CHAPTER II

RULE EXTRACTION PROCESS

Neural network is a universal approximator. It works best if the system model of the problem has a high tolerance to error. It is suitable for (i) capturing associations or discovering regularities within a set of patterns where the volume, number of variables, or diversity of the data are very high, and (ii) the relationship between variables are not well defined. Neural networks are typically organized in layers. Each layer consists of a number of interconnected nodes each of which contains an activation function. The hidden layer is connected to the output layer whose output values are used to defined classed as shown in Figure 2.1. The shown network is fully connected, which means that a neuron in any layer of the network is connected to all the nodes/neurons in the previous layer. Backpropagation algorithm uses gradient descent to adjust the parameters of the network to best fit the training set. The input data are treated as neuron excitation parameters and fed into the first layer. These excitations of the current layer neurons are propagated to the next layer neurons being amplified or weakened according to weights (numerical coefficients) ascribed to corresponding intra-neural connections. Based on this observation on the neuron activity, it is possible to extract some comprehensible rules governing the classification process.

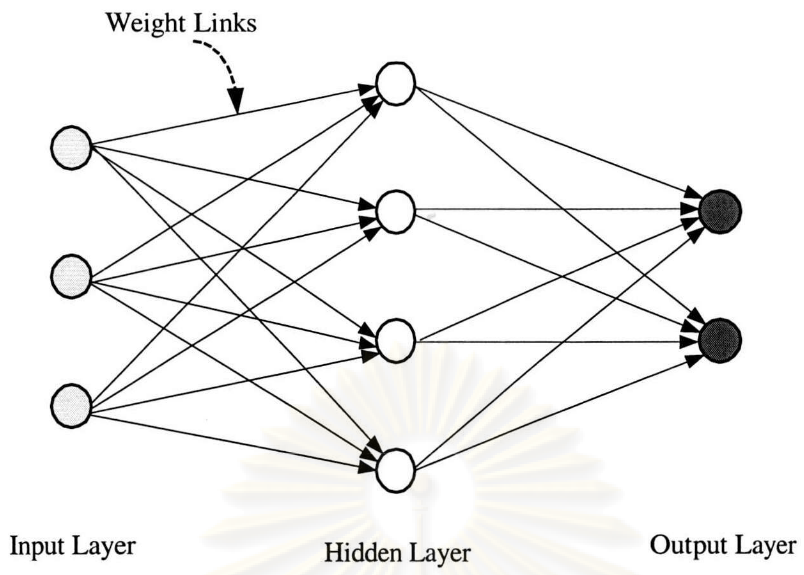
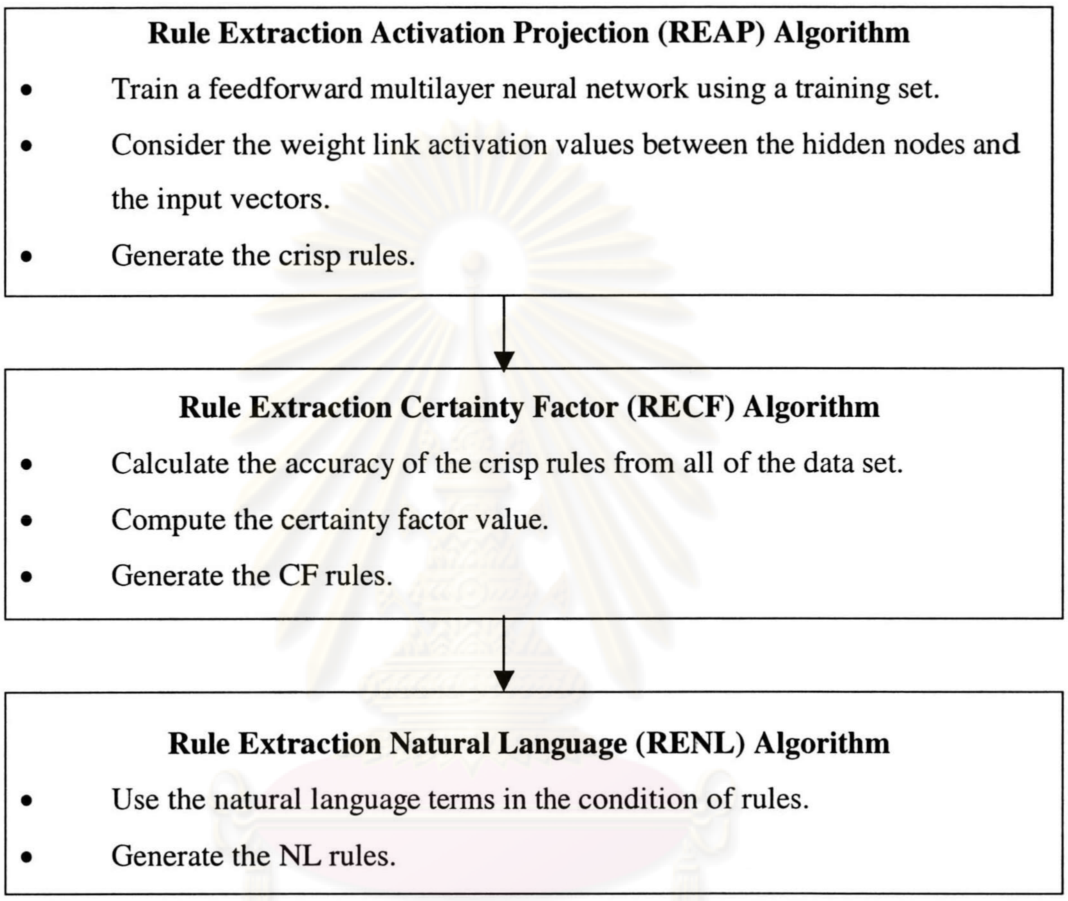


Figure 2.1 Neural network model.

The proposed rule extraction process has three parts as shown in Figure 2.2. The algorithms for a rule extraction process are *(i) Rule Extraction Activation Projection (REAP) algorithm* [23], *(ii) Rule Extraction Certainty Factor (RECF) algorithm* [24], and *(iii) Rule Extraction Natural Language (RENL) algorithm*. For the first part of (i) REAP algorithm, the training set are given to the MLP neural network, then the network was trained and produced the weight links. The REAP algorithm uses the activation values of the hidden nodes with the values of each input to extract the rules called *crisp rule*. In the second part of (ii) RECF algorithm, a constant called *certainty factor* is defined and attached with each crisp rule to provide the accuracy of each crisp rule. Rule with a certainty factor attached to each rule is called *CF rule*. This certainty value about the class is a useful piece of information for making decision in some applications such as medical diagnosis or weather forecast. The third part of (iii) RENL algorithm concerns the natural language terms used to express the extracted rules in a more human comprehensible term. The value of each input is captured by a quantitative natural language term such as "small", "medium", or "large". Rule with a natural language term is called *NL rule*. The details of all three algorithms (REAP algorithm, RECF algorithm, and RENL algorithm) are given in Sections 2.1 to 2.3, respectively.



Rule Extraction Activation Projection (REAP) Algorithm

- Train a feedforward multilayer neural network using a training set.
- Consider the weight link activation values between the hidden nodes and the input vectors.
- Generate the crisp rules.

Rule Extraction Certainty Factor (RECF) Algorithm

- Calculate the accuracy of the crisp rules from all of the data set.
- Compute the certainty factor value.
- Generate the CF rules.

Rule Extraction Natural Language (RENL) Algorithm

- Use the natural language terms in the condition of rules.
- Generate the NL rules.

Figure 2.2 Rule extraction process.

2.1 Rule Extraction Activation Projection (REAP) Algorithm

A neural network module with three-layer fully connected architecture (number of input units: number of hidden units: number of output units) is used. The network is based on the architecture of multi-layer perceptron (MLP). The MLP network is trained with a set of training data using the backpropagation learning rule. A single hidden layer of neurons partitions the feature space into convex intersections of halfspaces determined by neuron hyperplanes. An output layer of neurons is required to join the combinations of those convex regions into non-convex classes in the feature space.

2.1.1 Geometrical Meaning of Activation Equation

The goal of pattern classification is to specify a physical object to one of the pre-specified categories. Let m be the number of input vectors, n be the number of input features, and $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m\}$, where $\boldsymbol{p}_k = [x_1(k)\ x_2(k)\ \ldots\ x_n(k)]^{\mathrm{T}}$ be a set of given patterns to be classified into C categories. The category of each pattern $\boldsymbol{p}_k$ is determined prior to the classification process. Generally, a neuron linearly classifies a set of patterns by computing the activation values and passing these values to a threshold function. The activation value is defined as the dot products between the input vector $\boldsymbol{p}_k$ and its weight vector $\boldsymbol{w}_j$. For example, the activation value h_j with respect to pattern $\boldsymbol{p}_k = [x_1(k)\ \ x_2(k)]^{\mathrm{T}}$ of neuron j is computed by

$$h_j(p_k) = w_{j,1}x_1(k) + w_{j,2}x_2(k) + b_j. \qquad (1)$$

where $w_{j,i}$ is weight link i of neuron j, and b_j is the bias of neuron j. This equation can be viewed as a line in a 2-dimensional space. The concept can be extended to a higher dimensional space and the activation equation represents a hyperplane.

2.1.2 Analytic Geometry in Euclidean Space in Cartesian Coordinates

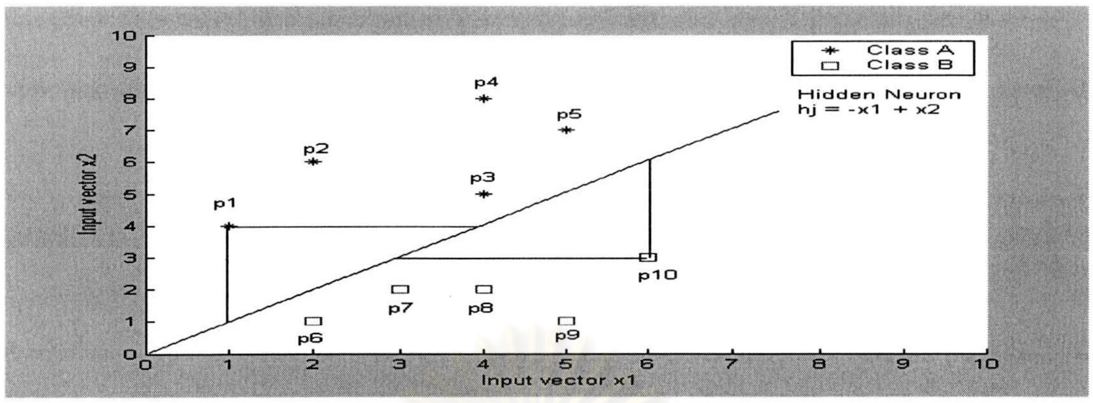
Class A is defined as a class which their dot products between an input vectors $\boldsymbol{p}_k$ with $\boldsymbol{w}_j$ are greater than or equal to zero and *class B* as a class which their dot products between input vectors $\boldsymbol{p}_k$ with $\boldsymbol{w}_j$ are less than zero. An example of this hyperplane is illustrated in Figure 2.3(a). All vectors denoted by asterisks above the separating line are in class A and all vectors denoted by boxes and beneath the separating line are in class B. Here, there are five input vectors in class A and five input vectors in class B.

To meaningfully relate each input feature x_i with each class, the activation value of each input vector $\boldsymbol{p_k}$ and the value of each x_i are considered. This can be achieved easily by plotting the activation value of each input vector $\boldsymbol{p_k}$ against the values of x_i as shown in Figure 2.3(b) and Figure 2.3(c).

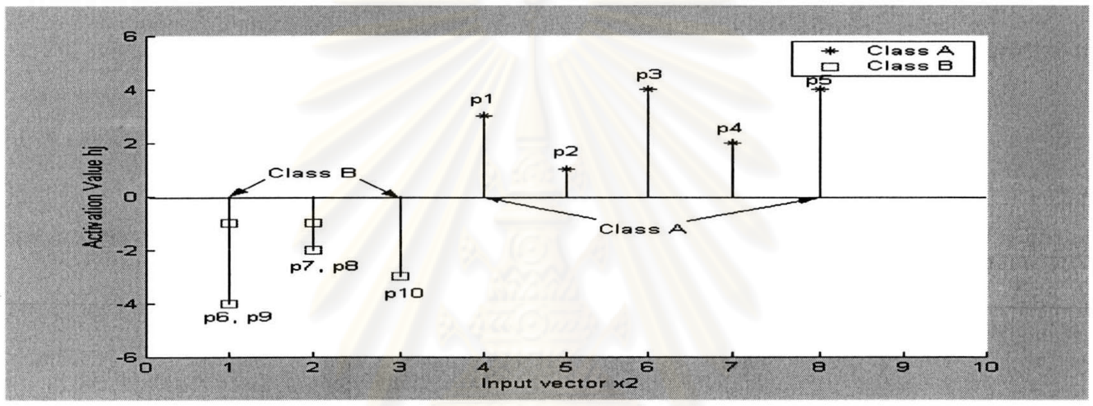
Let $\boldsymbol{p_1}$, $\boldsymbol{p_2}$, $\boldsymbol{p_3}$, $\boldsymbol{p_4}$, and $\boldsymbol{p_5}$ be input vectors in class A and $\boldsymbol{p_6}$, $\boldsymbol{p_7}$, $\boldsymbol{p_8}$, $\boldsymbol{p_9}$, and $\boldsymbol{p_{10}}$ be input vectors in class B with the following coordinates: $\boldsymbol{p_1} = (1,4)$, $\boldsymbol{p_2} = (2,6)$, $\boldsymbol{p_3} = (4,5)$, $\boldsymbol{p_4} = (4,8)$, $\boldsymbol{p_5} = (5,7)$, $\boldsymbol{p_6} = (2,1)$, $\boldsymbol{p_7} = (3,2)$, $\boldsymbol{p_8} = (4,2)$, $\boldsymbol{p_9} = (5,1)$, and $\boldsymbol{p_{10}} = (6,3)$. The weight vector $\boldsymbol{w_j}$ is equal to $[-1\ 1]^T$. The activation values of all input vectors, $\boldsymbol{p_1}$ to $\boldsymbol{p_{10}}$, are 3, 4, 1, 4, 2, –1, –1, –2, –4, and –3, respectively.

Suppose that we are interested in extracting the condition defined by the second element of an input feature, namely x_2. The activation values are plotted against the values of x_2. All the positive activation values (class A) are above the zero line and all the negative activation values (class B or non-class A) are beneath the zero line as shown in Figure 2.3(b). There is no mixture of both positive and negative activation values for a particular class.

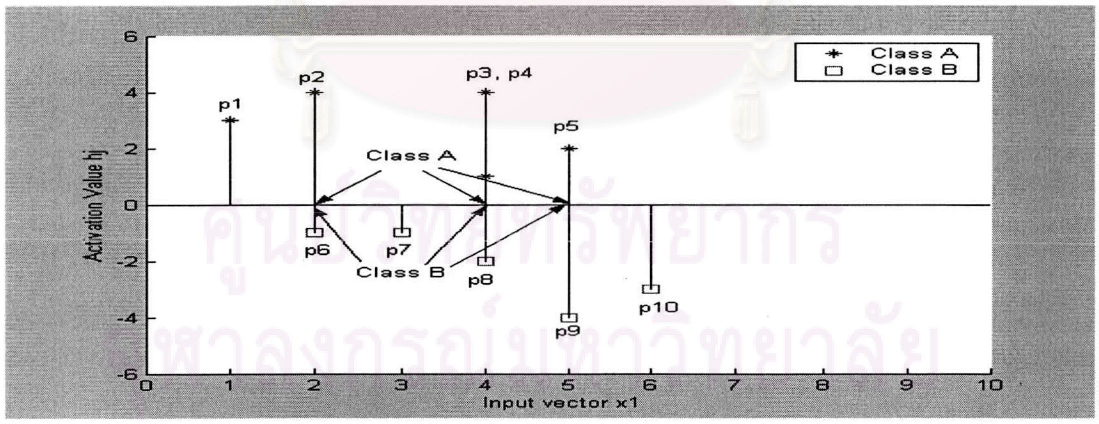
This implies that class A is defined by the interval [4,8] of x_2 while class B is defined by the interval [1,3] of x_2. In case of x_1, we can see that there are mixtures of positive and negative activation values in both class A and class B as illustrated in Figure 2.3(c). Hence, it is impossible to separately define intervals of x_1 for class A and class B.



(a)



(b)



(c)

Figure 2.3 Activation values projection. (a) Points in 2-dimensional space. (b) Point projection with h_j and x_2. (c) Point projection with h_j and x_1.

2.1.3 Basic Definitions for REAP Algorithm

The data are assured to be in Euclidean space. The relevant terminology to REAP is given as follows.

Let $[l_i,u_i]$ be an interval with respect to input feature x_i, where l_i is the lower bound and u_i is the upper bound. Interval $[l_i,u_i]$ defines *class A* if the activation values of all input vectors having x_i in $[l_i,u_i]$ have the same sign and *class B (non-class A)* if the activation values of all input vectors having x_i in $[l_i,u_i]$ have the opposite sign to that of class A. The input vectors of class A may be scattered into several intervals and are determined by a set of some input features. In this situation, the conditions for testing whether an input vector is in class A can be written in the following formats.

Let a be the number of intervals. If class A is defined by a set $\{[l_i^{(s)},u_i^{(s)}] \mid 1 \leq s \leq a\}$ such that $[l_i^{(s)},u_i^{(s)}] \cap [l_i^{(t)},u_i^{(t)}] = \varnothing$; $1 \leq s, t \leq a$ and $s \neq t$ with respect to input feature x_i, then the condition of an input feature to be in class A can be written as $[l_i^{(1)},u_i^{(1)}]$ + $([l_i^{(2)},u_i^{(2)}]$ +…+ $([l_i^{(a)},u_i^{(a)}]$. The notation + denotes the logical OR.

Let n be an input feature dimension. For any class, say class A, let $[l_i,u_i]$ be an interval with respect to input feature x_i, if class A can be defined by a set of intervals of input features $\{x_i \mid 1 \leq i \leq n\}$, then the condition of an input feature to be in class A can be written as $([l_1,u_1] * [l_2,u_2] * \dots * [l_n,u_n])$ where * denotes logical AND.

Let $\{[l_i^{(1)},u_i^{(1)}], [l_i^{(2)},u_i^{(2)}], \dots, [l_i^{(aj)},u_i^{(aj)}]\}$ be a set of size aj of intervals of input feature x_i. If class A can be defined by a set of intervals of input features $\{x_i \mid 1 \leq i \leq n\}$ then the conditions of an input feature to be in class A can be written as $([l_i^{(1)},u_i^{(1)}] + [l_i^{(2)},u_i^{(2)}] + \dots + [l_i^{(a1)},u_i^{(a1)}]) * ([l_i^{(1)},u_i^{(1)}] + [l_i^{(2)},u_i^{(2)}] + \dots + [l_i^{(a2)},u_i^{(a2)}]) * \dots * ([l_i^{(1)},u_i^{(1)}] + [l_i^{(2)},u_i^{(2)}] + \dots + [l_i^{(an)},u_i^{(an)}])$ where * denotes logical AND and + denotes logical OR.

Definition 1.

A crisp rule for classifying class A is a rule expressed in "**If** condition **Then** conclusion" format. The value of the condition is projected on the input features. Let $\{[l_i^{(1)},u_i^{(1)}]$, $[l_i^{(2)},u_i^{(2)}]$, ... , $[l_i^{(aj)},u_i^{(aj)}]\}$ be a set of size aj of intervals of input feature x_i. If class A can be defined by a set of intervals of input feature $\{x_i \mid 1 \le i \le n\}$ then the conditions are addressed by the intervals of the input features for class A in an n-dimensional space in the following format.

"**If** $([l_i^{(1)},u_i^{(1)}]$ or $[l_i^{(2)},u_i^{(2)}]$ or ... or $[l_i^{(a1)},u_i^{(a1)}])$ and
$([l_i^{(1)},u_i^{(1)}]$ or $[l_i^{(2)},u_i^{(2)}]$ or ... or $[l_i^{(a2)},u_i^{(a2)}])$ and .... and
$([l_i^{(1)},u_i^{(1)}]$ or $[l_i^{(2)},u_i^{(2)}]$ or ... or $[l_i^{(an)},u_i^{(an)}])$
Then class A."

An example of some crisp rules from a 3-dimensional space for input features x_1, x_2 and x_3 shown in Figure 2.4, where variables $a < b < c < d$ for x_1, $e < f < g < h$ for x_2, and $i < j$ input for x_3.

"**If** $((a \le x_1 \le b)$ or $(c \le x_1 \le d))$ and
$((e \le x_2 \le f)$ or $(g \le x_2 \le h))$ and
$(i \le x_3 \le j)$
Then class A".

Note that the crisp rule for input features x_1, x_2 and x_3 shown in Figure 2.4 can also be written in the following format.

"**If** $((a \le x_1 \le b)$ and $(e \le x_2 \le f)$ and $(i \le x_3 \le j))$ or
$((a \le x_1 \le b)$ and $(g \le x_2 \le h)$ and $(i \le x_3 \le j))$ or
$((c \le x_1 \le d)$ and $(e \le x_2 \le f)$ and $(i \le x_3 \le j))$ or
$((c \le x_1 \le d)$ and $(g \le x_2 \le h)$ and $(i \le x_3 \le j))$
Then class A".

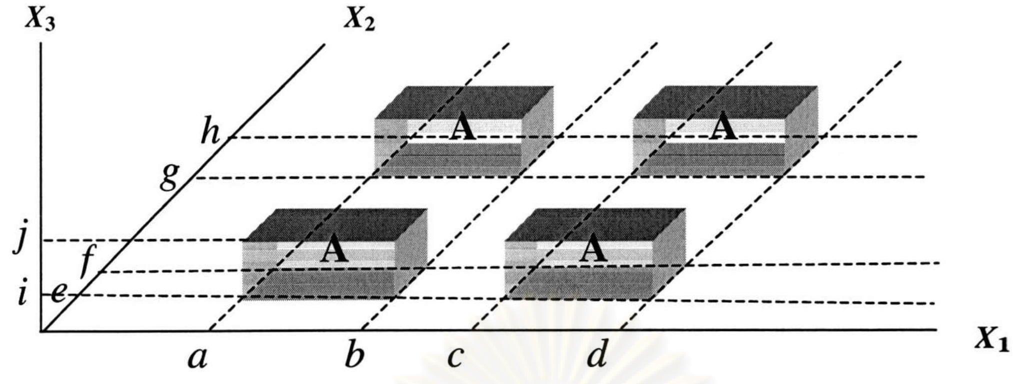


Figure 2.4 The example of crisp rules for class A.

Definition 2.

If the activation value with respect to a particular value of an input feature x_i for every input vector $\boldsymbol{p_k}$ is either positive or negative in the interval $[l_i,u_i]$, where l_i is the lower bound and u_i is the upper bound, then the interval $[l_i,u_i]$ is called the *unambiguous activation interval* with respect to input feature x_i.

Definition 3.

If there is at least one mixture between the positive and negative activation values of every input vector $\boldsymbol{p_k}$ in the interval $[l_i,u_i]$, then the interval $[l_i,u_i]$ is called *the ambiguous activation interval* with respect to input feature x_i.

Definition 4.

Let S_i^+ be the set of intervals on input features x_i consisting of all positive activation values for all patterns and S_i^- be the set of intervals on input features x_i consisting of all negative activation values for all patterns. X_i is called a *useful feature* if $| S_i^+ \cap S_i^- | < \omega$, where ω is a pre-specified constant. Otherwise, x_i is called a *useless feature*.

2.1.4 REAP Algorithm

REAP algorithm consists of two processes: *(i) a neural network training process* and *(ii) a class interval extraction of each input feature process*. In process (i), the neural network training process, a standard backpropagation neural network with one hidden layer is employed. The network structure has three layers: one input layer, one hidden layer, and one output layer. The value of each input vector is a numeric number and not limited only to binary values as those in M-of-N rule [7]. Unlike the KBANN [8], no domain knowledge is required to modify the links of the hidden nodes. The network will classify the input data into two groups. The first group is the interested class for rule base extraction while the second group is the remaining classes not in the first group. Suppose we have three classes of input data, namely, class A, class B, and class C. We need three networks for extracting the rules. The first network is the network for extracting the rules for class A. In this case, the output values for the patterns in class A are set to 1 and the output values for the patterns in the other classes (class B and class C) are set to 0 as shown in Figure 2.5(a). The second network is the network for extracting the rules for class B. In this case, the output values for the patterns in class B are set to 1 and the output values for the patterns from the other classes (class A and class C) are set to 0 as shown in Figure 2.5(b). And the third network is the network for extracting the rules for class C. In this case, the output values for the patterns in class C are set to 1 and the output values for the patterns from the other classes (class A and class B) are set to 0 as shown in Figure 2.5(c). So the number of networks to be trained is equal to the number of classes. The reasons for separating the network into individual class are to increase the classification accuracy and to use the activation value of each hidden node and its weight link to classify each class only.

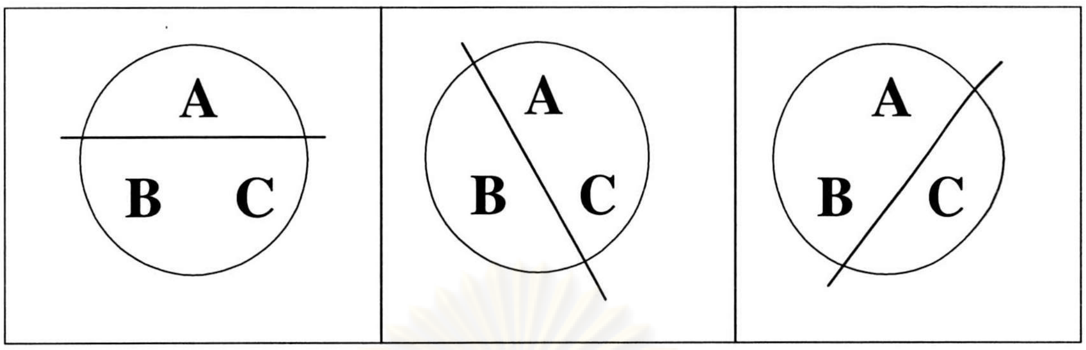


Figure 2.5 The classification schemes. (a) For class A. (b) For class B. (c) For class C.

The detail of REAP algorithm is given in Algorithms 1, 2, and 3. Algorithm 1 is for training the network and generating the weight links of all hidden neurons. In process (ii), the class interval extraction is determined and all extracted rules are generated using the class interval. The detail of extraction process is given in Algorithm 2.

```
Algorithm 1.   (Neural network training)
1. Define a 3 layer neural network.
2. Repeat
3.          Feed the input vectors to the network.
4.          Adjust weight of each link using the error backpropagation learning rules.
5. Until    the sum squared error between the targets and outputs of the network $\leq \tau$
            or the number of training iterations exceed $R$.
6. If       sum squared error between target and output of the network $> \tau$
7. Then     goto step 1.
8. Else     Accept the weight links of the trained network.
```

Algorithm 2. (Extract crisp rule using activation projection value)

```
1.  Set the threshold value ω for the useless input features.
2.  For each hidden neuron $D_j$, $1 \le j \le r$, where $r$ is the number of hidden
    neurons.
3.      For each input vector $\boldsymbol{p_k}$, $1 \le k \le m$, where $m$ is the number of patterns
4.          Do Compute the activation value of each $\boldsymbol{p_k}$
5.      End For
6.  End For
7.  For each input feature $x_i$, $1 \le i \le n$, where $n$ is the number of input dimensions
8.      For each hidden neuron $D_j$, $1 \le j \le r$
9.          For each input vector $\boldsymbol{p_k}$, $1 \le k \le m$
10.             If there is some mixture between the positive and negative
                   activation values of every input vector $\boldsymbol{p_k}$ in the interval $[l_i,u_i]$.
11.             Then Do Algorithm 3.
12.             If the mixture between the positive and negative activation values of
                   every input vector $\boldsymbol{p_k}$ in the interval $[l_i,u_i]$ is greater than a
                   threshold $\omega$.
13.             Then omit the input feature $x_i$ to form the crisp rule.
14.             Form the crisp rules for class A where the premises are
                   "($[l_i^{(1)},u_i^{(1)}]$ or $[l_i^{(2)},u_i^{(2)}]$ or ... or $[l_i^{(a1)},u_i^{(a1)}]$) and
                   ($[l_i^{(1)},u_i^{(1)}]$ or $[l_i^{(2)},u_i^{(2)}]$ or ... or $[l_i^{(a2)},u_i^{(a2)}]$) and .... and
                   ($[l_i^{(1)},u_i^{(1)}]$ or $[l_i^{(2)},u_i^{(2)}]$ or ... or $[l_i^{(an)},u_i^{(an)}]$)"
15.         End For
16.     End For
17. End For
```

Some input feature x_i can be omitted from forming a rule by considering the number of input vectors and their activation values. With respect to input feature x_i, if the difference between the number of input vectors with positive activation values and the number of input vectors with negative values is greater than a preset threshold ω, then x_i

is omitted from forming any rule. From Figure 2.3(c), if we set the threshold ω to 40%, in this case, the crisp rule will omit x_i from forming a rule. Because the activation values in the interval [1,6] for x_1 have 50% (5 data points) of positive activation values and 50% (5 data points) of negative activation values. Hence, it is impossible to define the intervals of x_1 for class A and class B, and x_1 must be omitted from the crisp rule. This process is the feature extraction technique in data mining application to reduce the number of dimensions of the input features by eliminating the useless feature.

If some input vectors $\boldsymbol{p_k}$ whose activation values with respect to a particular value of an input feature x_i have some ambiguous intervals, then these input vectors $\boldsymbol{p_k}$ will be left out from the rule checking process because their conditions will not satisfy any rules. To solve this problem, these ambiguous activation intervals must be considered during the rule extraction process.

For example, suppose in the interval $[l_i,u_i]$, there is 2% of negative activation values and there is 98% of positive activation value. In this case, it concluded that there is 2% of ambiguous interval for the interval $[l_i,u_i]$. Algorithm 3 is used to extract the crisp rules with ambiguous activation interval. For example, if the value ψ is set to 2% of ambiguous interval, then 2% of negative activation values with 98% of positive activation values are allowed for crisp rules. On the other hand, 2% of positive activation values with 98% of negative activation values are allowed for crisp rules. The ψ is used to allow the noise to be accepted to form the crisp rule (this value should be small around 1-5%).

Note that the maximum number of accepted ambiguous activation values ψ and the threshold value ω are not the same value. 6+ The ω is used for the *feature extraction* purpose to eliminate the useless input feature as describe in Definition 4 (this value should be around 40-50%).

```
Algorithm 3. (Rule extraction with ambiguous activation values)
1.  Find all intervals $[l_i,u_i]$ which are unambiguous and ambiguous, where $l_i$ is the
    minimum value and $u_i$ is the maximum value.
2.  Let $\psi$ be the maximum number of accepted mixture between minus sign
    and plus sign activation values in the interval.
3.  $J = 2,\ I = 1,\ K = 0$
4.  For each $\mathbf{x}_i$
5.          Repeat
6.              $[l_i^{(I)},u_i^{(I)}] = [l_i^{(I)},u_i^{(I)}] \cup [l_i^{(J)},u_i^{(J)}]$ and update $l_i$ and $u_i$
7.              If $[l_i^{(J)},u_i^{(J)}]$ had the mixture between minus sign and plus sign
                activation values
8.              Then $K = K + 1$
9.              $J = J + 1$
10.         Until $K > \psi$ or no more interval
11.         $I = I + 1$
12.         $J = I + 1$
13. End For
```

The time of an REAP algorithm is $m{\cdot}n{\cdot}h$ or $O(m{\cdot}n{\cdot}h)$, where m is the number of input vectors, n is the input dimensions, and h is the number of hidden nodes. The number of hidden neurons can be reduced by pruning those neurons that give all positive values or all negative values to all input features regardless of their classes. This pruning eliminates the non-separating neurons and will provide the small network with less hidden neurons. Note that the proposed REAP algorithm can be applied to the data set to project on each dimensional space.

2.2 Extraction Certainty Factor (RECF) Algorithm

The main reason to do this process is to provide more information about the certainty factors of the rules to the user. No further training is needed. The RECF algorithm calculates the certainty factor values from the crisp rules. The related definitions to RECF are given as follows.

2.2.1 Basic Definitions for RECF Algorithm

Definition 5.

A *certainty factor* with respect to class A, denoted by CF_A, is the percentage of all input vectors of class A classified correctly under the crisp rules from all the data set.

The certainty factor is used to measure the precision of the crisp rules and how accurate the rule is performed. The CF_A has the value between 0 to 100. For example, if the CF_A of class A equals to 90%, this means that there are 90 input vectors out of 100 input vectors correctly classified as in class A. Therefore, the higher the CF_A is, the more certainty of rule for classifying the data in the class A will be.

Definition 6.

A *non-vague input-feature interval* is the input feature interval that covers only one class of input vectors.

Definition 7.

A *vague input-feature interval* is the input feature interval that covers more than one class of input vectors.

There may be some vague input feature interval in any class. As illustrated in Figure 2.6, the non-vague input feature intervals are $[a,b]$, $[c,d]$ for x_1 and $[e,f]$, $[g,h]$ for x_2 while the vague input feature intervals are $[b,c]$ for x_1 and $[f,g]$ for x_2. These particular vague input feature intervals are considered to be the overlapped intervals of both class A and class B.

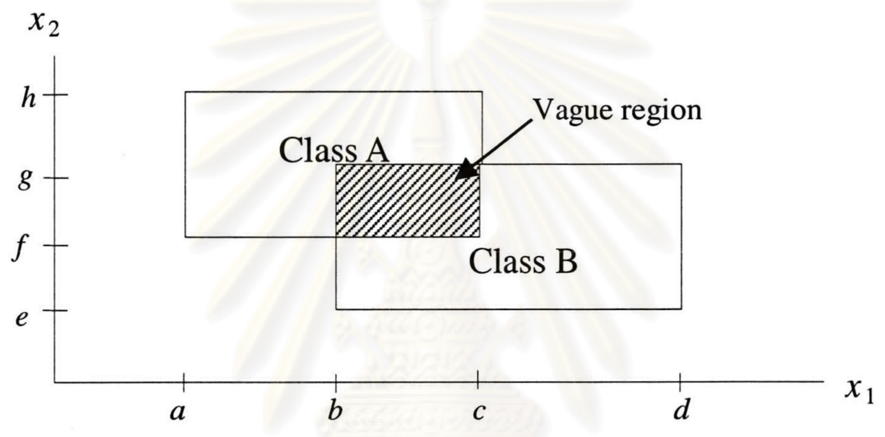


Figure 2.6 The vague region.

Definition 8.

A *CF rule* is the extension of a crisp rule with a certainty factor included in the conclusion. The CF rule format is

"**If**	conditions of class A	
Then	class A	CF_A = CF value of class A".

Definition 9.

A *definite rule* of class A is a CF_A rule which has the CF_A value equaled to 100.

The definite rule obtained by a non-vague region is a very useful piece of knowledge because the given intervals are definitely in the class. If the CF_A equals to

100, then no input vectors of other classes are within the assigned interval for class A. From Figure 2.6, we can generate the definite rules as follows:

rule(1) **If** ($a \leq x_1 \leq b$ and $f \leq x_2 \leq h$) or ($b \leq x_1 \leq c$ and $g \leq x_2 \leq h$)
Then class A $CF_A = 100$.

rule(2) **If** ($b \leq x_1 \leq c$ and $e \leq x_2 \leq f$) or ($c \leq x_1 \leq d$ and $e \leq x_2 \leq g$)
Then class B $CF_B = 100$.

Suppose that the vague region has 80% of data in class A and 20% of data in class B. We can generate the CF rule as follows:

rule(3) **If** ($b \leq x_1 \leq c$ and $f \leq x_2 \leq g$)
Then class A $CF_A = 80$.

rule(4) **If** ($b \leq x_1 \leq c$ and $f \leq x_2 \leq g$)
Then class B $CF_B = 20$.

Note that if a value of an input feature generates more than one conclusion, then the highest CF value is selected as an answer. For example, if the conclusion from a rule is class A with $CF_A = 90$ and the conclusion from another rule is class B has $CF_B = 60$, then the conclusion for this situation is class A with $CF_A = 90$.

2.2.2 RECF Algorithm

In the crisp rules, it is possible to compute the accuracy of each rule. By noticing the overlapping input vector intervals of the result given in the crisp rules, we try to eliminate the overlapping input vector intervals to produce a new set of rules and make the rule to give a higher accuracy rate of some rules. Since CF_A is the percentage of all input vectors of class A correctly classified under the crisp rules and has the value between 0 to 100, the RECF algorithm is the process to produce the definite rules where the intervals have CF_A value equaled to 100. In the first step, we need to find the vague intervals from the crisp rules and assign the intervals for the definite rules with CF_A value equals to 100. Next step is to compute the CF_A values for the vague input vector intervals and produce the CF_A rules. The CF_A value that is less than the preset threshold value ξ for accepting the certainty factor value will be eliminated. Suppose that, we set the threshold value equal to 50. From Figure 2.6, rule (4) can be eliminated because the CF_B value in this case is 20. This algorithm will give the CF rule base. The CF rules are composed of rule (1), rule (2), and rule (3). The detail of *RECF algorithm* is shown in Algorithm 4. The time of RECF algorithm is $m \cdot n$ or $O(m)$ where m is the number of input vectors and n is the number of input feature dimensions.

Algorithm 4. (Rule extraction with certainty factor)

```
1.   Set the threshold value $\xi$ for accepting the certainty factor value.
2.   For each input feature $x_i$, $1 \le i \le n$, where $n$ is the number of dimensions
3.      For each intervals given from the crisp rule.
4.         Find the overlapping interval for class A.
5.         Compute the accuracy of the input vectors of class A in each interval.
6.         Assign $CF_A$ value to each vague input feature interval.
7.         If     the $CF_A$s value is greater than the threshold value $\xi$.
8.         Then   Form a CF rule and assign the $CF_A$ equals to the accuracy
                  of the input vectors of class A.
9.         If     the $CF_A$s value = 100
10.        Then   Form a definite rule and assign the $CF_A$ value to 100.
11.     End For
12.  End For
```

2.3 Rule Extraction Natural Language (RENL) Algorithm

Rule extraction natural language algorithm uses natural language (NL) terms such as "small", "medium", or "large" in the rule base in order to make rules easy to understand. A user will have a conceptual relationship between the input feature values and the given class.

2.3.1 Basic Definition for RENL Algorithm

A rule expressed in a natural language has the same format as in the crisp rule but the condition of each x_i is expressed in terms of *linguistic quantities* such as “small”, “medium”, and “large”. The *NL rule* format is as follows.

“**If** conditions of class A expressed in natural language terms such as “small”, “medium”, and “large”
Then class A CF_A = CF value of class A”.

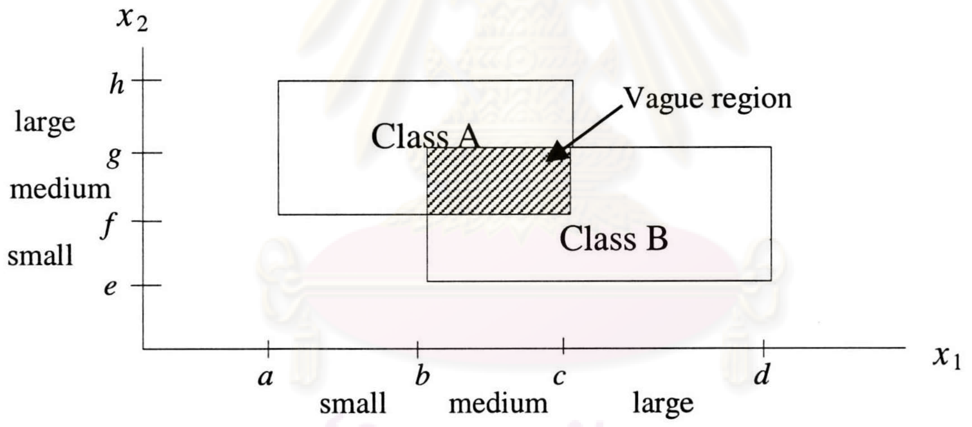


Figure 2.7 An example of the relation between the natural language terms and the values of each interval.

From Figure 2.7, we can devide x_1 into 3 ranges [a,b), [b,c), and [c,d), x_2 can be divided into 3 ranges [e,f), [f,g), and [g,h), and name them in NL terms as “small”, medium”, and “large”, respectively. The examples of the NL rules are as follows.

rule(1) **If** (x_1 is small and x_2 is medium or large) or (x_1 is medium and x_2 is large)
Then class A $CF_A = 100$.
rule(2) **If** (x_1 is medium and x_2 is small) or (x_1 is large and x_2 is small or medium)

Then class B $CF_B = 100$.

rule(3) **If** (x_1 is medium and x_2 is medium)

Then class A $CF_A = 80$.

2.3.2 RENL Algorithm

RENL algorithm is not based on the concept of fuzzy logic and fuzzy set where a membership function must be specified [25, 26]. The NL terms defined by a user. After dividing the NL terms, each interval is orderly assigned a quantitative term as shown in the following example.

Suppose the specified quantitative terms are small, medium, and large when the considered input features belong to $\boldsymbol{x}_i$. Since there are three quantitative terms, the values of input features of $\boldsymbol{x}_i$ must be divided into three subintervals: $[l_1,u_1]$, $[l_2,u_2]$, and $[l_3,u_3]$ where $u_1 < l_2$ and $u_2 < l_3$. Note that the intervals for natural language terms are not allowed the overlapping value. Each subinterval is assigned a quantitative term as follows:

$[l_1,u_1]$ is assigned term "small",

$[l_2,u_2]$ is assigned term "medium", and

$[l_3,u_3]$ is assigned term "large".

The detail of RENL algorithm is shown in Algorithm 5. When the number of intervals is greater than the number of NL terms, merging interval algorithm in Algorithm 6 is used so that all linguistic quantities can be deployed without introducing new linguistic terms. On the other hand, when the number of interval is less than the number of NL terms, then dividing interval algorithm in Algorithm 7 is used to split the existing intervals to deploy all linguistic terms.

Algorithm 5. (Rule extraction natural language algorithm)

```
1.  Let $NL\_Term$ = number of natural language terms.
2.  For each input dimensional space $x_i$, $1 \le i \le n$, where $n$ is the dimension.
3.      Let $M$ = the number of intervals for class A given from the CF rules.
4.      Let $I = \{[l_i^{(1)},u_i^{(1)}], [l_i^{(2)},u_i^{(2)}], \dots [l_i^{(M)},u_i^{(M)}]\}$ intervals for class A given
        from the CF rules.
5.      If    $M > NL\_Term$
6.      Then  Do Merge interval algorithm.
7.      If    $M < NL\_Term$
8.      Then  Do Divide interval algorithm.
9.      Assign each interval with a linguistic quantity.
10.     Form the NL rules using the natural language terms in step 9.
11. End For
```

Algorithm 6. (Merging interval algorithm)

```
1.  For $NL\_Term < M$
2.      $PD_j$ = (the number of data in class A of interval $j$)/(the number of all
          data in class A* (the area of interval $j$/the total area of all intervals))
3.      Let $CF_j$ = Certainty Factor of the interval.
4.      $Cost\_Function_j = PD_j * CF_j$
5.      Let $I_{min}$ = the interval with the minimum $Cost\_Function_j$
6.      If    $I_{min}$ is at the edge.
7.      Then  $I_{merge}$ = merge $I_{min}$ with the nearest interval.
8.      Else  $I_{merge}$ = merge $I_{min}$ with the adjacent interval whose cost value
                  is equal to minimum cost value of $I_{min}-1$ or $I_{min}+1$.
9.      Compute the new CF value of $I_{merge}$.
10.     $M = M - 1$
11. End for.
```

Algorithm 7. (Dividing interval algorithm)

```
1.  For $M < NL\_Term$
2.        $PD_j$ = (the number of data in class A of interval $j$)/(the number of all
              data in class A* (the area of interval $j$/the total area of all intervals))
3.        Let $CF_j$ = Certainty Factor of the interval.
4.        $Cost\_Function_j = PD_j * CF_j$
5.        Let $I_{max}$ = the interval with the maximum $Cost\_Function_j$
6.        Let $PD_{max}$ = the PD of $I_{max}$
7.        Divide $I_{max}$ into two parts until probability density value of $I_{divide1} = PD_{max}$
          and $I_{divide2} = PD_{max}$
8.        Compute the new CF values of $I_{divide1}$ and $I_{divide2}$.
9.        $M = M + 1$
10. End for.
```

However, how to select the interval to be merged or to be divided and to provide NL rules with a high accuracy is important. Let the *Probability Density* (PD) value of the interval equal to (the number of data in class A of each interval)/(the number of all data in class A* (the area of each interval/the total area of all intervals)). The criterions that will be used to select the divided or merged interval are the PD value and the CF value of each class. Let the *cost function* of any interval equal the multiplication of the probability density value and the CF value of its class. The input vector with a lower probability density value is less likely to occur and the lower CF value means the less certainty about the class. Then, the cost of the lower probability density value and the lower CF value is less likely to occur than the cost of the higher probability density value and the higher CF value. For merging interval algorithm, the selection of intervals should be represented by the event of the least likely to be happened or the minimum value of the cost function. But for dividing interval algorithm, the selection of intervals should be represented by the event most likely to be happened or the maximum value of the cost function. Suppose the dividing interval has the probability density value equals to PD, then the interval is divided into two subintervals having equal probability of PD.

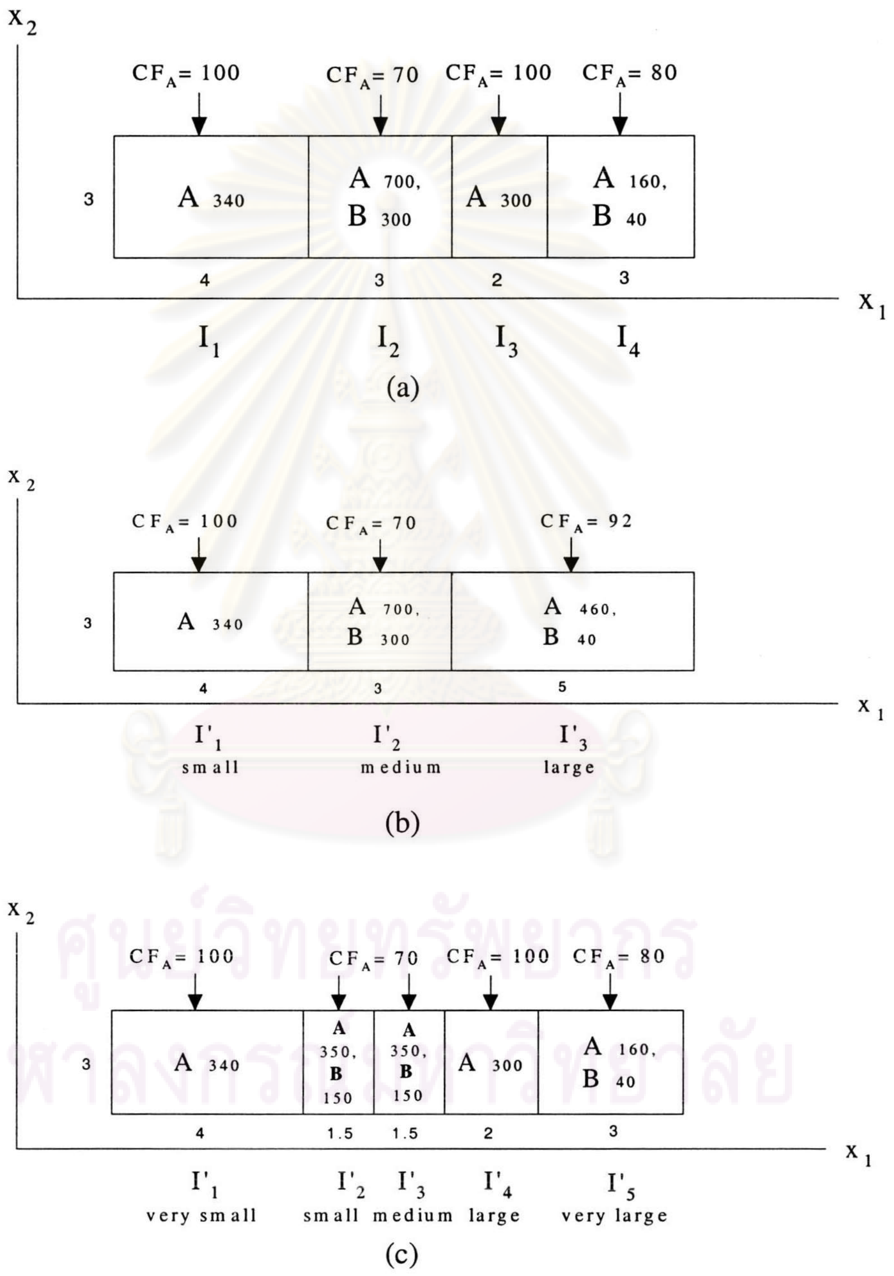


Figure 2.8 The NL rule base intervals. (a) The CF rule base intervals. (b) The merged intervals. (c) The divided intervals.

The NL rule intervals in Figure 2.8(a) illustrate how to calculate the cost function. In this figure, the CF rules give four intervals $\{I_1, I_2, I_3, I_4\}$ which the numbers of data in class A are {340, 700, 300, 160}, respectively. The total number of data in class A is 1500. The calculations of the probability density value of each intervals are {340/(1500×(4×3)/36), 700/(1500×(3×3)/36), 300/(1500×(2×3)/36), 160/(1500×(3×3)/36)} or {0.68, 1.87, 1.19, 0.43}, respectively. The CF values for *class A* are {100, 70, 100, 80}. The cost functions of each interval in$\{I_1, I_2, I_3, I_4\}$are {(0.68×100), (1.87×70),(1.19×100), (0.43×80)} or {68, 130.9, 119, 34.4}, respectively. The minimum value of the cost function is in I_4 and the maximum value of the cost function is in I_2.

Next, we will consider how to merge or divide the interval from the CF rules. If the number of NL terms is four, then we can assign each interval to have the same value as CF rules and name each intervals with NL terms (small, medium, large, and very large) as shown in Figure 2.8(a). However, if the number of NL terms is three (small, medium, and large), then we need to merge the intervals. In this case, the selection of intervals to be merged is the minimum value of the cost function I_4 where the merged interval is shown in Figure 2.8(b). On the other hand, if the number of NL terms is five (very small, small, medium, large, and very large), then we need to divide more intervals. In this case, the selection of intervals to be divided is the maximum value of the cost function I_2 where the divided interval is shown in Figure 2.8(c). From Algorithm 7, item 7, the selected interval is divided recursively into two parts until probability density value of $I_{divide1} = PD_{max}$ and $I_{divide2} = PD_{max}$. The time for dividing the interval into two parts is O(log m), where m is the number of input vectors. The time for RENL algorithm is $r \cdot \log m$ or $O(r \cdot \log m)$, where r is the number of intervals obtained from the crisp rules.