1. What is Classification and Prediction?

Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends.

Whereas classification predicts categorical (discrete, unordered) labels, prediction models continuous valued functions.

Example of classification:

We can build a classification model to categorize bank loan applications as either safe or risk.

Example of prediction:

Regression analysisis a statistical methodology that is most often used for numeric prediction.

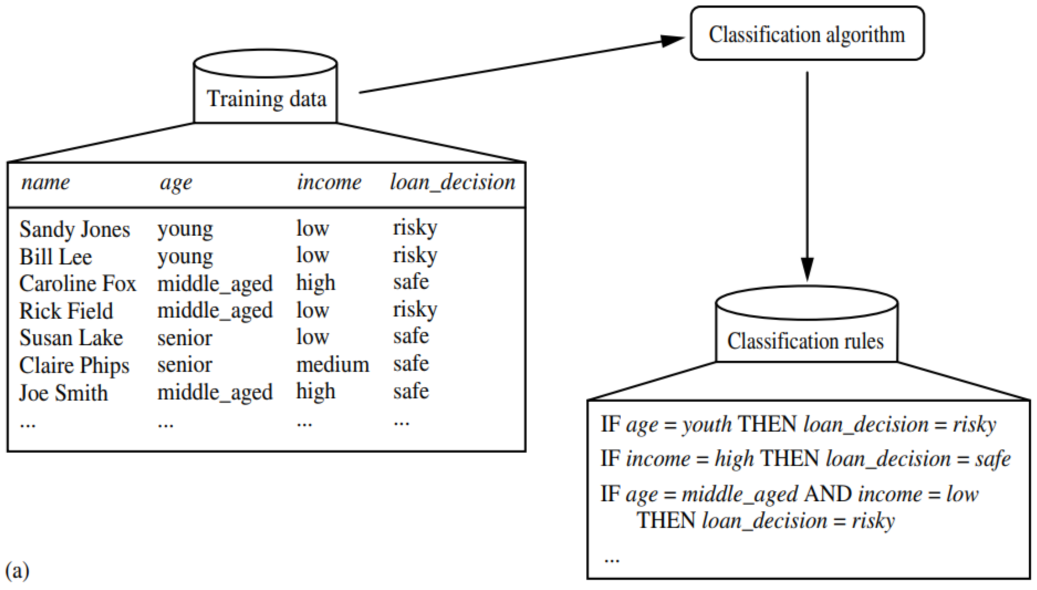
a prediction model to predict the expenditures in dollars of potential customers on computer equipment given their income and occupation.

2. What is data classification ?

Data classification is a two-step process.

In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels. A tuple, X, is represented by an n-dimensional attribute vector, X = (x1, x2,..., xn), depicting n measurements made on the tuple from n database attributes, respectively, A1, A2,..., An.

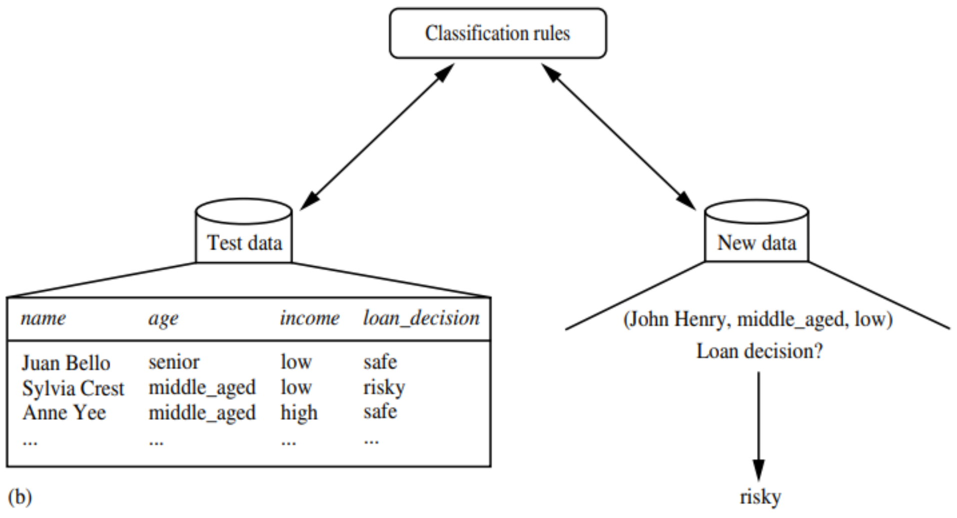
Each tuple, X, is assumed to belong to a predefined class as determined by another database attribute called the **class label attribute.**



Training data
Classification algorithm
Classification rules
IF *age* = *youth* THEN *loan_decision* = *risky*
IF *income* = *high* THEN *loan_decision* = *safe*
IF *age* = *middle_aged* AND *income* = *low* THEN *loan_decision* = *risky*
...

name	*age*	*income*	*loan_decision*
Sandy Jones	young	low	risky
Bill Lee	young	low	risky
Caroline Fox	middle_aged	high	safe
Rick Field	middle_aged	low	risky
Susan Lake	senior	low	safe
Claire Phips	senior	medium	safe
Joe Smith	middle_aged	high	safe
...	...	...	...

(a)



name	*age*	*income*	*loan_decision*
Juan Bello	senior	low	safe
Sylvia Crest	middle_aged	low	risky
Anne Yee	middle_aged	high	safe
...	...	...	...

The data classification process: (a) *Learning*: Training data are analyzed by a classification algorithm. Here, the class label attribute is *loan_decision*, and the learned model or classifier is represented in the form of classification rules. (b) *Classification*: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

Because the class label of each training tuple is provided, this step is also known as **supervised learning.**

It contrasts with **unsupervised learning** (or clustering), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance

In the second step (Figure(b)), the model is used for classification. Test set is used, made up of test tuples and their associated class labels. These tuples are randomly selected from the general data set.

The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier (is called as **classification accuracy)**

3. Issues regarding to Classification and Prediction

A. Preparing the Data for Classification and Prediction:

The following pre-processing steps may be applied to the data to help improve the accuracy, efficiency, and scalability of the classification or prediction process.

Data cleaning: This refers to the preprocessing of data in order to remove or reduce noise (by applying smoothing techniques, for example) and the treatment of missing values (e.g., by replacing a missing value with the most commonly occurring value for that attribute, or with the most probable value based on statistics)

Relevance analysis: Many of the attributes in the data may be redundant. Correlation analysis can be used to identify whether any two given attributes are statistically related.

Data transformation and reduction: The data may be transformed by normalization, particularly when neural networks or methods involving distance measurements are used in the learning step. Normalization involves scaling all values for a given attribute so that they fall within a small specified range, such as −1.0 to 1.0, or 0.0 to 1.0

B. Comparing Classification and Prediction Methods

Accuracy: The accuracy of a classifier refers to the ability of a given classifier to correctly predict the class label of new or previously unseen data (i.e., tuples without class label information).

Speed: This refers to the computational costs involved in generating and using the given classifier or predictor.

Robustness: This is the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values.

Scalability: This refers to the ability to construct the classifier or predictor efficiently given large amounts of data.

Interpretability: This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability is subjective and therefore more difficult to assess.

4. Bayesian Classification

Bayes' Theorem:

Let X be a data tuple. In Bayesian terms, X is considered "evidence." As usual, it is described by measurements made on a set of n attributes.

Let H be some hypothesis, such as that the data tuple X belongs to a specified class C. For classification problems, we want to determine P(H|X), the probability that the hypothesis H holds given the "evidence" or observed data tuple X.

P(H|X) is the posterior probability, or a posteriori probability, of H conditioned on X.

In contrast, P(H) is the prior probability, or a priori probability, of H

The posterior probability, P(H|X), is based on more information (e.g., customer information) than the prior probability, P(H), which is independent of X.

Bayes' theorem is useful in that it provides a way of calculating the posterior probability, P(H|X), from P(H), P(X|H), and P(X). Bayes' theorem is P

$$P(H|X) = P(X|H)\, P(H) \,/\, P(X)$$

Naïve Bayesian Classification

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = (x1, x2,..., xn), depicting n measurements made on the tuple from n attributes, respectively, A1, A2,..., An.
2. Suppose that there are m classes, C1, C2,..., Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X.

$$P(C_i \,|X) > P(C_j \,|X) \text{ for } 1 \le j \le m, \text{ j is not equal to i.}$$

Thus we maximize P(Ci |X). The classCi for which P(Ci |X) is maximized is called the maximum posteriori hypothesis.

By Bayes' theorem , P(Ci |X) = P(X|Ci) P(Ci) / P(X) .

3. As P(X) is constant for all classes, only P(X|Ci) P(Ci) need be maximized
4. Given data sets with many attributes, it would be extremely computationally expensive to compute P(X|Ci).

$$\begin{aligned} P(\boldsymbol{X}|C_i) &= \prod_{k=1}^{n} P(x_k|C_i) \\ &= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i). \end{aligned}$$

(a) If Ak is categorical, then P(xk|Ci) is the number of tuples of class Ci in D having the value xk for Ak, divided by |Ci,D|, the number of tuples of class Ci in D.

(b) A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}).$$

Example Problem- Bayesian Classification

Table 6.1 Class-labeled training tuples from the *AllElectronics* customer database.

RID	*age*	*income*	*student*	*credit_rating*	*Class: buys_computer*
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

The training data are in Table 6.1. The data tuples are described by the attributes age, income, student, and credit rating.

The class label attribute, buys computer, has two distinct values (namely, {yes, no}).

Let C1 correspond to the class buys computer = yes and

C2 correspond to buys computer = no. The tuple we wish to classify is X = (age = youth, income = medium, student = yes, credit rating = fair)

We need to maximize P(X|Ci)P(Ci), for i = 1, 2. P(Ci), the prior probability of each class, can be computed based on the training tuples:

P(buys computer = yes) = 9/14 = 0.643

P(buys computer = no) = 5/14 = 0.357

To compute PX|Ci), for i = 1, 2, we compute the following conditional probabilities:

P(age = youth | buys computer = yes) = 2/9 = 0.222

P(age = youth | buys computer = no) = 3/5 = 0.600

P(income = medium | buys computer = yes) = 4/9 = 0.444

P(income = medium | buys computer = no) = 2/5 = 0.400

P(student = yes | buys computer = yes) = 6/9 = 0.667

P(student = yes | buys computer = no) = 1/5 = 0.200

P(credit rating = fair | buys computer = yes) = 6/9 = 0.667

P(credit rating = fair | buys computer = no) = 2/5 = 0.400

P(X|buys computer = yes) =

P(age = youth | buys computer = yes) × P(income = medium | buys computer = yes) × P(student = yes | buys computer = yes) × P(credit rating = fair | buys computer = yes)

= 0.222×0.444×0.667×0.667 = 0.044.

Similarly,

P(X|buys computer = no) = 0.600×0.400×0.200×0.400 = 0.019

To find the class, Ci , that maximizes P(X|Ci)P(Ci),

we compute

P(X|buys computer = yes)P(buys computer = yes) = 0.044×0.643 = 0.028

P(X|buys computer = no)P(buys computer = no) = 0.019×0.357 = 0.007

Therefore, **the naïve Bayesian classifier predicts buys computer = yes for tuple X.**

5. Decision Tree Classification

Decision tree induction is the learning of decision trees from class-labeled training tuples.

A decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label.

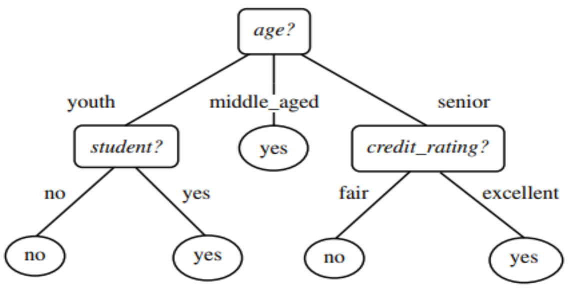


Fig. Decision Tree- Example

It represents the concept buys computer, that is, it predicts whether a customer at All Electronics is likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.

“How are decision trees used for classification?”

Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple.

Decision Tree Induction Algorithm:

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that “best” partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1)  create a node $N$;
(2)  if tuples in $D$ are all of the same class, $C$ then
(3)      return $N$ as a leaf node labeled with the class $C$;
(4)  if $attribute\_list$ is empty then
(5)      return $N$ as a leaf node labeled with the majority class in $D$; // majority voting
(6)  apply Attribute_selection_method($D$, $attribute\_list$) to find the "best" $splitting\_criterion$;
(7)  label node $N$ with $splitting\_criterion$;
(8)  if $splitting\_attribute$ is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      $attribute\_list \leftarrow attribute\_list - splitting\_attribute$; // remove $splitting\_attribute$
(10) for each outcome $j$ of $splitting\_criterion$
     // partition the tuples and grow subtrees for each partition
(11)     let $D_j$ be the set of data tuples in $D$ satisfying outcome $j$; // a partition
(12)     if $D_j$ is empty then
(13)         attach a leaf labeled with the majority class in $D$ to node $N$;
(14)     else attach the node returned by Generate_decision_tree($D_j$, $attribute\_list$) to node $N$;
     endfor
(15) return $N$;
```

Attribute Selection Measures

An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes.

1. Information gain

Let node N represent or hold the tuples of partition D. The attribute with the highest information gain is chosen as the splitting attribute for node N. This attribute minimizes the information needed to classify the tuple in the resulting partitions.

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is,

$$Gain(A) = Info(D) - Info_A(D).$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

The term $\frac{|D_j|}{|D|}$ acts as the weight of the jth partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions.

Example of Information Gain

Consider the Table 6.1

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits}.$$

$$\begin{aligned} Info_{age}(D) &= \frac{5}{14} \times (-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}) \\ &+ \frac{4}{14} \times (-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}) \\ &+ \frac{5}{14} \times (-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}) \\ &= 0.694 \text{ bits.} \end{aligned}$$

Hence, the gain in information from such a partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

2. Gain ratio

The gain ratio is defined as

GainRatio(A) = Gain(A) / SplitInfo(A)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right).$$

Computation of gain ratio for the attribute *income*. A test on *income* splits the data of Table 6.1 into three partitions, namely *low*, *medium*, and *high*, containing four, six, and four tuples, respectively. To compute the gain ratio of *income*,
to obtain

$$\begin{aligned} SplitInfo_A(D) &= -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right). \\ &= 0.926. \end{aligned}$$

we have $Gain(income) = 0.029$. Therefore, $GainRatio(income) = 0.029/0.926 = 0.031$. ■

3. Gini index

the Gini index measures the impurity of D, a data partition or set of training tuples, as

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

where p_i is the probability that a tuple in D belongs to class C_i and is estimated by $|C_{i,D}|/|D|$. The sum is computed over m classes.

When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on A partitions D into D_1 and D_2, the gini index of D given that partitioning is

$$Gini_A(D) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2).$$

Example:

Induction of a decision tree using gini index. Let D be the training data of Table 6.1 where there are nine tuples belonging to the class *buys_computer* = *yes* and the remaining five tuples belong to the class *buys_computer* = *no*. A (root) node N is created for the tuples in D. We first use Equation (6.7) for Gini index to compute the impurity of D:

$$Gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459.$$

To find the splitting criterion for the tuples in D, we need to compute the gini index for each attribute. Let's start with the attribute *income* and consider each of the possible splitting subsets. Consider the subset $\{low, medium\}$. This would result in 10 tuples in partition D_1 satisfying the condition "$income \in \{low, medium\}$." The remaining four tuples of D would be assigned to partition D_2. The Gini index value computed based on this partitioning is

$$\begin{aligned} &Gini_{income \in \{low, medium\}}(D) \\ &= \frac{10}{14}Gini(D_1) + \frac{4}{14}Gini(D_2) \\ &= \frac{10}{14}\left(1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2\right) \\ &= 0.450 \\ &= Gini_{income \in \{high\}}(D). \end{aligned}$$

Similarly, the Gini index values for splits on the remaining subsets are: 0.315 (for the subsets $\{low, high\}$ and $\{medium\}$) and 0.300 (for the subsets $\{medium, high\}$ and $\{low\}$). Therefore, the best binary split for attribute *income* is on $\{medium, high\}$ (or $\{low\}$) because it minimizes the gini index. Evaluating the attribute, we obtain $\{youth, senior\}$ (or $\{middle\_aged\}$) as the best split for *age* with a Gini index of 0.375; the attributes $\{student\}$ and $\{credit\_rating\}$ are both binary, with Gini index values of 0.367 and 0.429, respectively.

The attribute *income* and splitting subset $\{medium, high\}$ therefore give the minimum gini index overall, with a reduction in impurity of $0.459 - 0.300 = 0.159$.

Tree Pruning

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data. Such methods typically use statistical measures to remove the least reliable branches.

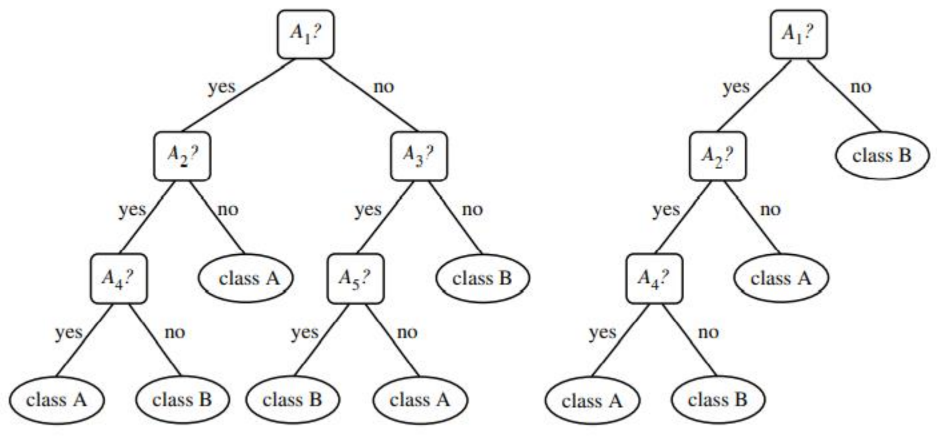
In the prepruning approach, a tree is "pruned" by halting its construction early. Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.

When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on can be used to assess the goodness of a split. If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted.

The second and more common approach is postpruning, which removes subtrees from a "fully grown" tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf.

Example:

Suppose that the most common class within this subtree is "class B." In the pruned version of the tree, the subtree in question is pruned by replacing it with the leaf "class B.



An unpruned decision tree and a pruned version of it.

Scalability and Decision Tree Induction

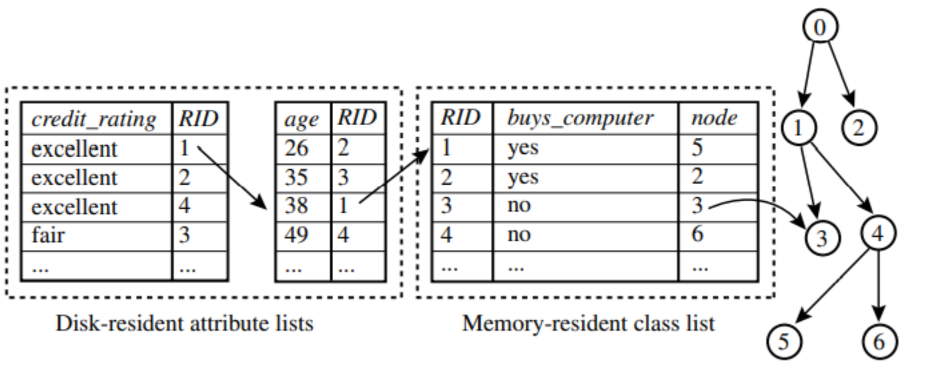
More recent decision tree algorithms that address the scalability issue have been proposed. Algorithms for the induction of decision trees from very large training sets include SLIQ and SPRINT, both of which can handle categorical and continuous valued attributes. Both algorithms propose pre sorting techniques on disk-resident data sets that are too large to fit in memory. Both define the use of new data structures to facilitate the tree construction.

SLIQ

SLIQ employs disk-resident attribute lists and a single memory-resident class list. The attribute lists and class list generated by SLIQ for the tuple data of Table 6.2 are shown in Figure 6.8. Each attribute has an associated attribute list, indexed by RID (a record identifier). Each tuple is represented by a linkage of one entry from each attribute list to an entry in the class list (holding the class label of the given tuple), which in turn is linked to its corresponding leaf node.

Table 6.2 Tuple data for the class *buys_computer*.

RID	*credit_rating*	*age*	*buys_computer*
1	excellent	38	yes
2	excellent	26	yes
3	fair	35	no
4	excellent	49	no
...	...	...	...



credit_rating	*RID*
excellent	1
excellent	2
excellent	4
fair	3
...	...

age	*RID*
26	2
35	3
38	1
49	4
...	...

RID	*buys_computer*	*node*
1	yes	5
2	yes	2
3	no	3
4	no	6
...	...	...

Figure 6.8 Attribute list and class list data structures used in SLIQ for the tuple data of Table 6.2.

The class list remains in memory because it is often accessed and modified in the building and pruning phases. The size of the class list grows proportionally with the number of tuples in the training set. When a class list cannot fit into memory, the performance of SLIQ decreases.

SPRINT

SPRINT uses a different attribute list data structure that holds the class and RID information, as shown in Figure 6.9. When a node is split, the attribute lists are partitioned and distributed among the resulting child nodes accordingly. When a list is partitioned, the order of the records in the list is maintained. Hence, partitioning lists does not require resorting. SPRINT was designed to be easily parallelized, further contributing to its scalability.

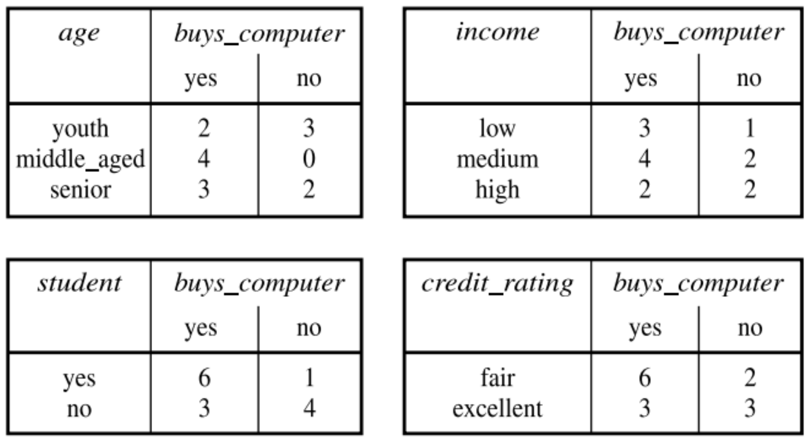
credit_rating	*buys_computer*	*RID*
excellent	yes	1
excellent	yes	2
excellent	no	4
fair	no	3
...	...	...

age	*buys_computer*	*RID*
26	yes	2
35	no	3
38	yes	1
49	no	4
...	...	...

Figure 6.9 Attribute list data structure used in SPRINT for the tuple data of Table 6.2.

RainForest

To further enhance the scalability of decision tree induction, a method called RainForest was proposed. It adapts to the amount of main memory available and applies to any decision tree induction algorithm. The method maintains an AVC-set (where AVC stands for “Attribute-Value, Classlabel”) for each attribute, at each tree node, describing the training tuples at the node. The AVC-set of an attribute A at node N gives the class label counts for each value of A for the tuples at N. Figure 6.10 shows AVC-sets for the tuple data of Table 6.1. The set of all AVC-sets at a node N is the AVC-group of N. The size of an AVC-set for attribute A at node N depends only on the number of distinct values of A and the number of classes in the set of tuples at N. Typically, this size should fit in memory, even for real-world data



age	*buys_computer*	
	yes	no
youth	2	3
middle_aged	4	0
senior	3	2

income	*buys_computer*	
	yes	no
low	3	1
medium	4	2
high	2	2

student	*buys_computer*	
	yes	no
yes	6	1
no	3	4

credit_rating	*buys_computer*	
	yes	no
fair	6	2
excellent	3	3

The use of data structures to hold aggregate information regarding the training data (such as these AVC-sets describing the data of Table 6.1) are one approach to improving the scalability of decision tree induction.

RainForest can use any attribute selection measure and was shown to be more efficient than earlier approaches employing aggregate data structures, such as SLIQ and SPRINT.

Bootstrapped Optimistic Algorithm for Tree Construction

BOAT (Bootstrapped Optimistic Algorithm for Tree Construction) is a decision tree algorithm that takes a completely different approach to scalability—it is not based on the use of any special data structures. Instead, it uses a statistical technique known as “bootstrapping” to create several smaller samples (or subsets) of the given training data, each of which fits in memory

Rule-Based Classification

a. Using IF-THEN Rules for Classification

A rule-based classifier uses a set of IF-THEN rules for classification.

An IF-THEN rule is an expression of the form IF condition THEN conclusion.

An example is rule R1:

R1: IF age = youth AND student = yes THEN buys computer = yes.

The “IF”-part (or left-hand side) of a rule is known as the rule antecedent or precondition. The “THEN”-part (or right-hand side) is the rule consequent.

R1 can also be written as

R1: (age = youth) ∧ (student = yes) ⇒ (buys computer = yes)

A rule R can be assessed by its coverage and accuracy. Given a tuple, X, from a class-labeled data set, D, let n_{covers} be the number of tuples covered by R; $n_{correct}$ be the number of tuples correctly classified by R; and $|D|$ be the number of tuples in D. We can define the **coverage** and **accuracy** of R as

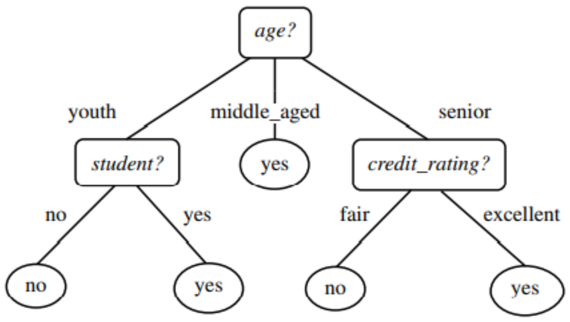
$$coverage(R) = \frac{n_{covers}}{|D|}$$

$$accuracy(R) = \frac{n_{correct}}{n_{covers}}.$$

Rule accuracy and coverage. Let’s go back to our data of Table 6.1. These are class-labeled tuples from the *AllElectronics* customer database. Our task is to predict whether a customer will buy a computer. Consider rule $R1$ above, which covers 2 of the 14 tuples. It can correctly classify both tuples. Therefore, $coverage(R1) = 2/14 = 14.28\%$ and $accuracy(R1) = 2/2 = 100\%$. ■

b. Rule Extraction from a Decision Tree

To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically ANDed to form the rule antecedent (“IF” part).



Extracting classification rules from a decision tree. The decision tree of Figure 6.2 can be converted to classification IF-THEN rules by tracing the path from the root node to each leaf node in the tree. The rules extracted from Figure 6.2 are

$R1$: IF $age = youth$ AND $student = no$ THEN $buys\_computer = no$
$R2$: IF $age = youth$ AND $student = yes$ THEN $buys\_computer = yes$
$R3$: IF $age = middle\_aged$ THEN $buys\_computer = yes$
$R4$: IF $age = senior$ AND $credit\_rating = excellent$ THEN $buys\_computer = yes$
$R5$: IF $age = senior$ AND $credit\_rating = fair$ THEN $buys\_computer = no$

A Multilayer Feed-Forward Neural Network

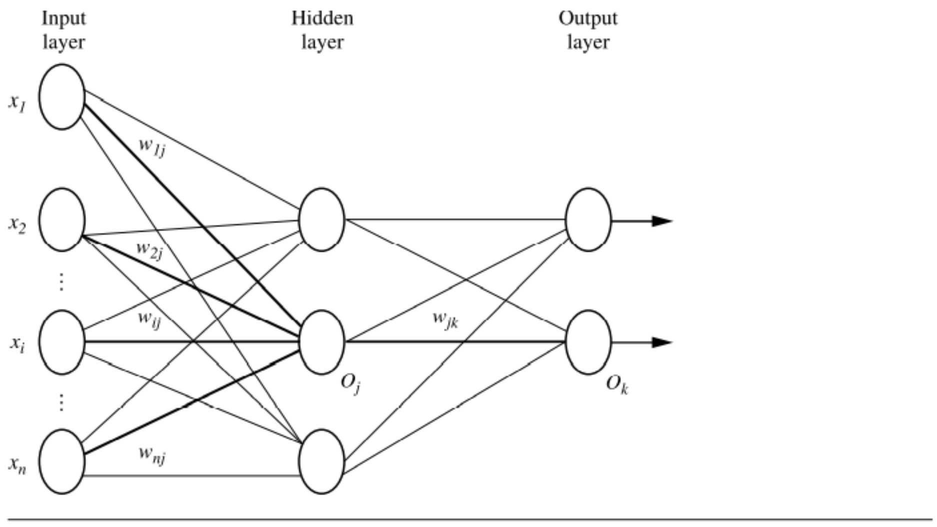
A multilayer feed-forward neural network consists of an <u>input layer, one or more hidden layers, and an output layer</u>. An example of a multilayer feed-forward network is shown in Figure 6.15.

<u>Each layer is made up of units.</u> The inputs to the network correspond to the attributes measured for each training tuple. The inputs are fed simultaneously into the units making up the input layer.

These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of "neuronlike" units, known as a hidden layer.

The outputs of the hidden layer units can be input to another hidden layer, and so on.

The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction for given tuples.



A multilayer feed-forward neural network.

<u>The network is feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer.</u> It is fully connected in that each unit provides input to each unit in the next forward layer.

<u>**Defining a Network Topology**</u>

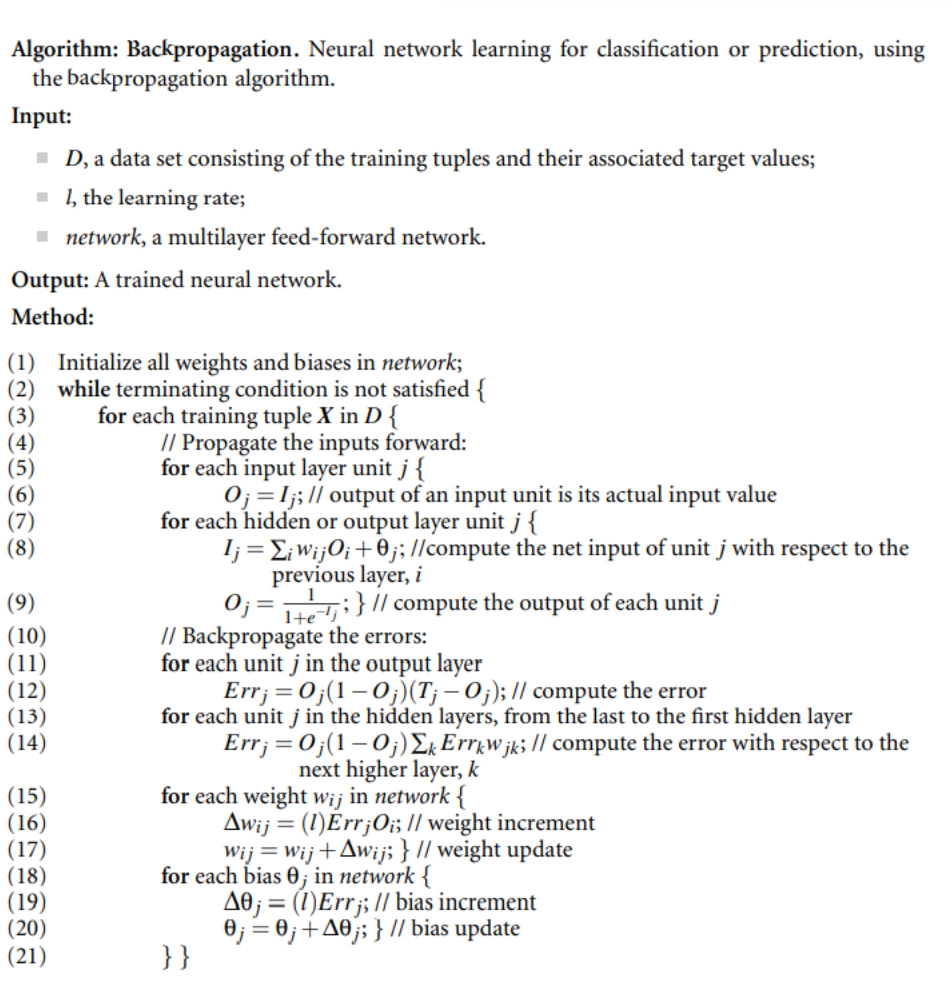
Before training can begin, the user must decide on the network topology by specifying the number of units in the input layer, the number of hidden layers (if more than one), the number of units in each hidden layer, and the number of units in the output layer.

Normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. Typically, input values are normalized so as to fall between 0.0 and 1.0.

There are no clear rules as to the "best" number of hidden layer units. Network design is a trial-and-error process and may affect the accuracy of the resulting trained network. The initial values of the weights may also affect the resulting accuracy.

Backpropagation

For each training tuple, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual target value. These modifications are made in the "backwards" direction, that is, from the output layer, through each hidden layer down to the first hidden layer (hence the name backpropagation.



Algorithm: Backpropagation. Neural network learning for classification or prediction, using the backpropagation algorithm.

Input:

- D, a data set consisting of the training tuples and their associated target values;
- l, the learning rate;
- *network*, a multilayer feed-forward network.

Output: A trained neural network.

Method:

```
(1)  Initialize all weights and biases in network;
(2)  while terminating condition is not satisfied {
(3)      for each training tuple $X$ in $D$ {
(4)          // Propagate the inputs forward:
(5)          for each input layer unit $j$ {
(6)              $O_j = I_j$; // output of an input unit is its actual input value
(7)          for each hidden or output layer unit $j$ {
(8)              $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit $j$ with respect to the
                     previous layer, $i$
(9)              $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit $j$
(10)         // Backpropagate the errors:
(11)         for each unit $j$ in the output layer
(12)             $Err_j = O_j(1-O_j)(T_j - O_j)$; // compute the error
(13)         for each unit $j$ in the hidden layers, from the last to the first hidden layer
(14)             $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the
                     next higher layer, $k$
(15)         for each weight $w_{ij}$ in network {
(16)             $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
(17)             $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
(18)         for each bias $\theta_j$ in network {
(19)             $\Delta \theta_j = (l) Err_j$; // bias increment
(20)             $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
(21)         } }
```

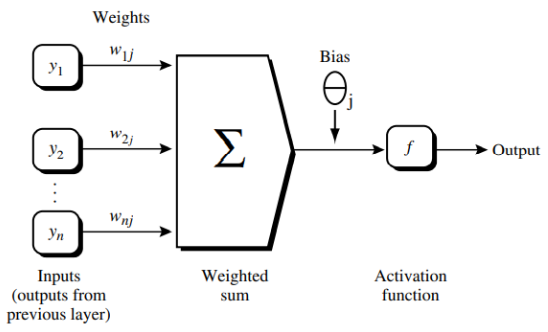
Initialize the weights: The weights in the network are initialized to small random numbers (e.g., ranging from −1.0 to 1.0, or −0.5 to 0.5).

Propagate the inputs forward:

That is, for an input unit, j, its output, O_j, is equal to its input value, I_j.

Given a unit j in a hidden or output layer, the net input, I_j, to unit j is

$$I_j = \sum_i w_{ij} O_i + \theta_j,$$



The **logistic**, or **sigmoid**, function is used. Given the net input I_j to unit j, then O_j, the output of unit j, is computed as

$$O_j = \frac{1}{1+e^{-I_j}}.$$

Backpropagate the error: The error is propagated backward by updating the weights and biases to reflect the error of the network's prediction. For a unit j in the output layer, the error Err_j is computed by

$$Err_j = O_j(1-O_j)(T_j-O_j),$$

To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit j in the next layer are considered. The error of a hidden layer unit j is

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk},$$

where w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer, and Err_k is the error of unit k.

The weights and biases are updated to reflect the propagated errors. Weights are updated by the following equations, where Δw_{ij} is the change in weight w_{ij}:

$$\Delta w_{ij} = (l)Err_j O_i$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

The variable l is the **learning rate**, a constant typically having a value between 0.0 and 1.0.

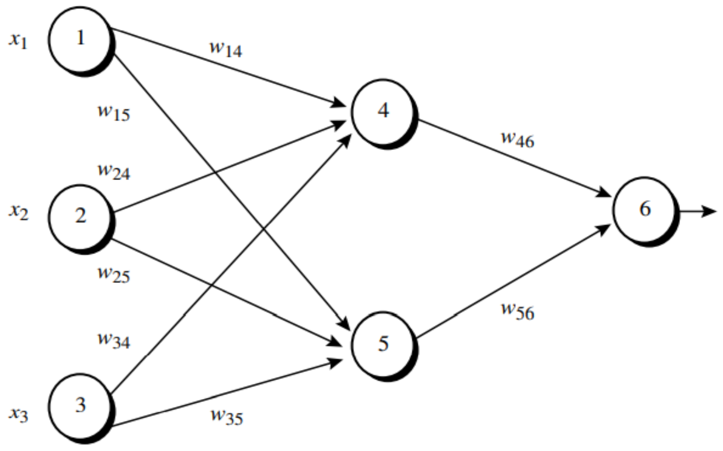
Biases are updated by the following equations below, where $\Delta\theta_j$ is the change in bias θ_j:

$$\Delta\theta_j = (l)Err_j$$

$$\theta_j = \theta_j + \Delta\theta_j$$

Example Problem:

X = (1, 0, 1), whose class label is 1.



B An example of a multilayer feed-forward neural network.

Initial input, weight, and bias values.

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

The net input and output calculations.

Unit j	*Net input,* I_j	*Output,* O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

Calculation of the error at each node.

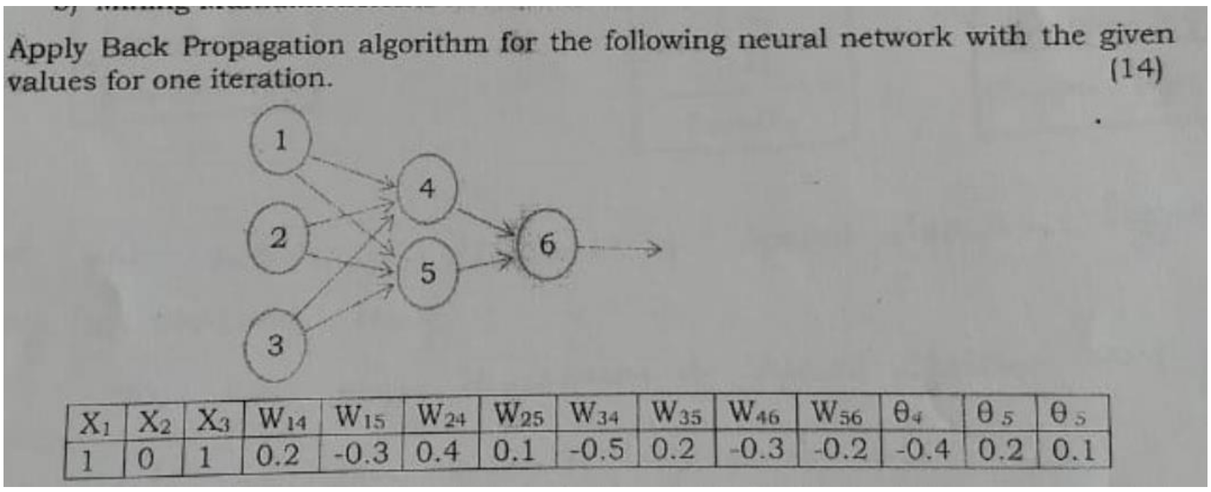
Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474)=0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

Calculations for weight and bias updating.

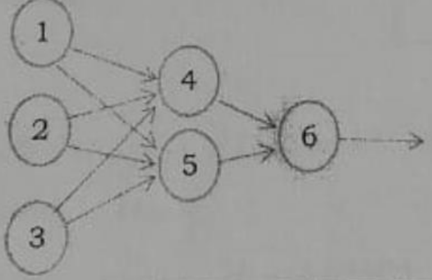
Weight or bias	*New value*
w_{46}	$-0.3+(0.9)(0.1311)(0.332)=-0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525)=-0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1)=0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1)=-0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0)=0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0)=0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1)=-0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1)=0.194$
θ_6	$0.1+(0.9)(0.1311)=0.218$
θ_5	$0.2+(0.9)(-0.0065)=0.194$
θ_4	$-0.4+(0.9)(-0.0087)=-0.408$

Questions:

1. **Explain about scalability in decision tree induction**
2. **Explain about repetition and replication in decision tree induction**
3. **A classifier is classifying 180 patterns out of 200 patterns correctly. What is its misclassification rate.**
4. **Why is tree pruning useful in decision tree induction? What is a drawback of using a separate set of tuples to evaluate pruning?**
5. **Explain about rule-based classification.**
6. **What is data classification**
7. **Write back propagation algorithm.**
8. **Explain back propagation classification**
9. **What are the evaluation criteria for classification and prediction**
10. **A classifier is classifying 90 patterns out of 110 patterns correctly. What is its accuracy.**
11. **Explain about the following**
    a. **Random subsampling**
    b. **Cross-validation method**
    c. **Boot strap method**
12.



Apply Back Propagation algorithm for the following neural network with the given values for one iteration. (14)



X_1	X_2	X_3	W_{14}	W_{15}	W_{24}	W_{25}	W_{34}	W_{35}	W_{46}	W_{56}	θ_4	θ_5	θ_5
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

13. What is bayes theorem
14. What are the issues in classification and prediction?
15. What is instance-based learners
16. Explain how the Bayesian belief networks is differing in naïve Bayesian classification.
17. Define information gain
18. What is boosting. Explain in detail why it may improve the accuracy of a decision tree.