Chapter 3

Learning Algorithms

3.1 Generic Elliptic Radial Basis Function

There are many parameters for adjusting in a radial basis function learning algorithms while using a non-linear RBF model. The controllable parameters of this function are a center vector (which control a position of hidden node), a width vector (which control a size of the hidden node) and a rotational vector (which control a direction of the hidden node). To work with the classification problem, the network should be controlled by a special cost function. According to using of the multivariate Gaussian function, the controllable parameters are a center vector and a covariance matrix [5] that make the traditional algorithm has a high cost. But our network proposes the using of a generic elliptic radial basis functions, which reduce that cost by eliminating the covariance matrix operation and including all of these features (the controllable parameters). The following notations are used in our proposed radial basis function.

Let	$\mathbf{x}$	:	input vectors (i.e. $\{x_1, x_2, \dots , x_n\}$)
	$\mathbf{c}$	:	center vectors (i.e. $\{c_1, c_2, \dots , c_n\}$)
	$\mathbf{w}$	:	width vectors (i.e. $\{w_1, w_2, \dots , w_n\}$)
	$\mathbf{r}$	:	rotation matrix (i.e. $\begin{bmatrix} r_{11}, r_{12} \cdots r_{1n} \\ r_{21}\ r_{22} \cdots r_{2n} \\ \cdots \\ r_{n1}\ r_{n2} \cdots r_{nn} \end{bmatrix}$)
	n	:	dimensions
	i, j	:	indices of dimensions

The proposed elliptic function is given in equation 3.1. The output of the function is composedly fed to a sigmoid function as shown in equation 3.2. These two functions are combined as a generic elliptic radial basis function (GERBF). The elliptic activation function was presented as $h_k(\mathbf{c}, \mathbf{w}, \mathbf{r})$ for each neuron k and the sigmoid function was shown below as $y_k(h)$ for each neuron k too.

$$h_k(\mathbf{c}, \mathbf{w}, \mathbf{r}) = \sum_{i=1}^{n} \frac{\left(\sum_{j=1}^{n} r_{i,j} \left(x_j - c_j \right) \right)^2}{w_i^2} - 1 \qquad (3.1)$$

$$y_k(h_k) = \frac{1}{1 + e^{-\beta h_k}} \qquad (3.2)$$

Figure 3.2 demonstrates the effects of $\mathbf{c}$ and $\mathbf{w}$ on the shape of the GERBF function. The circle is the result of plotting GERBF when $\mathbf{w}_1 = \mathbf{w}_2$ while the ellipse is the result of plotting GERBF when $\mathbf{w}_1 > \mathbf{w}_2$ and $\mathbf{r} \neq \mathbf{I}$ (where $\mathbf{I}$ is an Identity matrix) with the same center vector $\mathbf{c}$.

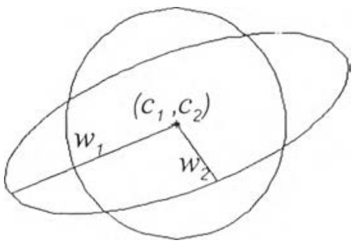

Figure 3.1 The figure of c and w

If we plot the composition function of $h_k(\mathbf{c},\mathbf{w},\mathbf{r})$ and $y_k(h)$ in three dimensions, the shape of this function looks like a multivariate Gaussian function. The steepness of the shape is controlled by the steepness constant β. The steepness of the function is increased accordingly to the value of β. In our algorithm, β is chosen randomly and used in each learning time for finding an optimal network (the network that used smallest number of hidden node).

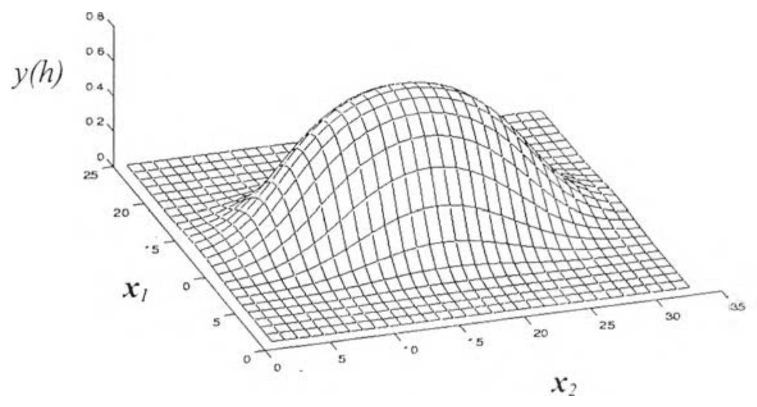

Figure 3.2 The figure of the composition function of $h_k(\mathbf{c},\mathbf{w},\mathbf{r})$ and $y_k(h)$

The rotation of our GERBF is controlled by $\mathbf{r}$. The translation is controlled by changing the value of $\mathbf{c}$ and the size is scaled by $\mathbf{w}$. The constant β is used to determine the steepness of our function. The output of the each data vector decreases (or increases) monotonically with the distance from a center vector. If the data vector is near the center vector, the output of this vector will close to one. On the other hand, the output is near zero if it locates far from the rim of the function. Then, we use this feature to classify the same class data vectors. If the data vectors are in the same class, the output of each one is close to one. So our network need a set of GERBFs for covering a class of data vector. In the same way, the output of the other class vectors should be nearly zero while they locate far from the rim of the set of GERBFs.

For the learning, the gradient descent of the cost function is used to control the variables $\mathbf{c}$, $\mathbf{w}$ and $\mathbf{r}$. From the meaning of covered data, the value of $y_k(h_k)$ should be nearly one. So our cost function is defined as follows.

$$c=\frac{1}{2}\left(1-\prod_{i=1}^{m}\phi_i\right)^2 \tag{3.3}$$

where m is the number of trained data vectors and ϕ_i is the output of GERBFs covering data vector $\mathbf{x}_j$. The cost function is minimal when every ϕ_i are equal to one. The parameters $\mathbf{c}$, $\mathbf{w}$ and $\mathbf{r}$ are adjusted by the gradient descent learning rule.

3.2 Learning Algorithm

Our learning algorithm is based on the learning of RAN [2]. To follow the main idea of RAN, we begin our algorithm without any hidden node in hidden layer. The learning is processed until all data in the same class are covered. To learn any data set, we pick up a class of input data set randomly first. Suppose the data set is in class A. A first data vector that was selected randomly from A is performed as an introducer of the first GRBF to the network. Then, the other same class data vector are randomly selected and fed sequentially one by one to the network. If it is not already covered by any hidden node then the adjusting will be started to cover this data vector. The shape of the nearest hidden node is stretched to cover it by adjusting the parameters $\mathbf{c}$, $\mathbf{w}$ and $\mathbf{r}$. The adjusting is done step by step by a gradient descent of the cost function. Enlarging a GERBF shape may cover the data in the other classes. If this even occurs, the enlarging process for this GERBF should be stopped and the size of the hidden node is shrunk instead. For example, the shrinking of w_l of the adjusted hidden node, which is an element of the width vectors, should be decreased. The delta value (the small value that is used to shrinking in each step of learning) is computed by a small ratio of the old w_l. If the shrinkage is not successful, a new GERBF neuron will be introduced to the network. Suppose that the data vectors in class A will be covered first. Let s be a constant greater than one.

Covering Algorithm

```
1.   Let $n = 1$
2.   Introduce a GERBF neuron, $f_a$. and initialize its parameters ($\mathbf{c}, \mathbf{w}, \mathbf{r}, \beta$).
3.   Select the first data vector randomly and set the first GERBF neuron $f_1$ to
     cover this vector by set $c_1 = x_1$.
4.   While there are some data vectors in class $A$ still not trained do
5.       Feed a new data vector $x_i$ that is selected randomly
6.       If $x_i$ is not covered by $f_a$ then
7.           Try to stretch the size of $f_a$ to cover $x_i$
8.           Let T = 1
9.           While there is a data vector from other classes covered
                 by this function and $T < s$ do
10.              Shrink the size of $f_a$ and T = T + 1
11.          End While
12.          If $T > s$ then
13.              $n = n + 1$ and introduce a new GERBF neuron
14.          End If
15.      End If
16.  End While
```

Figure 3.3 shows the process of adjusting the size of a GERBF to cover the data in class A. Each number indicates the sequence of size adjusting.

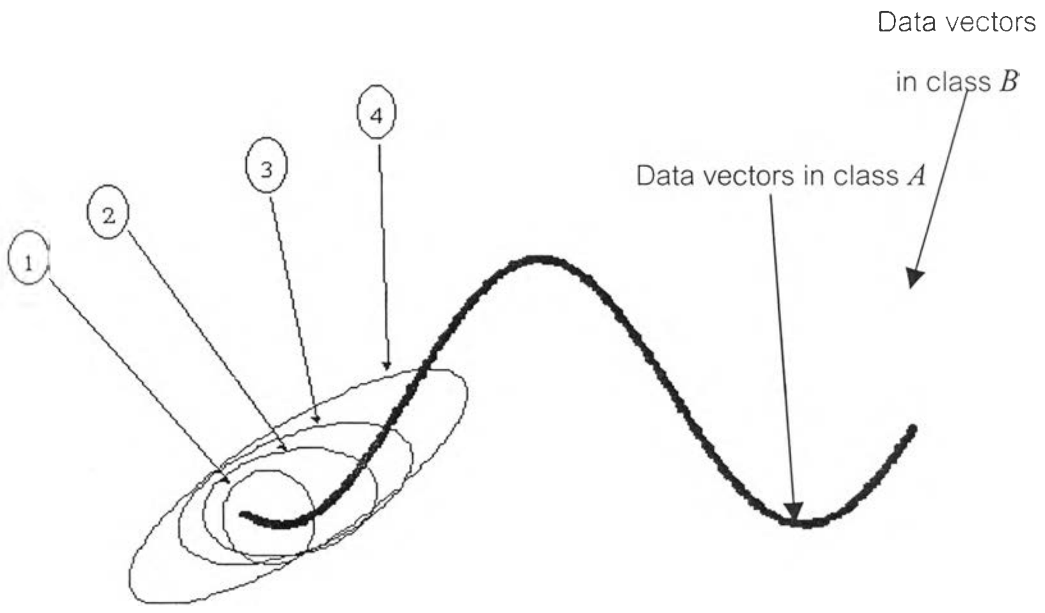

Figure 3.3 The enlargement of training hidden node in learning season.

3.3 Pruning Algorithm

After the learning process, the network may have the many redundant hidden nodes. A redundant GERBF node is the node whose all of its data vectors are covered by some other GERBF nodes. The pruning procedure is needed the reduced of the redundant nodes. Since the problem of pruning all redundant neurons can be viewed as a problem of finding minimum set cover [8], any existing heuristic algorithm can be applied to this problem. Here, we propose the following pruning algorithm.

Pruning Algorithm

```
1.  For each GERBF hidden node $f_a$
2.      If $f_a$ is not a first consideration or it has not been pruned, Then
3.          For each data vector in this hidden node
4.              If this data vector is covered by some other hidden node too, Then
5.                  Feed a next hidden node. /* This hidden does not be pruned. */
6.              End If
7.          End For
8.          Prune this hidden node.
9.      End If
10. End For
```

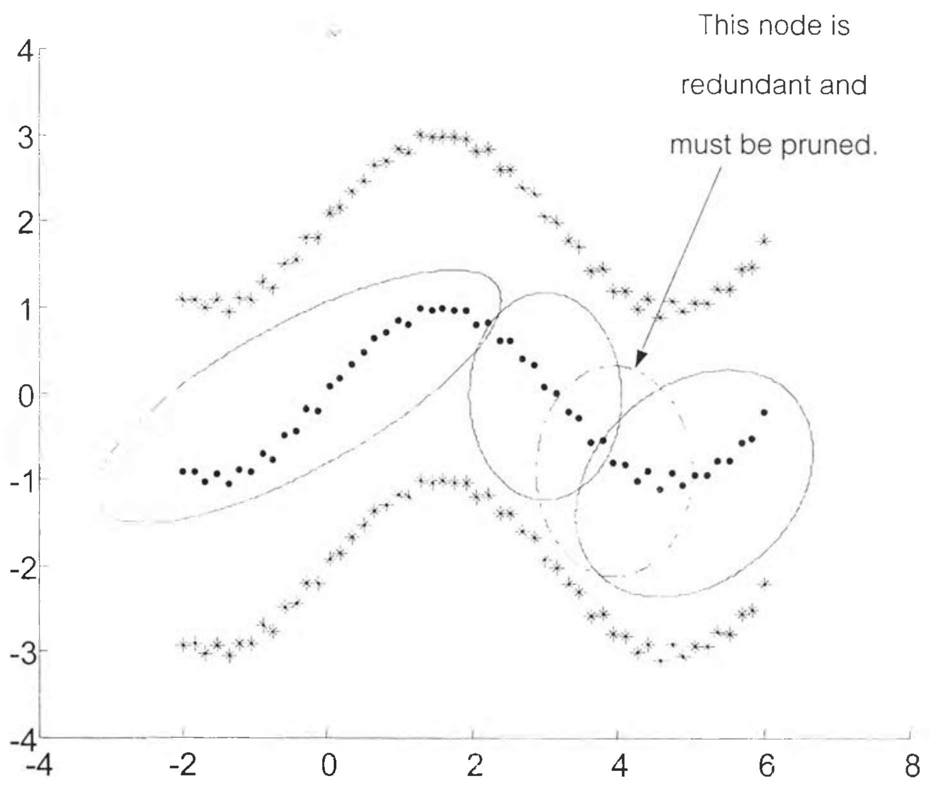

Figure 3.4 Pruning the redundant hidden.

Figure 3.4 shows an example of a redundant hidden node. Each ellipse presents the position of each hidden node, which is allocated after we finish the covering algorithm. The dashed ellipse presents the redundant hidden node that should be pruned.

3.4 Generalization Algorithm

The generalization of a GERBF network means the ability to correctly classify a new incoming data vector. This implies that all untrained vectors must properly be covr ed by some GERBF neurons. To make this coverage possible, the size of each GERBF neuron should be adjusted accordingly to the natural distribution of the untrained vectors and the location of each GERBF center must be estimated and relocated. Figure 3.5 shows an idea of estimating a new adjusted and relocated hidden node. All data vectors are in a 2-dimensional space and are shown by symbols "*" and "+" corresponding to their coordinates in x and y axes. The data of Class1 is the training data set and the data of Class2 is the other class data set.

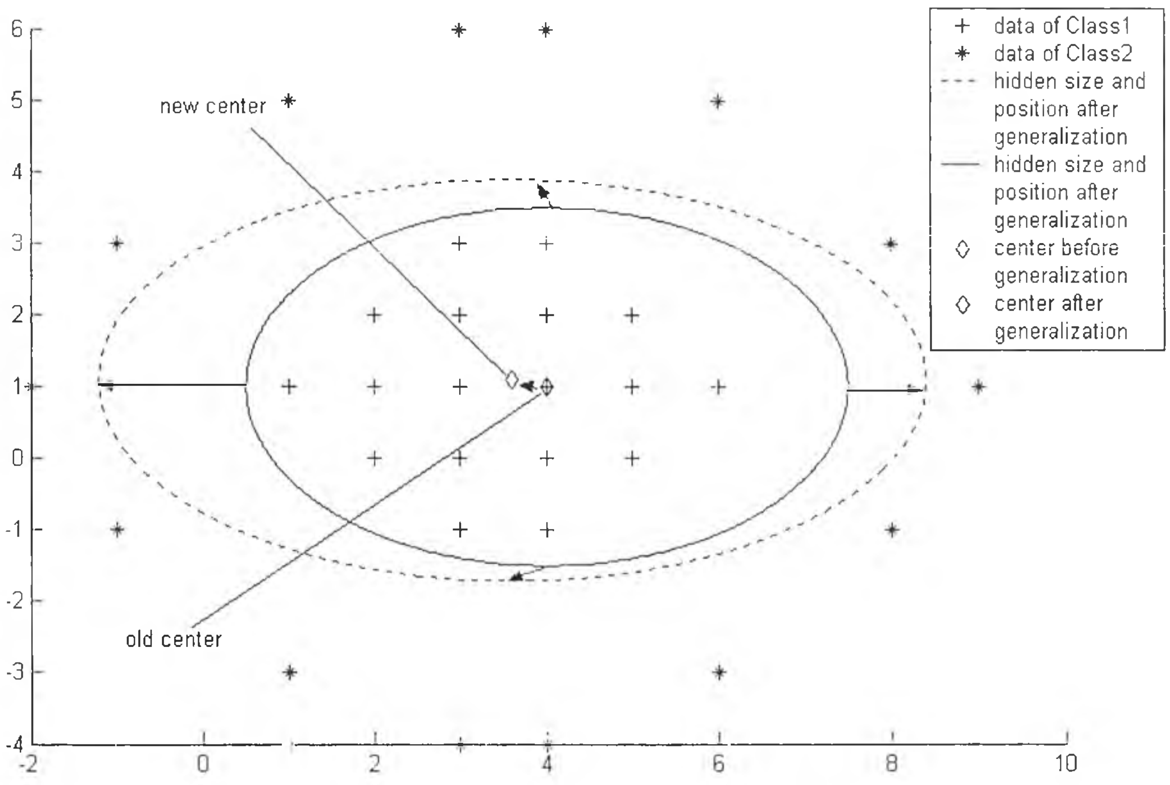

Figure 3.5 Generalization Idea.

We apply the technique of Bootstrap [5], [6], and [7] to estimate the mean as well as the variance of the size and find the new center of each GERBF. The estimation must be performed on both classes, the class covered by the GERBF and the class outside the GERBF. The details of the algorithms are given as follows. Let T be a constant, x_j the data vectors covered by g_a, N_{g_a} the number of data vectors covered by g_a, and C_{g_a} the new estimated center of GERBF g_a.

Estimating Center Location Algorithm

```
1.  For each GERBF $g_a$ do
2.      $l = 1$
3.      While $l < T$ do
4.          Let $k > 0$ be a random integer.
5.          Randomly select a set $S_l$ of $k$ data vectors covered by $g_a$.
6.          Compute the mean center
                $c_l = \frac{1}{k}\sum_{x_j \in S_l} \mathbf{x}_j$.
7.          $l = l + 1$.
8.      End While
9.      Let $C_{g_a} = \frac{1}{T}\sum_{l=1}^{T} c_l$.
10. End For
```

For all data vectors in g_a, the mean as well as the variance of the GERBF size are estimated in terms of the average distance and the variance of the distance with respect to the C_{g_a}. Let d_j be the Euclidean distance between C_{g_a} and $\mathbf{x}_j$ (where $\mathbf{x}_j$ are the data vectors that covered by g_a), D_{g_a} the estimated mean of the distances of all data vectors in g_a, and V_{g_a} the estimated variance of the distances of all data vectors in g_a.

Estimating Size Algorithm

```
1. For each GERBF $g_a$ do
2.     Compute $C_{g_a}$
3.     $l = 1$
4.     While $l < T$ do
5.         Let $k > 0$ be a random integer.
6.         Randomly select a set $S_l$ of $k$ data vectors covered by $g_a$.
7.         For each $\mathbf{x}_j$ in $S_l$ do
8.             Compute $d_j$.
9.         End For
10.        Let $\bar{d}_l = \frac{1}{k}\sum_{j=1}^{k} d_j$.
11.        $l = l + 1$.
12.    End While
13.    Let $D_{g_a} = \frac{1}{T}\sum_{l=1}^{T} \bar{d}_l$.
14.    For each $x_j$ covered by $g_a$ do
15.        Compute $d_j$
16.    End For
17.    Compute variance $V_{g_a} = \frac{1}{(N_{g_a} - 1)}\sqrt{\sum_{j=1}^{N_{g_a}} \left(d_j - D_{g_a}\right)^2}$
18. End For
```

Figure 3.6 shows an example of how the estimating size algorithm works. The small ellipse at the center of the figure is the size of D_{g_a} (where plotted by using D_{g_a} as the width of the GERBF to show the approximation size of D_{g_a}). The middle ellipse is the variance of training data set of class A, $V_{g_a^{(A)}}$, which covered by $g_a^{(A)}$ and the largest ellipse is the variance of the other class data set (class B), $V_{g^{(B)}}$, which covered by $g^{(B)}$.

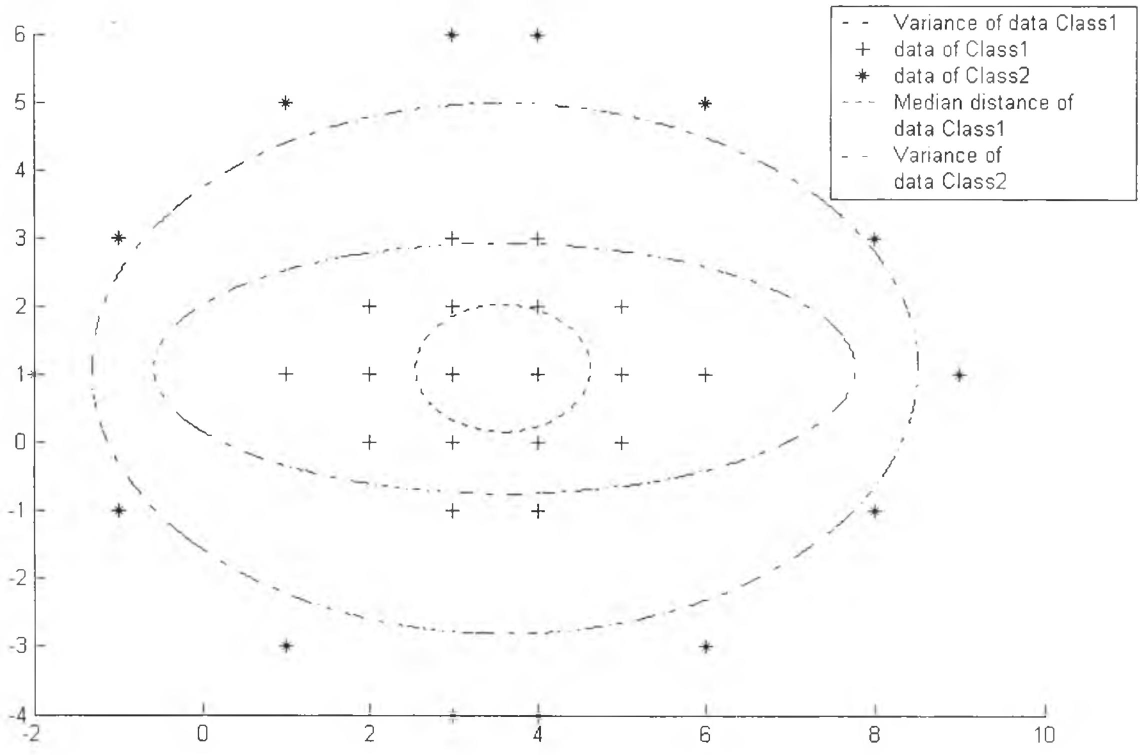

Figure 3.6 The variances of two classes of sample data set.

Suppose that class A is the considered class. To separate class A from class B , some GERBF neurons are required to cover the data vectors in class A and there is no need to use any GERBF to cover the data vectors in class B . In the other word, all data in class B can be considered as covered by only one large GERBF neuron. Hence, the above algorithm can be applied to both classes without any modification. The size adjustment must be performed on both classes. There may be the case that the estimated size of a GERBF in one class will be expanded over its neighboring GERBF in another class. To avoid this situation, a compensation of variances of both classes must be established. Let $g_a^{(A)}$ be a GERBF neuron covering the data vectors in class A and $g^{(B)}$ a GERBF neuron covering data vectors in class B . $g_a^{(A)}$ and $g^{(B)}$ are adjacent.

Let $w_{g_a^{(A)}}$ be the width of $g_a^{(A)}$ prior to applying the Bootstrap estimation. The width $w_{g_a^{(A)}}$ is adjusted by the following variance compensation.

$$\Delta V_{g_a^{(A)}} = V_{g_a^{(A)}} - V_{g^B} \tag{3.4}$$

$$\alpha = \frac{V_{g_a^{(A)}}}{V_{g^{(B)}}} \tag{3.5}$$

$$w^{new}_{g_a^{(A)}} = w^{old}_{g_a^{(A)}} + \alpha \times \Delta V_{g_a^{(A)}} \tag{3.6}$$

Figure 3.7 shows the approximated distribution size after generalization with enlarging by the variance ratio α (the middle ellipse) which is created by equation 3.6. The outer ellipse presents the variance of the data set in class B and the middle ellipse presents the estimated variance of the training data set of class A.

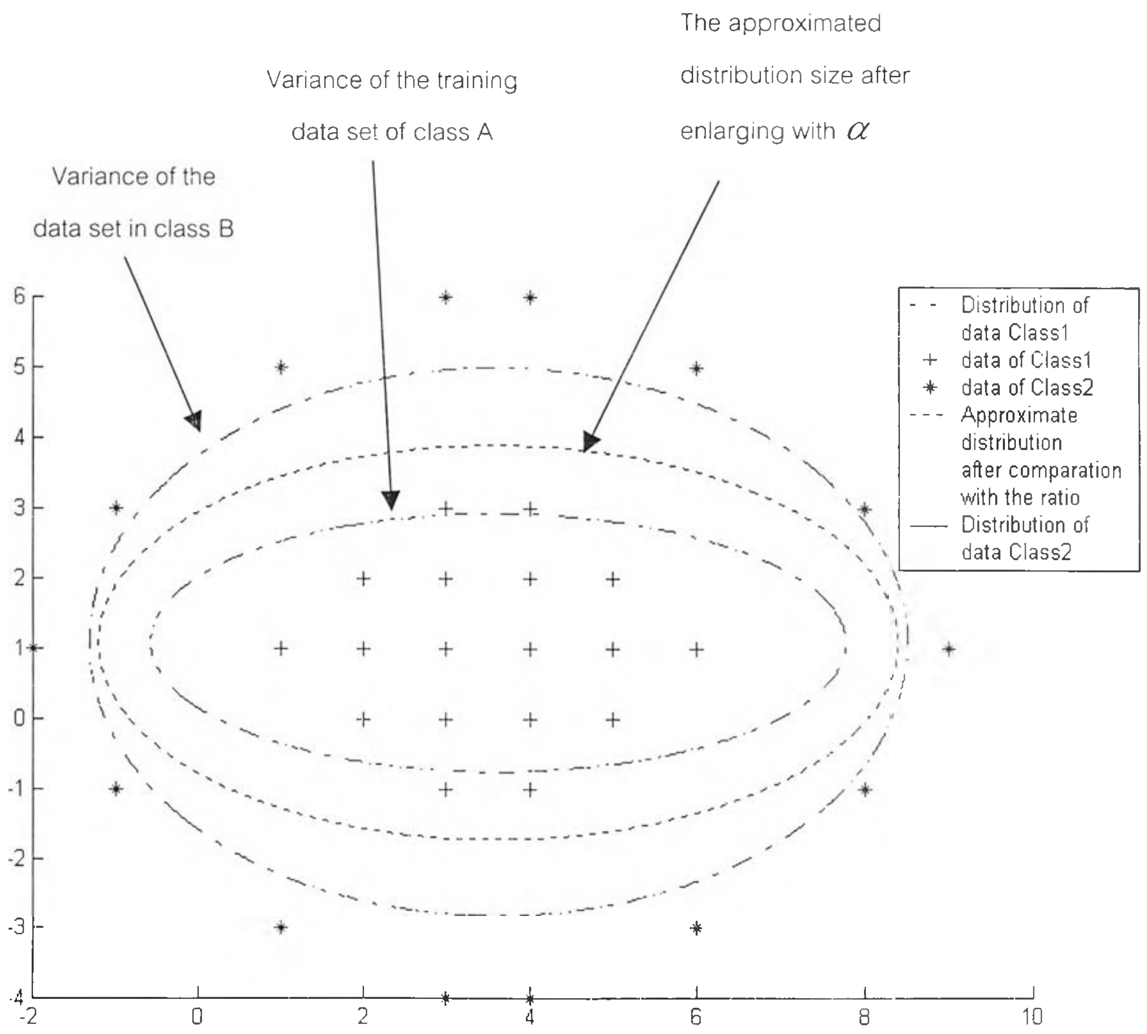

Figure 3.7 The approximated size after enlarging with the ratio.

Figure 3.8 shows the result of generalizaiion. The dashed ellipse comes from the middle ellipse in Figure 3.7 (generalized ellipse). The middle ellipse is the original size and position.

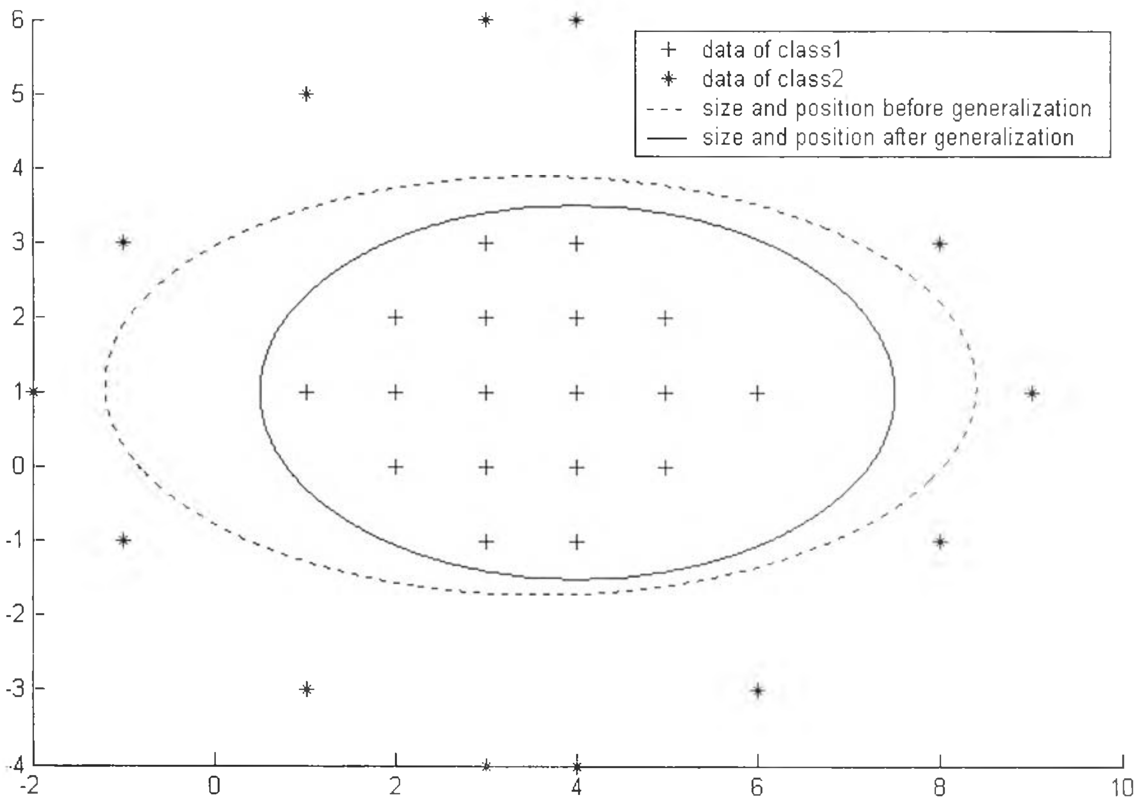

Figure 3.8 The comparison of the sizes of the GERBF before and after generalization.