1993) was developed to avoid this bias [11]. The information generated by dividing \( T \) into \( n \) subsets is given by

\[
\text{Split Info}(X) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \log_2 \left( \frac{|T_i|}{|T|} \right)
\]

The proportion of information generated by the split that is useful for classification is

\[
\text{Gain ration} = \frac{\text{gain}(X)}{\text{split info}(X)}
\]

If the split is near trivial, split information will be small and this ratio will be unstable. Hence, the gain ratio criterion selects a test to maximize the gain ratio subject to the constraint that the information gain is large.

### 3.4 Tests on Continuous Attributes

The main crux of a decision tree lies in deciding appropriate attributes and the corresponding threshold for each node of the tree. The algorithm for finding appropriate thresholds for continuous attributes is explained as follows:

In the induction of decision trees from continuous-valued data, a suitable threshold \( T \), which discretizes the continuous attribute \( A \) into two intervals \( A_1 = (\min (A), T) \) and \( A_2 = (T, \max (A)) \), is determined based on the classification information gain generated by the corresponding discretization. Given a threshold, the test \( A = T \) is assigned to the left branch of the decision node while \( A > T \) is assigned to the right branch.

Assuming there are to select an attribute for a node having a set \( S \) of \( N \) examples, these examples are sorted according to the values of the continuous attribute \( A \); and an ordered
sequence of distinct values \( a_1, a_2, \ldots a_N \) is formed. Every pair of adjacent data points suggest a potential threshold \( T = (a_i + a_{i+1}) / 2 \) to create a cut point and generate a corresponding partition of \( A \). Fayyad (1992) had proved that only the class boundary points could be the cut points to obtain the maximum information in classification, which implies if \( a_i \) and \( a_{i+1} \) belong to the same class, a cut point between them cannot lead to a partition that has maximum information gain. Therefore, generate a smaller set of candidate cut points from the class boundary points. Let there be \( k \) classes \( c_1, c_2, \ldots c_k \), and let \( p(c_j, S) \) be the proportion of examples in \( S \) that belong to class \( c_j \). The residual uncertainty in classification is expressed as the class entropy:

\[
E(S) = - \sum_{j=1}^{k} p(C_j, S) \log(p(C_j, S)) \tag{3.6}
\]

After the set of example \( S \) is partitioned into two subsets \( S_1 \) and \( S_2 \) by a threshold \( T \), the class information entropy is expressed as the weighted average of their resulting class entropy:

\[
E(A, T; S) = N_1/N \cdot E(S_1) + N_2/N \cdot E(S_2) \tag{3.7}
\]

\[
E(S_1) = - \sum_{j=1}^{k} p(C_j, S_1) \log(p(C_j, S_1)) \tag{3.8}
\]

\[
E(S_2) = - \sum_{j=1}^{k} p(C_j, S_2) \log(p(C_j, S_2)) \tag{3.9}
\]

Where, \( N_1 = |S_1|, N_2 = |S_2|, \) and \( N = |S| \) and are the number of examples in \( S_1, S_2, \) and \( S \), and \( p(c_j, S_1) \) and \( p(c_j, S_2) \) are the proportion of examples of class \( c_j \) in \( S_1 \) and \( S_2 \) respectively.

The cut point for which \( E(A, T_A; S) \) is minimal among all the candidate cut points of attribute \( A \) is used; and the attribute \( A_j \), for which the \( E(A_j, T_A^j; S) \) is minimum, or the information gain \( E(S) - E(A_j, T_A^j; S) \) is maximum, will be then selected to generate two child nodes. In each child node, discretization and attribute selection are performed again based on the partitioned examples. The above process is repeated recursively until a stopping criterion is matched.
3.5 Data Preparation for Decision Tree

The data for our experiment was prepared by the KDD Cup 1999 intrusion detection evaluation program by University of California, Irvine [3].

The data set has 41 attributes and one class label. The actual total data full data set contains 743M uncompressed with lacks records. Actual data set contains 22 attack types mainly those attacks divides four groups [3]

- Denial of Service (DoS)
- Unauthorized Access from A Remote Machine (R2L)
- Unauthorized Access to Local Super User (U2R)
- Surveillance And Other Probing (Probing)

3.5.1 Training and Testing Dataset

Stratified random sampling methods were used to collect separate training and test data sets from actual KDD Cup data set. The data collected by random sampling were divided into two subsets, one of which was used for training and the other for testing the classifiers. Three files were created [3]:

- **Name file:** The first essential file is the names file (shown Figure 3.2) that describes the attributes and classes. All the attributes (attributes from data set) shown Figure 3.3 with data type and attacks classes were defined in this file. It serves as a meta-data file for the decision tree training dataset. Shows at below

Mainly attributes data types are continuous and symbolic. The following Figure shows all attributes list and data types. Symbolic means having some fixed values for attribute that means example attribute protocol_type is symbolic it is having only tcp or icmp or udp not possible to allow other values. Here some attributes are Booleans
but that attributes are also converted to symbolic, for example Is_host_login and Is_guest_login.
Training File: It is a notepad file, which provides information on the training cases from which classification rules, or knowledge is extracted. One by one training data were selected and their corresponding values for all the attributes were written in this file. Classes for corresponding record were written at the end of the record.

For example:

0, tcp, http, SF, 54540, 8314, 0, 0, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 9, 0.00, 0.00, 0.50, 0.11, 1.00, 0.00, 0.22, 255, 229, 0.90, 0.01, 0.00, 0.00, 0.00, 0.00, 0.01, 0.00 – 41 attributes values, dos -- class label
### Figure 3.4: Training File

- **Test file**: The third file consists of test cases on which the classifier can be evaluated. The test and the training files are almost similar only cases inside are different.
Figure 3.5: Test Data

Using the training set samples created above, the classifier was built in the form of a decision tree. Figure 3.6 shows the decision tree generated using See5 algorithm.

Read 175 cases (41 attributes) from nusp.data

Decision tree:

num_file_creations > 0: u2r (30)
num_file_creations (= 0):
    ...dst_host_same_src_port_rate > 0.49: Probing (30)
    dst_host_same_src_port_rate (= 0.49):
    ...num_compromised > 0: dos (48)
    num_compromised (= 0):
    ...src_bytes > 51: normal (38/1)
    src_bytes (= 51):
    ...srv_count = 1: r2l (30)
    srv_count > 1: normal (3)

Figure 3.6: Output of Decision Tree
To investigate the affect of the size of training set on the accuracy of the classifier, different training samples of different sizes were prepared (for example 5000, 20000, 30000 samples etc). Each classifier generated from particular training sample dataset was then tested on unseen test cases to know how the accuracy of the classifier is related to the size of the training set. For every training set the classifier’s accuracy on training and test cases was checked and graphs were plotted to study the patterns.

### 3.6 Decision Trees to Rules

Though pruning makes a decision tree both simpler and more accurate, decision trees can still be cumbersome and complex. In fact, it does not matter whether it is pruned or not pruned, for decision trees are always difficult to understand.

To simplify a decision tree and convert them into rules, which are easier to understand and to implement, every path from the root to a leaf is converted to an initial rule by regarding all the test conditions appearing in the path as the conjunctive rule antecedents, while regarding the class label held by the leaf as the rule consequence (Quinlan, 1993). From figure 3.7, derive rules from the decision tree, such as:

**Rule 1.** (48, lift 3.6)

\[
\text{num\_compromised} \quad > \quad 0
\]

\[
\rightarrow \quad \text{class\_dos} \quad [0.980]
\]

**Rule 2.** (30, lift 5.8)

\[
\text{dst\_host\_same\_src\_port\_rate} \quad > \quad 0.49
\]

\[
\rightarrow \quad \text{class\_Probing} \quad [0.969]
\]

**Figure 3.7 Derived Rules Set from Decision Tree**

After that, each initial rule is generalized by removing antecedents that do not seem helpful for distinguishing a specific class from other classes, which is performed by a pessimistic estimate of the accuracy of the rule. In detail, the accuracy of the initial rule
and that of its variant where an antecedent is removed are estimated. If the latter is not worse than the former then the initial rule is replaced by the variant of it.

It is worth noting that usually there are several rule antecedents that could be removed. In such cases, See5 Rule carries out a greedy elimination, that is, the removal of the antecedent that produces the lowest pessimistic error rate of the generalized rule is kept, and such kind of removal is repeatedly performed until the rule could not be generalized further [10].

After all the initial rules are generalized, they are grouped into rule sets corresponding to the classes respectively. All rule sets are polished with the help of the Minimum Description Length (MDL) Principle so that rules that do not contribute to the accuracy of a rule set are removed (Quinlan, 1993).

Then, the rule sets are sorted according to the ascending order of their false positive error rates. Finally, a default rule is created for dealing with instances that are not covered by any of the generated rules. The default rule has no antecedent and its consequence is the class that contains the most training instances not covered by any rule.

### 3.6.1 Classification Rules

Decision trees can sometimes be quite difficult to understand. An important feature of See5 is its ability to generate classifiers called rule sets that consist of unordered collections of (relatively) simple “if then” rules, as shown in the Figure no.

Each rule consists of:

- A rule number, which serves only to identify the rule.

- Statistics \((n, \text{lift } x)\) or \((n/m, \text{lift } x)\) that summarizes the performance of the rule. Similarly to a leaf, \(n\) is the number of training cases covered by the rule and \(m\), if
it appears, shows how many of them do not belong to the class predicted by the rule. The rule's accuracy is estimated by the Laplace ratio \((n-m+1) / (n+2)\). The lift \(x\) is the result of dividing the rule's estimated accuracy by the relative frequency of the predicted class in the training set.

- One or more condition that must all be satisfied if the rule is to be applicable.
- A class predicted by the rule.
- A value between 0 and 1 that indicates the confidence with which this prediction is made.

For each corresponding path from the root of a decision tree to a leaf, a separate classification rule is generated. There are chances that some redundant and undesirable rules are also extracted, therefore care should be taken to remove such rules which do not contribute in improving the accuracy of the classification. The following strategy was adopted to filter out the desired rules out all extracted rules:

- If only one rule is activated, which means the attribute values match the conditions of this rule, let the final class be the same as stated by this rule.

- If several rules are activated, let the final class be the same as stated by the rule with the maximum confidence.

- If several rules are activated and the confidence values are the same, then let the final class be the same as stated by the rule with the maximum coverage of learning samples.

- If no rule is activated, then let the final class be the same as stated by the default class.
Rules:

Rule 1: (48, lift 3.6)
\[ \text{num\_compromised} > 0 \rightarrow \text{class dos} \quad [0.980] \]

Rule 2: (30, lift 5.8)
\[ \text{dst\_host\_same\_src\_port\_rate} > 0.49 \rightarrow \text{class Probing} \quad [0.969] \]

Rule 3: (30, lift 5.8)
\[ \begin{align*}
& \text{src\_bytes} \leq 51 \\
& \text{srv\_count} \leq 1 \\
& \text{dst\_host\_same\_src\_port\_rate} \leq 0.49 \\
\end{align*} \rightarrow \text{class r2l} \quad [0.969] \]

Rule 4: (30, lift 5.8)
\[ \text{num\_file\_creations} > 0 \rightarrow \text{class u2r} \quad [0.969] \]

Rule 5: (38/1, lift 4.3)
\[ \begin{align*}
& \text{src\_bytes} > 51 \\
& \text{num\_compromised} < 0 \\
& \text{num\_file\_creations} < 0 \\
\end{align*} \rightarrow \text{class normal} \quad [0.950] \]

Rule 6: (35/1, lift 4.2)
\[ \begin{align*}
& \text{num\_compromised} < 0 \\
& \text{srv\_count} > 1 \\
\end{align*} \rightarrow \text{class normal} \quad [0.946] \]

Default class: dos

\text{Figure 3.8: Rule Set}