Chapter – 2

Analysis of Data Mining Algorithms

With an enormous amount of data stored in databases and data warehouses, it is increasingly important to develop powerful tools for analysis of such data and mining interesting knowledge from it. Data mining is a process of inferring knowledge from such huge data. Data Mining has three major components Clustering or Classification, Association Rules and Sequence Analysis.

By simple definition, in classification/clustering we analyze a set of data and generate a set of grouping rules which can be used to classify future data. For example, one may classify diseases and provide the symptoms which describe each class or subclass. This has much in common with traditional work in statistics and machine learning. However, there are important new issues which arise because of the sheer size of the data. One of the important problems in data mining is the Classification-rule learning which involves finding rules that partition given data into predefined classes. In the data mining domain where millions of records and a large number of attributes are involved, the execution time of existing algorithms can become prohibitive, particularly in interactive applications.

An association rule is a rule which implies certain association relationships among a set of objects in a database. In this process we discover a set of association rules at multiple levels of abstraction from the relevant set(s) of data in a database. For example, one may discover a set of symptoms often occurring together with certain kinds of diseases and further study the reasons behind them. Since finding interesting association rules in databases may disclose some useful patterns for decision support, selective marketing, financial forecast, medical diagnosis, and many other applications, it has attracted a lot of attention in recent data mining research. Mining
association rules may require iterative scanning of large transaction or relational databases which is quite costly in processing. Therefore, efficient mining of association rules in transaction and/or relational databases has been studied substantially. In sequential Analysis, we seek to discover patterns that occur in sequence. This deal with data that appear in separate transactions (as opposed to data that appear in the same transaction in the case of association). For e.g.: If a shopper buys item A in the first week of the month, then s/he buys item B in the second week etc.

There are many algorithms proposed that try to address the above aspects of data mining. Compiling a list of all algorithms suggested/used for these problems is an arduous task. I have thus limited the focus of this research to list only some of the algorithms that have had better success than the others.

2.1 Classification Algorithms

In Data classification one develops a description or model for each class in a database, based on the features present in a set of class-labeled training data. There have been many data classification methods, including decision-tree methods, such as C4.5, statistical methods, neural networks, rough sets, database-oriented methods etc.

2.1.1 Data Classification methods

- **Statistical Algorithms** Statistical analysis systems such as SAS and SPSS have been used by analysts to detect unusual patterns and explain patterns using statistical models such as linear models. Such systems have their place and will continue to be used.

- **Neural Networks** Artificial neural networks mimic the pattern-finding capacity of the human brain and hence some researchers have suggested applying Neural Network algorithms to pattern-mapping. Neural networks have been applied successfully in a few applications that involve classification.
- **Genetic algorithms** Optimization techniques that use processes such as genetic combination, mutation, and natural selection in a design based on the concepts of natural evolution.

- **Nearest neighbor method** A technique that classifies each record in a dataset based on a combination of the classes of the k record(s) most similar to it in a historical dataset. Sometimes called the k-nearest neighbor technique.

- **Rule induction** The extraction of useful *if-then* rules from data based on statistical significance.

- **Data visualization** The visual interpretation of complex relationships in multidimensional data.

### 2.1.2 Data Abstraction

Many existing algorithms suggest abstracting the test data before classifying it into various classes. There are several alternatives for doing abstraction before classification: A data set can be generalized to either a minimally generalized abstraction level, an intermediate abstraction level, or a rather high abstraction level. Too low an abstraction level may result in scattered classes, bushy classification trees, and difficulty at concise semantic interpretation; whereas too high a level may result in the loss of classification accuracy.

### 2.1.3 Classification-rule learning

Classification-rule learning involves finding rules or decision trees that partition given data into predefined classes. For any realistic problem domain of the classification-rule learning, the set of possible decision trees is too large to be searched exhaustively. In fact, the computational complexity of finding an optimal classification decision tree is NP hard.

Most of the existing induction-based algorithms use Hunt's method as the basic algorithm. Here is a recursive description of Hunt's method for constructing a decision tree from a set T of training cases with classes denoted \(\{C_1, C_2, \ldots, C_k\}\).
**Case 1** T contains one or more cases, all belonging to a single class $C_j$: The decision tree for $T$ is a leaf identifying class $C_j$.

**Case 2** T contains no cases: The decision tree for $T$ is a leaf, but the class to be associated with the leaf must be determined from information other than $T$.

**Case 3** T contains cases that belong to a mixture of classes: A test is chosen, based on a single attribute, that has one or more mutually exclusive outcomes $\{O_1, O_2, \ldots, O_n\}$. $T$ is partitioned into subsets $T_1, T_2, \ldots, T_n$, where $T_i$ contains all the cases in $T$ that have outcome $O_i$ of the chosen test. The decision tree for $T$ consists of a decision node identifying the test, and one branch for each possible outcome. The same tree building machinery is applied recursively to each subset of training cases.

<table>
<thead>
<tr>
<th>outlook</th>
<th>Temp(F)</th>
<th>Humidity(%)</th>
<th>Windy?</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>75</td>
<td>70</td>
<td>true</td>
<td>Play</td>
</tr>
<tr>
<td>sunny</td>
<td>80</td>
<td>90</td>
<td>true</td>
<td>Don't Play</td>
</tr>
<tr>
<td>sunny</td>
<td>85</td>
<td>85</td>
<td>false</td>
<td>Don't Play</td>
</tr>
<tr>
<td>sunny</td>
<td>72</td>
<td>95</td>
<td>false</td>
<td>Don't Play</td>
</tr>
<tr>
<td>sunny</td>
<td>69</td>
<td>70</td>
<td>false</td>
<td>Play</td>
</tr>
<tr>
<td>overcast</td>
<td>72</td>
<td>90</td>
<td>true</td>
<td>Play</td>
</tr>
<tr>
<td>overcast</td>
<td>83</td>
<td>78</td>
<td>false</td>
<td>Play</td>
</tr>
<tr>
<td>overcast</td>
<td>64</td>
<td>65</td>
<td>true</td>
<td>Play</td>
</tr>
<tr>
<td>overcast</td>
<td>81</td>
<td>75</td>
<td>false</td>
<td>Play</td>
</tr>
<tr>
<td>rain</td>
<td>71</td>
<td>80</td>
<td>true</td>
<td>Don't Play</td>
</tr>
<tr>
<td>rain</td>
<td>65</td>
<td>70</td>
<td>true</td>
<td>Don't Play</td>
</tr>
<tr>
<td>rain</td>
<td>75</td>
<td>80</td>
<td>false</td>
<td>Play</td>
</tr>
<tr>
<td>rain</td>
<td>68</td>
<td>80</td>
<td>false</td>
<td>Play</td>
</tr>
</tbody>
</table>
Table 2.1: A small training data set

<table>
<thead>
<tr>
<th>Attribute Value</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Play</td>
</tr>
<tr>
<td>sunny</td>
<td>2</td>
</tr>
<tr>
<td>overcast</td>
<td>4</td>
</tr>
<tr>
<td>rain</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2.2: Class Distribution Information of Attribute Outlook
<table>
<thead>
<tr>
<th>Attribute Value</th>
<th>Binary Test</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Play</td>
</tr>
<tr>
<td>65</td>
<td>&gt;</td>
<td>1</td>
</tr>
<tr>
<td>70</td>
<td>&gt;</td>
<td>8</td>
</tr>
<tr>
<td>75</td>
<td>&gt;</td>
<td>3</td>
</tr>
<tr>
<td>78</td>
<td>&gt;</td>
<td>6</td>
</tr>
<tr>
<td>80</td>
<td>&gt;</td>
<td>4</td>
</tr>
<tr>
<td>85</td>
<td>&gt;</td>
<td>5</td>
</tr>
<tr>
<td>90</td>
<td>&gt;</td>
<td>4</td>
</tr>
<tr>
<td>95</td>
<td>&gt;</td>
<td>5</td>
</tr>
<tr>
<td>96</td>
<td>&gt;</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.3: Class Distribution Information of Attribute Humidity

Table 2.1 shows a training data set with four data attributes and two classes. Figure 2.1 shows how the Hunt's method works with the training data set. In the case 3 of the Hunt's method, a test based on a single attribute is chosen for expanding the current node. The choice of an attribute is normally based on the entropy gains of the attributes. The entropy of an attribute is calculated from class distribution.
information. For a discrete attribute, class distribution information of each value of the attribute is required. Table 2.2 shows the class distribution information of data attribute Outlook.

For a continuous attribute, binary test involving all the distinct values of the attribute is considered. Table 2.3 shows the class distribution information of data attribute Humidity. Once the class distribution information of all the attributes are gathered, the entropy is calculated based on either information theory or Gini Index. One attribute with the most entropy gain is selected as a test for the node expansion.

2.1.4 ID3 algorithm

The ID3 algorithm (Quinlan86) is a decision tree building algorithm which determines the classification of objects by testing the values of their properties. It builds the tree in a top down fashion, starting from a set of objects and a specification of properties. At each node of the tree, a property is tested and the results used to partition the object set. This process is recursively done till the set in a given sub tree is homogeneous with respect to the classification criteria - in other words it contains objects belonging to the same category. This then becomes a leaf node. At each node, the property to test is chosen based on information theoretic criteria that seek to maximize information gain and minimize entropy. In simpler terms, that property is tested which divides the candidate set in the most homogeneous subsets.

2.1.5 C4.5 algorithm

This algorithm was proposed by Quinlan (1993). The C4.5 algorithm generates a classification-decision tree for the given data-set by recursive partitioning of data. The decision is grown using Depth-first strategy. The algorithm considers all the possible tests that can split the data set and selects a test that gives the best information gain. For each discrete attribute, one test with outcomes as many as the number of distinct values of the attribute is considered. For each continuous attribute, binary tests
involving every distinct values of the attribute are considered. In order to gather the entropy gain of all these binary tests efficiently, the training data set belonging to the node in consideration is sorted for the values of the continuous attribute and the entropy gains of the binary cut based on each distinct values are calculated in one scan of the sorted data. This process is repeated for each continuous attributes.

2.1.6 SLIQ algorithm

SLIQ (Supervised Learning In Quest) developed by IBM's Quest project team, is a decision tree classifier designed to classify large training data [1]. It uses a pre-sorting technique in the tree-growth phase. This helps avoid costly sorting at each node.

SLIQ keeps a separate sorted list for each continuous attribute and a separate list called class list. An entry in the class list corresponds to a data item, and has a class label and name of the node it belongs in the decision tree. An entry in the sorted attribute list has an attribute value and the index of data item in the class list. SLIQ grows the decision tree in breadth-first manner. For each attribute, it scans the corresponding sorted list and calculates entropy values of each distinct values of all the nodes in the frontier of the decision tree simultaneously. After the entropy values have been calculated for each attribute, one attribute is chosen for a split for each node in the current frontier, and they are expanded to have a new frontier. Then one more scan of the sorted attribute list is performed to update the class list for the new nodes.

While SLIQ handles disk-resident data that are too large to fit in memory, it still requires some information to stay memory-resident which grows in direct proportion to the number of input records, putting a hard-limit on the size of training data. The Quest team has recently designed a new decision-tree-based classification algorithm, called SPRINT (Scalable Parallelizable Induction of decision Trees) that for the removes all of the memory restrictions.
2.2 Parallel Algorithms

Most of the existing algorithms use local heuristics to handle the computational complexity. The computational complexity of these algorithms ranges from $O(AN \log N)$ to $O(AN(\log N)^2)$ with $N$ training data items and $A$ attributes. These algorithms are fast enough for application domains where $N$ is relatively small. However, in the data mining domain where millions of records and a large number of attributes are involved, the execution time of these algorithms can become prohibitive, particularly in interactive applications. Parallel algorithms have been suggested by many groups developing data mining algorithms. We discuss below two approaches that have been used.[2]

- Basic Idea:

Initially $N$ training data items are randomly distributed to $P$ processors such that each processor has $N=P$ data items. At this point, all the processors cooperate to expand the root node of a decision tree. For this, processors need to decide on an attribute to use to generate child nodes of the root. This can be done in three steps. In the first step, every processor collects the class distribution information of the local data. In the second step, the processors exchange the local class distribution information using global reduction. Finally, each processor can simultaneously compute entropy gains of the attributes and find the best attribute for splitting the root node.

There are two approaches for further progress. In *Synchronous Tree Construction Approach*, the entire set of processors synchronously expands one node of the decision tree at a time. In *partitioned Tree Construction Approach*, each new generated node is expanded by a subset of processors that helped the expansion of the parent node.
2.2.1 Synchronous Tree Construction Approach

In this approach, all processors construct a decision tree synchronously by sending and receiving class distribution information of local data. Major steps for the approach are shown below:

1. Select a node to expand according to a decision tree expansion strategy (eg. Depth-First, Breadth-First or Best-First), and call that node as the current node.
2. For each data attribute, collect class distribution information of the local data at the current node.
3. Exchange the class distribution information with all other processors and add up the class distribution information to get a complete distribution of all the attributes.
4. Calculate the entropy gains of each attribute and select the best attribute for child node expansion.
5. Based on the attribute values, create child nodes and partition the data according to the values of the attribute chosen.
6. Repeat the above steps (1--5) until no more nodes are available for the expansion.

The advantage of this approach is that it does not require any movement of the training data items. However, this algorithm suffers from high communication cost and load imbalance.

Load imbalance can be reduced if all the nodes on the frontier are expanded simultaneously, i.e. one pass of all the data at each processor is used to compute the class distribution information for all nodes on the frontier. Note that this improvement also reduces the number of times communications are done and reduces the message start-up overhead, but it does not reduce the overall volume of communications.
Now the only source of load imbalance is when some leaf nodes become terminal nodes. This load imbalance can be further minimized if the training data set is distributed randomly.

2.2.2 Partitioned Tree Construction Approach

In this approach, each leaf node \( n \) of the frontier of the decision tree is handled by a distinct subset of processors \( P(n) \). Once the node \( n \) is expanded into child nodes, \( n_1, n_2, \ldots, n_k \), the processor group \( P(n) \) is also partitioned into \( k \) parts, \( P_1, P_2, \ldots, P_k \), such that \( P_i \) handle node \( n_i \). All the data items are shuffled such that the processors in group \( P_i \) have data items that belong to the leaf \( n_i \) only. Major steps for the approach are shown below:

1. Expand a node based on the method discussed in the beginning of the Section 3 for expanding the root node.

2. (a) If the number of leaf nodes is less than \( |P(n)| \),

   1. Assign a subset of processor to each leaf node such that number of processors assigned to a leaf node is proportional to the number of the data items contained in the node.
   2. Shuffle the training data such that each subset of processors has data item that belongs to the leaf nodes it is responsible for.
   3. Processor subsets assigned to different nodes develop subtrees of the responsible nodes independently, by following the above steps recursively.

(b) Otherwise,

   1. Partition leaf nodes into \( |P(n)| \) groups such that each group has about the equal number of data items. Assign each processor to one node group.
2. Shuffle the training data such that each processor has data item that belongs to the leaf nodes it is responsible for.

3. Now the expansion of the subtrees rooted at a node group proceeds completely independently at each process.

At the end, the whole decision tree is constructed by combining subtrees of each processor.

The advantage of this approach is that once a processor becomes solely responsible for a node, it can develop a subtree of the decision tree independently without any communication overhead. There are a number of disadvantages of this approach. First disadvantage is that it requires data movement after each node expansion.

The Messages List determines how a Message is displayed in the body element of the page. The Message List is a message-by-message listing of the messages that make up (are contained in) a given folder. The Messages List section is dynamic in the sense that the contents come from the forum database and will change from folder to folder, and from minute to minute as users add more messages to the discussion.

2.3 DECISION TREES

A decision tree is a predictive modeling technique used in classification, clustering, and prediction tasks. Decision trees use a “divide and conquer” technique to split the problem search space into subsets. It is based on the “Twenty Questions” game that children play, as illustrated by Example 1. Figure 2.3 graphically shows the steps in the game. This tree has as the root the first question asked. Each subsequent level in the tree consists of questions at that stage in the game. Nodes at the third level show questions asked at the third level in the game. Leaf nodes represent a successful guess.
as to the object being predicted. This represents a correct prediction.

Each question successively divides the search space much as a binary search does. As with a binary search, questions should be posed so that the remaining space is divided into two equal parts. Often young children tend to ask poor questions by being too specific, such as initially asking "Is it my Mother?" This is a poor approach because the search space is not divided into two equal parts.
EXAMPLE 1:

Stephanie and Shannon are playing a game of "Twenty Questions." Shannon has in mind some object that Stephanie tries to guess with no more than 20 questions. Stephanie's first question is "Is this object alive?" Based on Shannon's answer, Stephanie then asks a second question. Her second question is based on the answer that Shannon provides to the first question. Suppose that Shannon says "yes" as her first answer. Stephanie's second question is "Is this a person?" When Shannon responds "yes," Stephanie asks "Is it a friend?" When Shannon says "no," Stephanie then asks "Is it someone in my family?" When Shannon responds "yes," Stephanie then begins asking the names of family members and can immediately narrow down the search space to identify the target individual. This game is illustrated in Figure 2.3.

DEFINITION 2.3.1. A decision tree (DT) is a tree where the root and each internal node is labeled with a question. The arcs emanating from each node represent each possible answer to the associated question. Each leaf node represents a prediction of a solution to the problem under consideration.

DEFINITION 2.3.2. A decision tree (DT) model is computational model consisting of three parts:

1. A decision tree as defined in Definition 2.3.1.
2. An algorithm to create the tree.
3. An algorithm that applies the tree to data and solves the problem under consideration.

The building of the tree may be accomplished via an algorithm that examines data techniques differ in how the tree is created. We examine several decision tree
techniques. Algorithm 2.3.3 shows the basic steps in applying a tuple to the DT. We assume here that the problem to be performed is one of prediction, so the last step is to make the prediction as dictated by the final leaf node in the tree. The complexity of the algorithm is straightforward to analyze. For each tuple in the database, we search the tree from the root down to a particular leaf. At each level, the maximum number of comparisons to make depends on the product factor at that level. So the complexity depends on the product of the number of levels and the maximum branching factor.

**ALGORITHM 2.3.3**

Input:
- T //Decision tree
- D //Input database

Output:
- M //Model prediction

**DTProc algorithm:**
//Simplistic algorithm to illustrate prediction technique using DT

for each t ∈ D do
  n = root node of T;
  while n not leaf node do
    Obtain answer to question on n applied to t;
    Identify arc from t, which contains correct answer;
    n = node at end of this arc;
    Make prediction for t based on labeling of n;

2.4 DECISION TREE-BASED ALGORITHMS

The decision tree approach is most useful in classification problems. With this technique, a tree is constructed to model the classification process. Once the tree is built, it is applied to each tuple in the database and results in a classification for that tuple. There are two, basic steps in the technique; building the tree and applying the tree to the database. Most research has focused on how to build effective trees as the
The decision tree approach to classification is to divide the search space into rectangular regions. A tuple is classified based on the region into which it falls. A definition for a decision tree used in classification is contained in Definition 2.3.1. There are alternative definitions; for example, in a binary DT the nodes could be labeled with the predicates themselves and each arc would be labeled with yes or no (like in the "Twenty Questions" game).

**DEFINITION 2.3.1** Given a database \( D = \{t_1, t_2, \ldots, t_n\} \) where \( t_i = \langle t_i; A_1, t_i; A_2, \ldots, t_i; A_h\rangle \) and the database schema contains the following attributes \( \{A_1, A_2, \ldots, A_h\} \). Also given is a set of classes \( C = \{C_1, C_2, \ldots, C_m\} \). A **decision tree (DT)** or **classification tree** is a tree associated with \( D \) that has the following properties:

- Each internal node is labeled with an attribute, \( A_i \).
- Each arc is labeled with a predicate that can be applied to the attribute associated with the parent.
- Each leaf node is labeled with a class, \( C_j \).

Solving the classification problem using decision trees is a two-step process:

1. **Decision tree induction**: Construct a DT using training data.
2. For each \( t_i \in D \), apply the DT to determine its class.
2.5 C4.5 AND C5.0 Algorithm

The decision tree algorithm C4.5 improves ID3 in the following ways:

- Missing data: When the decision tree is built, missing data are simply ignored. That is, the gain ratio is calculated by looking only at the other records that have a value for that attribute. To classify a record with a missing attribute value, the value for that item can be predicted based on what is known about the attribute values for the other records.

- Continuous data: The basic idea is to divide the data into ranges based on the attribute values for that item that are found in the training sample.

- Pruning: There are two primary pruning strategies proposed in C4.5:
  - With subtree replacement, a subtree is replaced by a leaf node if this replacement results in an error rate close to that of the original tree. Subtree replacement works from the bottom of the tree up to the root.
  - Another pruning strategy, called subtree raising, replaces a subtree by its most used subtree. Here a subtree is raised from its current location to a node higher up in the tree. Again, we must determine the increase in error rate for this replacement.

- Rules: C4.5 allows classification via either decision trees or rules generated from them. In addition, some techniques to simplify complex rules are proposed. One approach is to replace the left-hand side of a rule by a simpler version if all records in the training set are treated identically. An "otherwise" type of rule can be used to indicate what should be done if no other rules apply.

- Splitting: The ID3 approach favors attributes with many divisions and
thus may lead to overfitting. In the extreme, an attribute that has a unique value for each tuple in the training set would be the best because there would be only one tuple (and thus one class) for each division. An improvement can be made by taking into account the cardinality of each division. This approach uses the GainRatio as opposed to Gain. The GainRatio is defined as

$$GainRatio(D,S) = \frac{Gain(D,S)}{H\left(\frac{|D_1|}{|D|}, \frac{|D_2|}{|D|}\right)}$$

For splitting purposes, C4.5 uses the largest GainRatio that ensures a larger than average information gain. This is to compensate for the fact that the GainRatio value is skewed toward splits where the size of one subset is close to that of the starting one.

C5.0 (called See 5 on Windows) is a commercial version of C4.5 now widely used in many data mining packages such as Clementine and RuleQuest. It is targeted toward use with large datasets. The DT induction is close to that of C4.5, but the rule generation is different. Unlike C4.5, the precise algorithms used for C5.0 have not been divulged. C5.0 does include improvements to generate rules. Results show that C5.0 improves on memory usage by about 90 percent, runs between 5.7 and 240 times faster than C4.5, and produces more accurate rules.

One major improvement to the accuracy of C5.0 is based on boosting. Boosting is an approach to combining different classifiers. While boosting normally increases the time that it takes to run a specific classifier, it does improve the accuracy. The error rate has been shown to be less than half of that found with C4.5 on some datasets [ResOl]. Boosting does not always help when the training data contains a lot of noise.
Boosting works by creating multiple training sets from one training set. Each item in the training set is assigned a weight. The weight indicates the importance of this item to the classification. A classifier is constructed for each combination of weights used. Thus, multiple classifiers are actually constructed. When C5.0 performs a classification, each classifier is assigned a vote, voting is performed, and the target tuple is assigned to the class with the most number of votes.

2.6 CART

Classification and regression trees (CART) are a technique that generates a binary decision tree. As with ID3, entropy is used as a measure to choose the best splitting attribute and criterion. Unlike ID3, however, where a child is created for each subcategory, only two children are created. The splitting is performed around what is determined to be the best split point. At each step, an exhaustive search is used to determine the best split, where "best" is defined by

\[
\phi(s/t) = 2P_L P_R \sum_{j=1}^{m} |P(C_j | t_L) - P(C_j | t_R)|
\]

This formula is evaluated at the current node, \( t \), and for each possible splitting attribute and criterion, \( s \). Here \( L \) and \( R \) are used to indicate the left and right subtrees of the current node in the tree. \( P_L, P_R \) are the probability that a tuple in the training set will be on the left or right side of the tree.

This is defined as \( \frac{|\text{tuples in subtree}|}{|\text{tuples in training set}|} \). We assume that the right branch is taken on equality. \( P(C_j | t_R) \) is the probability that a tuple is in this class, \( C_j \), and in the left or right subtree. This is defined as the \( \frac{|\text{tuples of class } j \text{ in subtree}|}{|\text{tuples at the target node}|} \). At each step, only one criterion is chosen as the best over all possible criteria.

CART forces that an ordering of the attributes be used. CART handles missing data by simply ignoring that record in calculating the goodness of a split on that attribute.
The tree stops growing when no split will improve the performance. Note that even though it is the best for the training data, it may not be the best for all possible data to be added in the future. The CART algorithm also contains a pruning strategy, which we will not discuss here.

2.7 NEURAL NETWORK-BASED ALGORITHMS

With neural networks (NNs), just as with decision trees, a model representing how to classify any given database tuple is constructed. The activation functions typically are sigmoidal. When a tuple must be classified, certain attribute values from that tuple are input into the directed graph at the corresponding source nodes. There often is one sink node for each class. The output value that is generated indicates the probability that the corresponding input tuple belongs to that class. The tuple will then be assigned to the class with the highest probability of membership. The learning process modifies the labeling of the arcs to better classify tuples. Given a starting structure and value for all the labels in the graph, as each tuple in the training set is sent through the network, the projected classification made by the graph can be compared with the actual classification.

Based on the accuracy of the prediction, various labelings in the graph can change. This learning process continues with all the training data or until the classification accuracy is adequate.

Solving a classification problem using NNs involves several steps:

1. Determine the number of output nodes as well as what attributes should be used as input. The number of hidden layers (between the source and the sink nodes) also must be decided. This step is performed by a domain expert.

2. Determine weights (labels) and functions to be used for the graph.

3. For each tuple in the training set, propagate it through the network and evaluate the output prediction to the actual result. If the prediction is accurate, adjust labels to ensure that this prediction has a higher output weight the next time. If
the prediction is not correct, adjust the weights to provide a lower output value for this class.

4. For each tuple $t_i \in D$, propagate $t_i$ through the network and make the appropriate classification.

There are many issues to be examined:

- **Attributes (number of source nodes):** This is the same issue as determining which attributes to use as splitting attributes.
- **Number of hidden layers:** In the simplest case, there is only one hidden layer.
- **Number of hidden nodes:** Choosing the best number of hidden nodes per hidden layer is one of the most difficult problems, where using NNs. There have been many empirical and theoretical studies attempting to answer this question. The answer depends on the structure of the NN; types of activation functions, training algorithm, and problem being solved. If too few hidden nodes are used, the target function may not be learned, (under fitting). If too many nodes are used, over fitting may occur. Rules of thumb are often given that' are based on the size of the training set.
- **Training data:** As with DTs, with too much training data the NN may suffer from over fitting, while too little and it may not be able to classify accurately enough.
- **Number of sinks:** Although it is usually assumed that the number of output nodes is the same as the number of classes, this is not always the case. For example, with two classes there could only be one output node, with the resulting value being the probability of being in the associated class. Subtracting this value from one would give the probability of being in the second class.
- **Interconnections:** In the simplest case" each node is connected to all nodes in the next level.
- **Weights:** The weight assigned to an arc, indicates the, relative weight between
those two nodes. Initial weights are usually assumed to be small positive numbers and are assigned randomly.

- Activation functions: Many different types of activation functions can be used.
- Learning technique: The technique for adjusting the weights is called the learning technique. Although many approaches can be used, the most common approach is some form of back propagation, which is discussed in a subsequent subsection.
- Stop: The learning may stop when all the training tuples have propagated through the network or may be based on time or error rate.

There are many advantages to the use of NNs for classification:

- NNs are more robust than DTs because of the weights.
- The NN improves its performance by learning. This may continue even after the training set has been applied.
- The use of NNs can be parallelized for better performance.
- There is a low error rate and thus a high degree of accuracy once the appropriate training has been performed.
- NNs are more robust than DTs in noisy environments.

Conversely, NNs have many disadvantages:

NNs are difficult to understand. No technical users may have difficulty to understanding how NNs work. While it is easy to explain decision trees, NNs are much more difficult to understand.

- Generating rules from NNs is not straightforward.
- Input attribute values must be numeric.
- Testing
- Verification
- As with DTs, over fitting may result.
- The learning phase may fail to converge.
- NNs may be quite expensive to use.
2.8 Reasons for using C4.5 Algorithm in our Research

We have used C4.5 Algorithm for Construction of Decision Support Tree & for Prediction of Data Results. The Reason why we decided to use C4.5 Algorithm is the following:

(a) C4.5 takes both Numeric & Non Numeric data inputs. CHAID Analysis takes only Non-numeric inputs while CART Analysis takes only Numeric inputs.

(b) C4.5 works best in the presence of Noise. It gives minimal errors. Classification Accuracy of C4.5 Algorithm is better in Noise scenario compared to other Decision Tree Algorithms.