A Geometrical Representation of McCulloch–Pitts Neural Model and Its Applications

Ling Zhang and Bo Zhang

***Abstract*— In this paper, a geometrical representation of McCulloch-Pitts neural model is presented. From the representation, a clear visual picture and interpretation of the model can be seen. Two interesting applications based on the interpretation are discussed. They are 1) a new design principle of feedforward neural networks and 2) a new proof of mapping abilities of three-layer feedforward neural networks.**

***Index Terms*—Feedforward neural networks, measurable functions, neighborhood covering.**

I. Introduction

In 1943, McCulloch and Pitts [1] first presented a mathematical model (M-P model) of a neuron. Since then many artificial neural networks have developed from the well-known M-P model [2], [3].

An M-P neuron is an element with n inputs and one output. The general form of its function is

$$y = \operatorname{sgn}(wx - \varphi)$$

where

$x = (x_1, x_2, \cdots, x_n)^T$—an input vector
$w = (w_1, w_2, \cdots, w_n)$—a weight vector
φ—a threshold

$$\operatorname{sgn}(v) = \begin{cases} 1, & v > 0 \\ -1, & v \le 1. \end{cases} \tag{1}$$

Rumelhart *et al.* [12] presented the concept of feedforward neural networks and their corresponding learning algorithm, back propagation (BP), which provided the means for neural networks to be practicable. In essence, the BP is a gradient descent approach. In BP, the node function is replaced by a class of sigmoid functions, i.e., infinitely differentiable functions, in order to use mathematics with ease. Although BP is a widely used learning algorithm it is still limited by some disadvantages such as low convergence speed, poor performance of the network, etc.

In order to overcome the learning complexity of BP and other well-known algorithms, several improvements have been presented. Since the node function $f(wx-\varphi)$ of the M-P model can be regarded as a function of two functions: a linear function $wx-\varphi$ and a sign (or characteristic) function $f(\cdot)$, generally, there are two ways to reduce the learning complexity. One is to replace the linear function by a quadratic function or a distance function. For example, the polynomial-time-trained hyperspherical classifier presented in [7] and [8] and the restricted Coulomb energy algorithm presented in [6] and [14] are neural networks using some distance function as their node functions. A variety of classifiers such as those based on radial basis functions (RBF) [9]–[13] and fuzzy classifiers with hyperboxes [15] and ellipsoidal regions based on Gaussian basis function [16], etc., use the same idea, i.e., replacing the linear function by a quadratic function. Although the learning capacity of a neural network can be improved by making the node functions more complicated, the improvement of the learning complexity would be limited due to the complexity of functions. Another way to enhance the learning capacity is by changing the topological structure of the network. For example, in [17] and [18], the number of hidden layers and/or the number of connections between layers are increased. Similarly, the learning capacity is improved at the price of increasing the complexity of the network. A detailed description of the above methods can be found in [19].

Now the problem is whether we can reduce the learning complexity of a neural network and still maintain the simplicity of the M-P model and its corresponding network. Some researchers tried to do so by directly using the geometrical interpretation of the M-P model, however, they have been unsuccessful. Note that $wx-\varphi=0$ can be interpreted as a hyperplane P in an n-dimensional space. When $(wx-\varphi)>0$, input vector x falls into the positive half-space of the hyperplane P. Meanwhile, $y = \operatorname{sgn}(wx-\varphi) = 1$. When $(wx-\varphi)<0$, input vector x falls into the negative half-space of P, and $y=-1$. In summary, the function of an M-P neuron can geometrically be regarded as a spatial discriminator of an n-dimensional space divided by the hyperplane P. Rujan *et al.* [4], [5] intended to use such a geometrical interpretation to analyze the behavior of neural networks. Unfortunately, when the dimension n of the space and the number m of the hyperplanes P (i.e., the number of neurons) increase, the mutual intersection among these hyperplanes in n-dimensional space will become too complex to analyze. Therefore, so far the geometrical representation has still rarely been used to improve the learning capacity of complex neural networks.

In order to overcome this difficulty, a new representation is presented as follows. First, assume that each input vector x has an equal length (norm). Thus all input vectors will be restricted to an n-dimensional sphere S^n. (In general cases, the

Manuscript received March 11, 1999; revised January 30, 1998, December 15, 1998, and March 11, 1999. This work was supported by the National Nature Science Foundation of China and the National Key Basic Research Program of China.

L. Zhang is with The State Key Lab of Intelligent Technology and Systems, Artificial Intelligence Institute, Anhui University, Anhui, China.

B. Zhang is with the Department of Computer Science, Tsinghua University, Beijing, China.

Publisher Item Identifier S 1045-9227(99)05482-X.

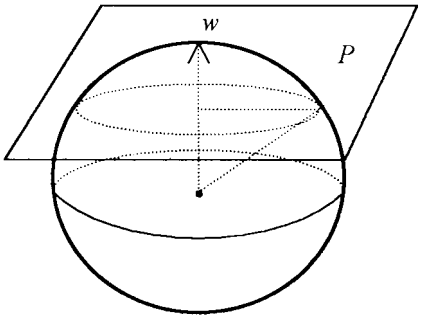


Fig. 1. A sphere neighborhood.

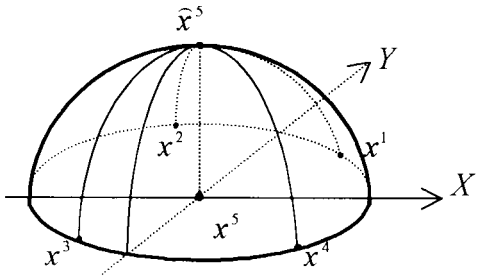


Fig. 2. Input vectors and their projections.

definition of S^n will be given later on.) Then, $(wx-\varphi)>0$ represents the positive half-space partitioned by the hyperplane P and the intersection between the positive half-space and S^n is called a "sphere neighborhood" as shown in Fig. 1. When input x falls into the region, output $y=1$, otherwise, $y=-1$.

If the weight vector w has the same length as input x, then w becomes the center of the sphere neighborhood, and $r(\varphi)$, a monotonically decreasing function of φ, becomes its radius. If the node function is a characteristic function $\sigma(v)$

$$\sigma(v)=\begin{cases}1, & v>0\\ 0, & \text{otherwise}\end{cases} \tag{2}$$

then the function $\sigma(wx-\varphi)$ of a neuron is a characteristic function of the sphere neighborhood on the sphere, i.e., an M-P neuron corresponds to a sphere neighborhood in an n-dimensional sphere. The preceding clear visual picture of an M-P neuron is a great help to the analysis of neural networks. In the preceding discussion, input vectors are assumed to have equal length. This is not the case in general. Although normalization could turn any input vector into a unit vector (a vector with a unit length), it will reduce the original n-dimensional space to an $(n-1)$-dimensional space. Some input information will be lost, but a way to circumvent the limitation is presented as follows.

Assume that the domain of input vectors is a bounded set D of an n-dimensional space. S^n is an n-dimensional sphere of an $(n+1)$-dimensional space.

Define a transformation $T: D\to S^n, x\in D$, such that

$$T(x)=(x,\sqrt{d^2-|x|^2})$$

where $d\geq \max\{|x||x\in D\}$.

Thus all points of D are projected upward on the S^n by transformation T (Fig. 2). Obviously, a neuron (w,φ) corresponds to a characteristic function of a "sphere neighborhood" on S^n with w as its center and $r(\varphi)$ as its radius.

Example 1: Five input vectors on a plane are given, i.e.,

$$\begin{aligned}D&=\{(1,1),(1,-1),(-1,1),(-1,-1),(0,0)\}\\&=\{x^1,x^2,x^3,x^4,x^5\},\end{aligned}$$

as shown in Fig. 2.
Construct a neural network to classify D into two classes

$$D^1=\{x^1,x^2,x^3,x^4\}\quad\text{and}\quad D^2=\{x^5\}.$$

Solution: First, D is projected on sphere S^2 so that each vector x^i in D is transformed into a three-dimensional vector y^i having the equal length $\sqrt{2}$. Then, we have

$$\begin{aligned}\tilde{D}=\{&y^1=(1,1,0),y^2=(1,-1,0),y^3=(-1,1,0),y^4\\&=(-1,-1,0),y^5=(0,0,\sqrt{2})\}.\end{aligned}$$

From the geometrical representation (Fig. 2), it is known that only one neuron (w,φ) is needed to partition $\tilde{D}$ into two classes, i.e., $\tilde{D}^1$ and $\tilde{D}^2$, where $w=(0,0,\sqrt{2})$ and $\varphi=\frac{\sqrt{2}}{2}$.

Note that in the original space D, at least two lines (neurons) are needed to partition D into D^1 and D^2. It is shown that the partition based on sphere neighborhoods is more powerful than the "half-space" partition. A hyperplane, i.e., the same linear form $(wx-\varphi)$, in the transformed curved (quadratic) space S^n is equivalent to a quadratic curved surface in the original flat space. Thus the partition by planes on a sphere space is equivalent to the partition by quadratic curved surfaces in the original flat space. For any given m points on S^n, m sphere neighborhoods can always be used to partition them such that each point falls into just one neighborhood. It is not always the case in the flat-space partition.

II. A Constructive Algorithm of Neural Networks

A framework of the proposed constructive algorithm of neural classifiers can be stated as follows. A set $K=\{l^t=(x^t,y^t),t=0,1,2,\cdots,s\}$ of training samples is given. Assume that output y^t in K has only k different values, i.e., the training samples will be classified into k classes. There is no loss of generality in assuming that the first k outputs have mutually different values. Let $I(t)$ be a set of indices of samples with the same output y^t, $t=0,1,\cdots,k-1$, i.e., $I(t)=\{i\mid y^i=y^t\}$, the input set corresponding to $I(t)$ be $p(t)=\{x^i\mid i\in I(t)\},t=0,1,\cdots,k-1$.

The design of a neural classifier can be divided into two stages: the input covering stage and the forward propagation algorithm stage.

A. Input Covering Stage

As a classifier (or an associative memory), the function of a neural network can be stated as follows. Given a set $K=\{l^t=(x^t,y^t),t=0,1,\cdots,s\}$ of training samples, after learning, the network should store the input pairs (x^t,y^t). Moreover, when the input becomes $x^t+\Delta^t$, the output y^t still remains the same, where Δ^t is regarded as a noise or an error.

For an M-P neuron (w, φ), when $w = x^t$ and $r(\varphi) = \Delta^t$, as mentioned before, it geometrically corresponds to a sphere neighborhood on S^n with x^t as its center and Δ^t as its radius. When an input falls into the neighborhood, the neuron always has outputs 1, otherwise, -1. An M-P neuron acts as a discriminator of an input class. Therefore, a network consisting of these neurons has the same function as a classifier.

The first design stage of a neural classifier can be transformed to an input vectors (a point set of an n-dimensional space) covering problem. Namely, a set of sphere neighborhoods is chosen to cover the inputs having the same output, i.e., belonging to the same class. Different classes of inputs are covered by different sets of sphere neighborhoods. Namely, an M-P neuron also corresponds to a covering on S^n.

It means that a set $\{C_j(t), j = 1, 2, \cdots, k_t\}$ of sphere neighborhoods (coverings) is chosen such that $C(t) = \bigcup c_j(t)$ covers every input $x^i \in p(t)$ and does not cover any input $x^i \notin p(t)$, and $C(t)'s, t = 0, 1, \cdots, k-1$ are mutually disjoint.

One possible covering approach is as follows.

Fix t, for any $x^i \in p(t)$
let

$$\left.\begin{aligned} d^1(i) &= \max_{j \notin I(t)} \{\langle x^i, x^j \rangle\} \\ d^2(i) &= \min_{j \in I(t)} \{\langle x^i, x^j \rangle > d^1(i)\} \\ d(i) &= \frac{d^1(i) + d^2(i)}{2} \end{aligned}\right\} \tag{3}$$

where $\langle x, y \rangle$ denotes the inner-product of x and y, and

$d^1(i)$ minimum "distance" between x^i and $x^j \notin p(t)$;
$d^2(i)$ maximum "distance" between x^i and x^j belonging to $p(t)$ and at a distance less than $d^1(i)$ from x^i.

Choose

$$w^i = x^i, \quad \varphi_i = d(i), \qquad i = 1, 2, \cdots, k_t. \tag{4}$$

Let

$$\begin{aligned} C_i(t) &= \{x \mid \langle x, x^i \rangle > \varphi_i\} \\ C(t) &= \bigcup_{i \in I(t)} C_i(t), \qquad t = 0, 1, \cdots, k-1. \end{aligned} \tag{5}$$

An M-P neuron (w^i, φ_i) described by (4) corresponds to a covering $C_i(t)$, a neighborhood on S^n with x^i as its center. The number of coverings $C_i(t)$ determines the number of neurons needed in the first layer of a neural classifier. The inputs are classified into different classes by the first layer (hidden layer). The second layer is used to form the given outputs y^t. From $C_i'(t)$, if a subcovering $C_i'(t)$ of $p(t)$ can be found, the amount of first-layer neurons will be reduced.

Therefore, the first design stage of a neural classifier is reduced to a minimum coverings problem. Since the minimum coverings problem is a classical one in many fields, e.g., Combinatorics [23], there are many well-known algorithms to deal with it. Neural network research can draw lessons from these algorithms. Therefore, a variety of covering approaches can be adopted. For example, it is not necessary to take the training samples as the centers of every coverings, etc. This can be seen later on.

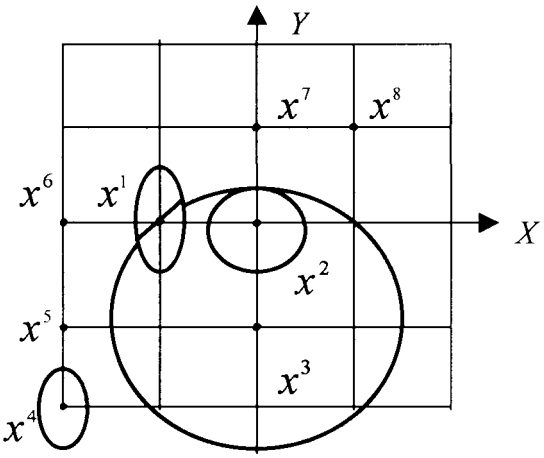


Fig. 3. Input vector and their coverings.

Example 2: Eight training samples (points) K are given in a plane (Fig. 3). Design a three-layer neural network to classify K into K_1 and K_2, where

$$\begin{aligned} K_1 = \{&x^1 = (-1, 0), x^2 = (0, 0), x^3 = (0, -1), \\ &x^4 = (-2, -2)\} \\ K_2 = \{&x^5 = (-2, -1), x^6 = (-2, 0), x^7 = (0, 1), \\ &x^8 = (1, 1)\}. \end{aligned}$$

Solution: Project the input vectors on the sphere, we have

$$\begin{aligned} K_1 = \{&y^1 = (-1, 0, \sqrt{7}), y^2 = (0, 0, \sqrt{8}), y^3 = (0, -1, \sqrt{7}), \\ &y^4 = (-2, -2, 0)\} \\ K_2' = \{&y^5 = (-2, -1, \sqrt{3}), y^6 = (-2, 0, 2), y^7 = (0, 1, \sqrt{7}), \\ &y^8 = (1, 1, \sqrt{6})\}. \end{aligned}$$

According to (3) and (5), we can find a set $C(1) = \{C_1(1), C_2(1), C_3(1), C_4(1)\}$ of coverings which cover every point in K_1' and do not cover any point in K_2', where $C_1(1), C_2(1), C_3(1)$, and $C_4(1)$ cover y^1, y^2, y^3, and y^4, respectively. The minimum coverings of K_1' are $C_3(1)$ and $C_4(1)$. The projection of the coverings on the XY plane is shown in Fig. 3. Two neurons are needed in the first layer, specifically,

$$\begin{aligned} w^1 = y^3 = (0, -1, \sqrt{7}), &\quad \varphi_1 = 6.5 \\ w^2 = y^4 = (-2, -2, 0), &\quad \varphi_2 = 7. \end{aligned}$$

The neuron in the second layer is an OR element. The network is shown in Fig. 4.

B. Forward Propagation Algorithm

Given a set of training samples

$$K = \{l^{(t)} = (x^t, y^t), t = 0, 1, \cdots, p-1\}$$

by a covering approach, the input vectors have been covered by a set of coverings based on their classification. Assume that a set C of $p-1$ coverings is obtained as follows:

$$C = \{C(i), i = 1, 2, \cdots, p-1\}.$$

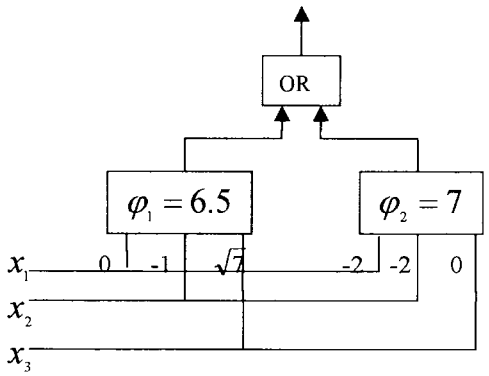


Fig. 4. A neural network example.

Assuming that x^i is the center of covering $C(i)$ and y^t is its corresponding output, $i = 1, 2, \cdots, p-1$. For simplicity, y^t's are assumed to be mutually different, i.e., one class corresponds to only one covering. (When one class corresponds to a set of coverings, the constructive approach is almost the same.) The domain of each component of x^i (y^t) is assumed to be $\{1, -1\}$. Then a three-layer neural network, i.e., two layer elements, can be constructed by the Forward Propagation (FP) algorithm presented in [20].

In contrast to the well-known BP algorithm, the FP algorithm starts from the first layer to the last one, so it is called forward propagation. When the components of input and output vectors are real-valued rather than $\{-1, 1\}$, by making small changes, the FP algorithm is still available.

III. The Comparison of Results

Since the minimum coverings problems are known to be NP-hard, the key point in the design of the proposed neural network is to find a feasible covering approach such that a satisfactory result can generally be obtained. A covering method called covering-deletion algorithm and its computer simulation results were presented in [20]. In order to show the performance of the proposed neural network, one of the problems, two-spiral discrimination, and its input covering result are shown in Fig. 5.

Example 3 (Two-Spiral Discrimination Problem): Two spirals K_1 and K_2 (in polar coordinates)

$$\begin{aligned} K_1&: \quad \rho = \theta \\ K_2&: \quad \rho = \theta + \pi \\ &\quad \frac{\pi}{2} \le \rho \le 6\pi. \end{aligned}$$

The projection of ten coverings on the XY plane is shown in Fig. 5. The correct classification rate for 20 000 training samples is 100%. The correct rate for 10 000 testing samples is 99.95%. Compared to the results in [17], [21], and [22], we can see that the classification of two spirals based on BP fails [21], and it takes 3000 iterations and only has 89.6% correct classification rate by using the generating-shrinking algorithm presented in [22]. Fahlman [17] uses a cascade network with more than ten layers to treat the same problem, and part of his results is presented in [18]. (It can be seen that the result presented in Fig. 5 is better than that presented in [18, Fig. 6.3.4]).

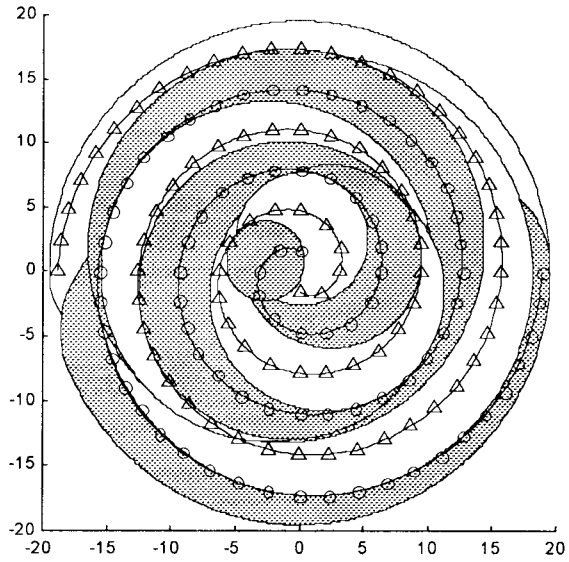


Fig. 5. Two-spirals problem and its input coverings.

IV. Mapping Abilities of Three-Layer Neural Networks

A well-known theorem about the mapping abilities of feedforward neural networks is shown as follows.

Theorem: Any function in $L^2(D^n)$ can be approximated to any desired degree of accuracy by a three-layer feedforward neural network, where D^n is a unit hypercube in an n-dimensional Euclidean space.

Many researchers have proved some similar theorems [8]–[10] from various approaches. A new visual proof of the theorem will be given below by using the geometrical interpretation of M-P neurons.

Definition 1: Assume that D is a bounded measurable set on S^n, where S^n is an n-dimensional sphere in an $(n+1)$-dimensional space. Function $f(x) : D \to R$ and $f(x) = \sum_{i=1}^{m} a_i d(D_i)$, where $d(D_i)$ is a characteristic function in D_i, and D_i is a measurable set. Function $f(x)$ is called a step function. If each D_i is a sphere neighborhood, then $f(x)$ is called a step function on sphere neighborhoods.

Definition 2: Assume that D is a bounded measurable set in R^n, and function $f(x) : D \to R$ is a bounded measurable function. A function $g(x) : D \to R$, if there exists (δ, ε) such that $\mu(\{x \mid |f(x) - g(x)| > \varepsilon\}) < \delta$ is called a (δ, ε)-measure approximation of $f(x)$, where μ is a measure in R^n.

In this paper, the "measurable" always means "Lebesque measurable."

Since D_i is a sphere neighborhood, each characteristic function $d(D_i)$ can be represented by a neuron A_i based on the above geometrical interpretation of the M-P neuron. If the first layer of a feedforward network is composed of these neurons A_i, $i = 1, 2, \cdots, m$, and the second layer is made by a linear element, then the network can realize the step function $f(x)$. From any textbook on Theory of Real Variable Functions [27],

it is known that any function on L^2(L^2-integrable function) can be approximated by a step function. Therefore, to prove the theorem, all we need to do is to confirm that any measurable set can be approximated by a set of sphere neighborhoods. This is also a well-known result in Theory of Real Variable Functions. So the theorem can be proven by slightly changing the well-known results. We have

Basic Theorem: D is a bounded measurable set on S^n and $f(x): D \rightarrow R$ is an almost bounded measurable function. For any δ, $\varepsilon > 0$, there exists a three-layer feedforward network such that its function $g(x)$ is a (δ, ε)-measure approximation of $f(x)$.

Corollary 1: D is a bounded measurable set in R^n, and $f(x): D \rightarrow R$ a L^2-integrable function. For any $\varepsilon > 0$, there exists a three-layer neural network such that $d_{L^2}(f(x), g(x)) < \varepsilon$, where $g(x)$ is the mapping function of the network.

This is the theorem presented in Hecht-Nielsen [26].

V. Conclusion

The key points of our work are the following. The original input space is transferred into a quadratic space, and the well-known (point set) covering method can be applied to perform partition of data in the transformed space. At the same time, the simplicity of the M-P model and its corresponding network still holds true. There is no need either to increase the complexity of node functions or the complexity of network's structure. A feasible covering-deletion design algorithm is presented as well, and computer simulation results show that the proposed neural network is quite efficient.

References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in neurons activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.

[2] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing*, vols. 1, 2. Cambridge, MA: MIT Press, 1986.

[3] R. J. Mammone and Y. Y. Zeevi, *Neural Networks: Theory and Applications*. San Diego, CA: Academic, 1991.

[4] P. Rujan and M. Marchand, "A geometric approach to learning in neural networks," in *Proc. IJCNN'89*, Washington, DC, vol. II, pp. 105–110.

[5] U. Ramacher and M. Wesseling, "A geometric approach to neural network design," in *Proc. IJCNN'89*, Washington, DC, vol. II, pp. 147–154.

[6] P. W. Cooper, "A note on adaptive hypersphere decision boundary," *IEEE Trans. Comput.*, pp. 948–949, 1996.

[7] A. Roy and S. Mukhopadhyay, "Pattern classification using linear programming," *ORSA J. Computer*, vol. 3, no. 1, pp. 66–80, 1991.

[8] S. Mukhopadhyay, A. Roy, L. S. Kim, and S. Govil, "A polynomial time algorithm for generating neural networks for pattern classification: Its stability properties and some test results," *Neural Comput.*, vol. 5, no. 2, pp. 317–330, 1993.

[9] T. Poggio and F. Girosi, "A theory of networks for approximation and learning," AI Memo 1140, MIT, Cambridge, MA, 1989.

[10] M. T. M. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, "On the training of radial basis function classifiers," *Neural Networks*, vol. 5, pp. 593–603, 1992.

[11] D. S. Broomhead and D. Lowe, "Multivariate functional interpolation and adaptive networks," *Complex Syst.*, vol. 2, pp. 321–355, 1988.

[12] S. Lee and R. Kil, "Multilayer feedforward potential function network," in *Proc. IEEE 2nd Int. Conf. Neural Networks*, San Diego. New York: IEEE, 1988, vol. I, pp. 161–171.

[13] J. Moody and D. Darken, "Fast learning in networks of locally-tuned processing unit," *Neural Comput.*, vol. I, no. 2, pp. 281–294.

[14] D. L. Reilly and L. N. Cooper, "An overview of neural networks: Early models to real world systems," in *An Introduction to Neural and Electronic Networks*, S. F. Zornetzer, J. L. Davis, and C. Lau, Eds. San Diego, CA: Academic, 1990, pp. 227–246.

[15] S. Abe and M.-S. Lan, "A method for fuzzy rules extraction directly from numerical data and its application to pattern classification," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 1, pp. 18–28, 1995.

[16] S. Abe and R. Thawonmas, "A fuzzy classifier with ellipsoidal regions," *IEEE Trans. Fuzzy Syst.*, vol. 5, no. 3, pp. 358–368, 1997.

[17] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems*, vol. 2, D. S. Tourdtzhy, Ed. San Mateo, CA: Kaufmann, 1990, pp. 524–532.

[18] K. J. Lang and M. L. Witbrock, "Learning to tell two spirals apart," in *Proc. 1988 Connectionists Models Summer Schools* Pittsburgh, PA, 1988, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. San Mateo, CA: Kaufmann, pp. 52–59.

[19] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. Cambridge, MA: MIT Press, 1995.

[20] L. Zhang and B. Zhang, "Neural network based classifiers for a vast amount of data," in *Proc. 3rd Pacific-Asia Conf. PAKDD-99, Methodologies for Knowledge Discovery and Data Mining*, Beijing, China, Apr. 26–28, 1999, pp. 238–246.

[21] E. B. Baum and K. J. Lang, "Constructing hidden units using examples and queries," in *Neural Information Processing Systems*, vol. 3, R. P. Lippman *et al.*, Eds. San Mateo, CA: Kaufmann, 1991, pp. 904–910.

[22] Q. C. Chen *et al.*, "Generating-shrinking algorithm for learning arbitrary classification," *Neural Networks*, vol. 7, pp. 1477–1489, 1994.

[23] F. S. Roberts, *Applied Combinatorics*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[24] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183–192, 1989.

[25] M. Arai, "Mapping abilities of three-layer networks," in *Proc. IJCNN'89*, Washington, DC, vol. 1, pp. 419–423.

[26] R. Hecht-Nielsen, "Theory of the back propagation neural networks," in *Proc. IJCNN'89*, Washington, DC, vol. 1, pp. 593–605.

[27] N. Bourbaki, *Fonctions d'une Variable*. Paris, France: Hermann, 1951.

策的概率粗糙集模型和传统粗糙集模型的比较分析[4]，表明三支决策的概率粗糙集模型的性能优于其他两个模型。近十多年来，三支决策粗糙集模型被深入研究，学者们进行了决策粗糙集的属性约简、聚类的研究等，并进一步将决策粗糙集引入不完备系统和多智能体系统中，应用于投资决策、信息过滤、垃圾邮件等[5-20]，取得很大成效。

在决策粗糙集模型中，正域、边界域和负域是由一对阈值(α,β)决定的，其中$0\leqslant\beta<\alpha\leqslant 1$。与 Pawlak 三个域相比，$\alpha$、$\beta$分别取代了概率极值 1 和 0，它们能够容忍一定程度的错误，概率在$[0,\beta)$范围的为负域，在$(\alpha,1]$范围的为正域，在(α,β)范围的为边界域。在这种情况下，正域的错误分类率小于等于$1-\alpha$，负域的错误分类率小于等于β，这为α和β给出了一种基于错误分类率的解释。这两个阈值参数由损失函数λ决定，但这个损失参数的大小却是由实验或者专家的意见给出，通过人类的经验获得，具有很大的主观性。因此，阈值α和β的解释与计算以及概率正域、负域、边界域的解释和应用成了目前需要解决的两大问题。

基于覆盖的构造性神经网络[21,22]，在学习过程中，依据学习数据集的物理分布，构造性地形成了对分类识别问题的正域、负域和不能确定对应分类结果的空白域以及被正域、负域重叠覆盖的不可确定分类特性的区域，这种特征与三支决策正域、负域、边界域的语义相似，而构造性地形成正域、负域、空白域(或重叠域)，可以不必人为决定关键参数，使得如何获得损失函数λ、阈值α和β的取值问题得以解决。另一方面，将三支决策方法引入构造性神经网络中，进一步完善和拓展了神经网络的传统二支决策形式。

本章首先给出构造性覆盖算法实现的球形几何意义、实现过程及特点；接着介绍构造性神经网络中神经元阈值的选取对边界域形成的影响，并给出对落入边界域样本的决策策略；然后，通过在 10 个数据集上的实验结果表明，本章提出的算法的有效性和该算法三支决策的合理性、可解释性；最后，给出总结。

6.2　构造性覆盖算法

本章讨论的算法基于 M－P 神经元的球形几何意义，由球形领域覆盖样本的思想实现，因此称为“覆盖算法”。又因为该算法从空隐层开始，即初始化时，隐层神经元个数为 0，在训练的过程中，隐层神经元的个数不断增加，直到所有的样本都被学习到。因为这种不断增加神经元的过程是构造性的，因此称本章的算法是构造性算法。结合这两个特点，将本章的算法称为“构造性覆盖算法”，由中国学者张铃、张钹提出[21,22]。

在介绍构造性覆盖算法之前，首先引入构造性覆盖算法实现的球形几何意义。

6.2.1　M－P 神经元的球形几何意义

1943 年，McCulloch 和 Pitts[23]根据神经元传递中的“0，1 律”和神经传递中信号不但有不同的强度，而且有兴奋和抑制两种情况，提出了神经元的数学模型(M－P 模型)，如图 6.1 所示。

一个 M-P 神经元是一个有 n 个输入和单个输出的处理单元，其输入输出关系为

$$y=\sigma(\langle w,x\rangle-\theta) \tag{6.1}$$

式中：$x=(x_1,x_2,\cdots,x_n)^{\mathrm{T}}$ 是 n 维输入向量；$w=(w_1,w_2,\cdots,w_n)^{\mathrm{T}}$ 是权向量；θ 是阈值；$y=\sigma(x')$ 是符号函数，即 $\sigma(x')=\begin{cases}1,x'>0,\\-1,x'\leqslant 0。\end{cases}$

如果把一个 M-P 神经元看成是由超平面划分的空间位置的分类识别器，那么分类的边界则为

$$\langle w,x\rangle-\theta=0 \tag{6.2}$$

式(6.2)代表的是 n 维空间中的一个超平面 P，其中 w 为超平面的法向量，若 w 为单位向量时，θ 的绝对值则表示坐标原点到该超平面的距离。当 $\langle w,x\rangle-\theta>0$ 时，表示点 X 落在超平面 P 的正半空间内，此时 $\sigma(\langle w,x\rangle-\theta)=1$；当 $\langle w,x\rangle-\theta<0$ 时，表示点 X 落在超平面 P 的负半空间内，此时 $\sigma(\langle w,x\rangle-\theta)=-1$，如图 6.2 所示。

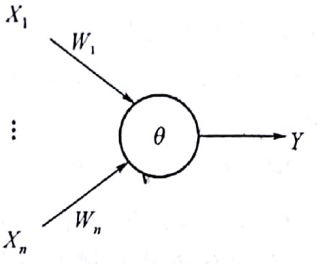


图 6.1　M-P 神经元模型图

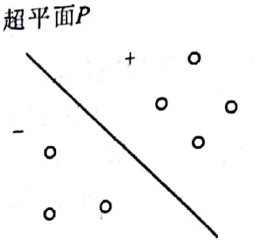


图 6.2　超平面 P：$\sum_{i=1}^{n} w_i x_i-\theta=0$ 的划分图

这给神经元一个相当直观的理解，人们也曾利用这个直观的理解进行神经网络学习的研究。然而，当两三个平面时，尚可直观理解，当 n、m 较大时，n 维空间中 m 个超平面的相交情况非常复杂，很不直观。总之，很难用超平面的模型帮助人们直观地理解神经网络的内在性质。故此后，很少有人再用神经元的几何意义(指用超平面的表示)进行神经网络的学习研究。

张铃、张钹在 1998 年对神经元的几何意义加以改变，将神经元的平面几何意义变为神经元的球形几何意义，为问题理解提供了新的直观帮助。

下面简述其基本思想。

若限定输入向量的长度相等，即输入向量限定在 n 维空间的某个球面上，此时 $(\langle w,x\rangle-\theta)>0$，表示球面上落在 P(P 为 $(\langle w,x\rangle-\theta)=0$ 描述的超平面)的正半空间的部分，这个部分恰好是球面上的某个“球形领域”，若取 w 与 x 等长，则这个“球形领域”的中心恰好是 w，其半径 $r(\theta)$ 是 θ 的单调下降的函数。

设 $\sigma'(x)=\begin{cases}1,x>0,\\0,\text{其他},\end{cases}$ 且取神经元的激励函数为 $\sigma'(\langle w,x\rangle-\theta)$，则一个神经元的激励函数正好是它所代表的球面上“球形领域”的特征函数(图 6.3)。

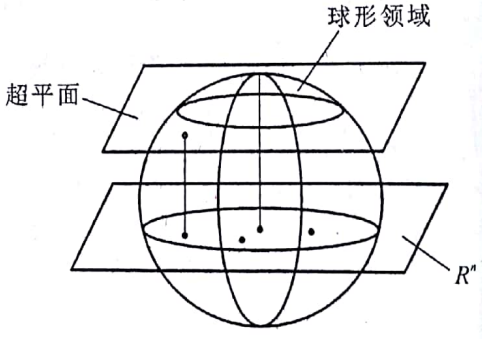


图 6.3　球形领域

利用神经元的这种球形几何意义，人们就能非常直观地进行神经网络的各种研究。由上面给出的神经元的球形几何意义得知，构造一个网络，使对给定的样本集能进行符合要求的分类，等价于求出一组领域，对给定样本集 X 中的点，能按分类的要求用球形领域覆盖将它们分隔开来。这样，就将神经网络的最优设计问题转化成某种求最优覆盖的问题。

6.2.2　构造性覆盖算法的网络结构

文献[21]中指出，假设样本集 $X=\{(x_1,y_1),(x_2,y_2),\cdots,(x_p,y_p)\}$，多层前向网络作为分类器和联想记忆器时，“记忆”功能是指当输入为 $x_i(i=1,2,\cdots,p)$ 时，网络的输出也相应地应该为 $y_i(i=1,2,\cdots,p)$，网络能够记住 x_i 与 y_i 的对应关系；“联想”功能则指当网络输入为 $x_i+\Delta_i(i=1,2,\cdots,p)$ 时，其中 Δ_i 是噪声，其输出仍然为 $y_i(i=1,2,\cdots,p)$。我们可以直观地理解为当输入样本落在 x_i 的附近时，其输出都应为 y_i。如果根据神经元的球形几何意义，进一步将 x_i 的附近看做是以 x_i 为中心、Δ_i 为半径的“球形领域”所对应的神经元，则多层前向神经网络作为分类器的设计相当于用若干个“球形领域”将输入 x_i 按照所属的类别区分开，换句话说，即对每一类，用一组球形领域将属于该类的 x_i 覆盖住，又不覆盖不属于该类的 x_j，于是不同类的输入被不同组的球形领域所覆盖，然后再将属于同组球形领域对应的神经元的输出用或门集中起来，即给出一个分类器的设计。因为每一个覆盖只与某类样本有关，因此与整个训练样本集之间并不是全连接关系，这样可以大大简化网络结构，如图 6.4 所示。

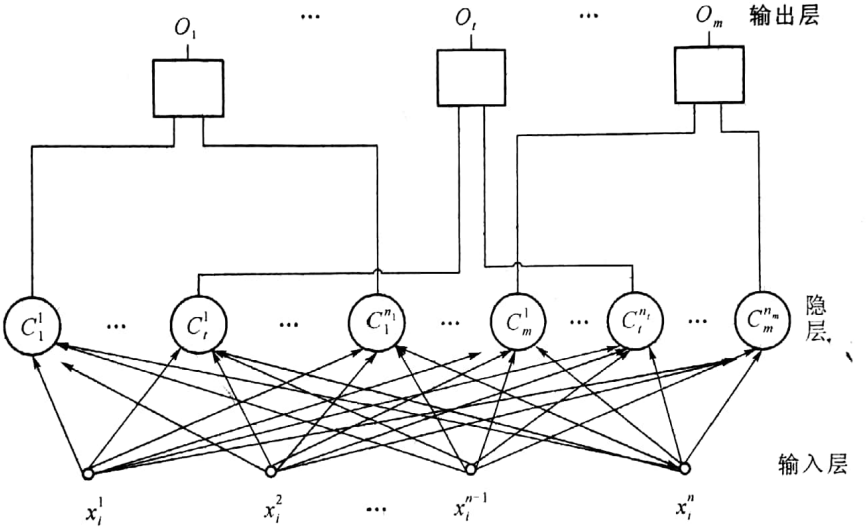


图 6.4　构造性覆盖算法的三层网络结构

问题：设给定一输入样本集 $\boldsymbol{X}=\{(x_1,y_1),(x_2,y_2),\cdots,(x_p,y_p)\}$，$\boldsymbol{X}$ 是 n 维欧氏空间的点集，共有 p 个样本，分 m 类，其中 $x_i=(x_i^1,x_i^2,\cdots,x_i^n)$ 表示第 i 个样本的 n 维特征属性，y_i 表示第 i 个样本的决策属性，即类别，$i=1,2,\cdots,p$。设计分类器，求作一个三层网络，使得属于第 j 类的样本输出为 $\boldsymbol{O}_j(o_1=0,o_2=0,\cdots,o_j=1,\cdots,o_m=0)$，即输出向量 $\boldsymbol{O}_j$ 的第 j 维分量为 1，其余分量为 0。

输入层：共 n 个神经元，每个神经元对应样本的一维，即样本的特征属性，$x_i=(x_i^1,x_i^2,\cdots,x_i^n)$，该层神经元只负责接收外部信息，自身无信息处理能力。

隐层：共 s 个神经元。初始时，隐层神经元为 0 个，每求得一个球形覆盖，增加一个神

经元，直到将所有的样本都被覆盖，从而求得一组覆盖：$C=\{C_1^1,C_1^2,\cdots,C_1^{n_1},C_2^1,C_2^2,\cdots,C_2^{n_2},\cdots,C_m^1,\cdots,C_m^{n_m}\}$，其中 C_i^j 表示第 i 类样本的第 j 个覆盖，是隐层中的一个神经元，隐层共有 $s=\sum n_i$ 个覆盖，第 i 类有 n_i 个覆盖，$i=1,2,\cdots,m$。神经元的权值是覆盖的中心，阈值为覆盖的半径。具体求法见 6.2.3 节。

输出层：共 m 个神经元，第 t 个神经元的输入为同类的一组覆盖，输出为该组覆盖的类别。$\boldsymbol{O}_t=(o_1=0,o_2=0,\cdots,o_t=1,\cdots,o_m=0)$，即 $o_t=1,o_j=0,t\neq j,t,j\in\{1,2,\cdots,m\}$，表示第 t 类样本的输出。该层神经元向外部输出处理信息。

6.2.3 构造性覆盖算法的实现

构造性覆盖算法属于有监督学习，分为训练和测试两个过程。

1. 训练过程

Step 1：归一化输入样本。

采用常见的归一化公式，将样本的特征值归一化至[0,1]，本文采用公式(6.3)。

$$x'=\frac{x-\text{MinValue}}{\text{MaxValue}-\text{MinValue}} \tag{6.3}$$

式中：x、x'为归一化前后的值；MaxValue、MinValue 分别为特征属性的最大值和最小值。

Step 2：将样本投影至 $n+1$ 维球面空间，使每个样本的长度相等。

大部分情况下所给定的输入样本向量的长度不相等，设输入的定义域样本为 n 维空间中的有界集合 X，取 $R\geqslant\max\{|x|,x\in X\}$，文献[21]通过公式(6.4)变换将样本点投射到球面 S^{n+1} 上并使得投射后的样本向量等长，如图 6.5 所示。

$$T:X\to S^{n+1},T(x)=(x,\sqrt{R^2-|x|^2}) \tag{6.4}$$

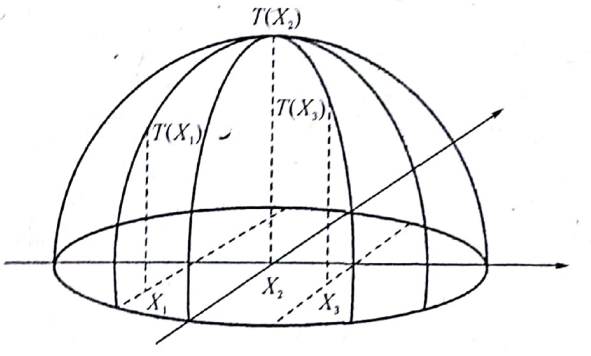


图 6.5 $T:D\to S^n$ 的样本点投射图

正是这种变换将无限的超平面转换为有限的超球形领域，使得分类器的设计成为可能。

Step 3：构造隐层神经元。

随机选取样本集中某一样本 x_k，作为覆盖中心，即隐层神经元的权值 w；根据同类样本及异类样本与覆盖中心的距离，求取覆盖半径，得隐层神经元的阈值 θ。每个神经元(w,θ)就是在超球面 S^{n+1} 上，以 w 为中心，以 θ 为半径的一个“球形领域”。将覆盖内的样本标记为“已学习”。

常用的获取覆盖半径(即神经元阈值)的方法有：

(1) 最小半径法

$$d_1(k) = \min_{y_i \neq y_k} \text{dist}(x_k, x_i) \qquad i \in \{1,2,\cdots,p\} \tag{6.5}$$

$$d_2(k) = \max_{y_i = y_k} \{\text{dist}(x_k, x_i) \mid \text{dist}(x_k, x_i) < d_1(k)\} \qquad i \in \{1,2,\cdots,p\} \tag{6.6}$$

$$\theta = d_2(k) \tag{6.7}$$

注：各样本经过归一化和投影后，样本向量等长，由欧式距离公式

$$\begin{aligned}\text{dist}(x_k, x_i) &= \sqrt{(x_k^1 - x_i^1)^2 + (x_k^2 - x_i^2)^2 + \cdots + (x_k^{n+1} - x_i^{n+1})^2} \\ &= \sqrt{(x_k^1)^2 + \cdots + (x_k^{n+1})^2 + (x_i^1)^2 + \cdots + (x_i^{n+1})^2 - 2(x_k^1 x_i^1 + \cdots + x_k^{n+1} x_i^{n+1})}\end{aligned}$$

和内积公式$\langle x_k, x_i \rangle = x_k^1 x_i^1 + \cdots + x_k^{n+1} x_i^{n+1}$知，求最大距离与求最小内积是等价的。同理，求最小距离与求最大内积是等价的。公式(6.5～6.7)可以改写为公式(6.8～6.10)。

$$d_1(k) = \max_{y_i \neq y_k} \{\langle x_k, x_i \rangle\} \qquad i \in \{1,2,\cdots,p\} \tag{6.8}$$

$$d_2(k) = \min_{y_i = y_k} \{\langle x_k, x_i \rangle \mid \langle x_k, x_i \rangle > d_1(k)\} \quad i \in \{1,2,\cdots,p\} \tag{6.9}$$

$$\theta = d_2(k) \tag{6.10}$$

最小半径法指以距离圆心最近的异类样本为界，以该界以内最远的同类点到圆心的距离为半径，即该神经元的阈值，如图 6.6 所示。

(2) 最大半径法

$$d_1(k) = \min_{y_i \neq y_k} \text{dist}(x_k, x_i) \qquad i \in \{1,2,\cdots,p\} \tag{6.11}$$

$$\theta = d_1(k) \tag{6.12}$$

最大半径法指以距离圆心最近的异类样本点到圆心的距离为半径，即该神经元的阈值，如图 6.7 所示。

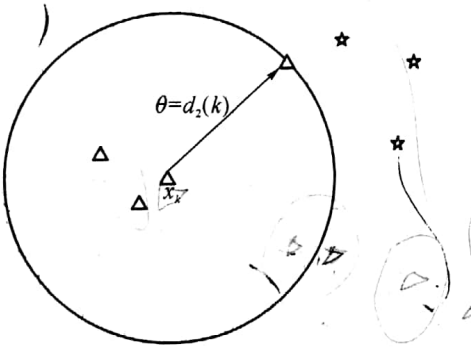


图 6.6　最小半径法示意图

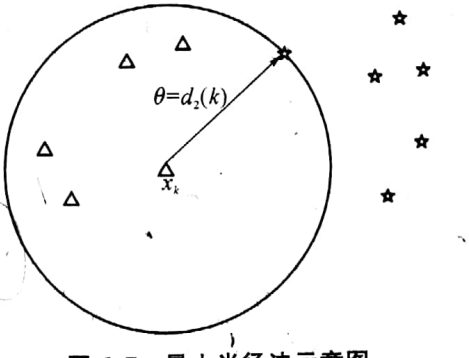


图 6.7　最大半径法示意图

(3) 折中半径法

$$d_1(k) = \min_{y_i \neq y_k} \text{dist}(x_k, x_i) \qquad i \in \{1,2,\cdots,p\} \tag{6.13}$$

$$d_2(k) = \max_{y_i = y_k} \{\text{dist}(x_k, x_i) \mid \text{dist}(x_k, x_i) < d_1(k)\} \qquad i \in \{1,2,\cdots,p\} \tag{6.14}$$

$$\theta=(d_1(k)+d_2(k))/2 \tag{6.15}$$

折中半径法指分别计算距圆心最近的异类样本距离，和该距离以内最远的同类点到圆心的距离，以两者距离的中间长度为半径，即该神经元的阈值，如图 6.8 所示。

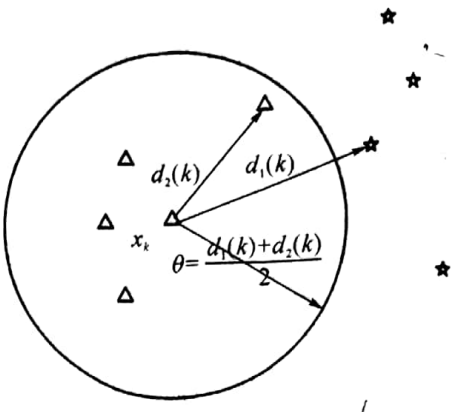


图 6.8 折中半径法示意图

Step 4：若所有的样本均被标记为“已学习”，结束。否则，转 Step 3。

算法最终将求出一组覆盖 $\boldsymbol{C}=\{C_1^1,C_1^2,\cdots,C_1^{n_1},C_2^1,C_2^2,\cdots,C_2^{n_2},\cdots,C_m^1,\cdots,C_m^{n_m}\}$，算法所得到的解。若令 $C_i=\bigcup C_i^j,j=1,2,\cdots,n_i$，则每个 C_i 表示第 i 类样本的所有覆盖。每个覆盖相当于一个神经元，根据这些覆盖可以构造一个神经网络，网络可以将所有的训练样本全部无误地分开。

2. 测试过程

根据训练阶段求出的一组覆盖 $\boldsymbol{C}=\{C_1^1,C_1^2,\cdots,C_1^{n_1},C_2^1,C_2^2,\cdots,C_2^{n_2},\cdots,C_m^1,\cdots,C_m^{n_m}\}$，对待测样本 x 进行测试，假设每个神经元 C_t^j 的权值为 w_t^j，阈值为 $r_t^j,t=1,2\cdots,m,j=1,\cdots,n_t$。

若采用最小半径和折中半径法，则神经元 C_t^j 输出为

$$o_t^j=\begin{cases}1,\text{若 } \mathrm{dist}(x,w_t^j)\leqslant r_t^j \text{ 且 } \mathrm{dist}(x,w_t^j)>r_t^j,\text{对所有的 } l\neq t,i=1,\cdots,n_t,l=1,\cdots,m\\0,\text{若 } \mathrm{dist}(x,w_t^j)>r_t^j \text{ 且 } \mathrm{dist}(x,w_t^j)\leqslant r_t^j,\text{仅对某个 } l\neq t,i=1,\cdots,n_t,l=1,\cdots,m\\\text{延迟决策,}\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{其他}\end{cases} \tag{6.16}$$

若采用最大半径法，则神经元 C_t^j 输出为

$$o_t^j=\begin{cases}1,\text{若 } \mathrm{dist}(x,w_t^j)<r_t^j \text{ 且 } \mathrm{dist}(x,w_t^j)\geqslant r_t^j,\text{对所有的 } l\neq t,i=1,\cdots,n_t,l=1,\cdots,m\\0,\text{若 } \mathrm{dist}(x,w_t^j)\geqslant r_t^j \text{ 且 } \mathrm{dist}(x,w_t^j)<r_t^j,\text{仅对某个 } l\neq t,i=1,\cdots,n_t,l=1,\cdots,m\\\text{延迟决策,}\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{其他}\end{cases} \tag{6.17}$$

式(6.17)表示：若待测样本 x 只落入第 t 类的某个覆盖，则将第 t 类的这个覆盖输出设 1；待测样本 x 只落入第 i 类的某个覆盖，则将第 t 类的覆盖输出设 0；若没有落到任何一个覆盖，或同时落入多个覆盖，则延迟决策。延迟决策策略将在第三节讨论，最终给出 o_t^j 的值。

因此，输出层第 t 个神经元的输入为 o_t^j，输出为 $o_t=\vee o_t^j,j=1,2\cdots,n_t$。

6.2.4 构造性覆盖算法的特点

根据本节对构造性覆盖模型及算法的讨论，将其主要特点总结如下：

(1) 快。在训练阶段，每个样本只学习一次。在学习过程中没有迭代，速度大大高于传统的基于搜索和梯度下降技术的神经网络方法。

(2) 构造性学习。构造性覆盖算法从 0 个隐层元开始，根据样本形成学习规则，不断增

加新的隐层元，直到样本学习完毕。该方法有效地避免了在学习初期盲目给定网络结构和参数的缺点。

(3) 可解释性强。一般神经元的权值和阈值的形成是黑匣子式的，形成原因和物理意义均不明确，构造性覆盖算法中隐层神经元的权值与阈值分别对应于(超)球形领域的圆心和半径，整个神经网络的结构和神经元的物理意义均是透明的，可解释性强。

(4) 网络结构简单。在神经网络的技术中，神经元原先的几何意义[23]是将它看做对无限空间线性划分的分类器，但是随着神经元个数的增加，问题的复杂性也随之以更大的速度增加。构造性覆盖算法利用球形映射将神经元变换成对有限空间划分的分类器，将无限空间转变成有限空间，大大降低了问题描述的复杂性，同时将神经网络长期未解决的学习问题转换成覆盖问题进行求解。

(5) 对学习过样本的 100%记忆。构造性覆盖算法根据训练样本将有限空间划分成若干个(超)球形领域，所有已经学习过的样本均落在某个(超)球形领域中，对这些样本，构造性覆盖算法有 100%的正确识别率，即如果测试样本是已经学习过的，该样本一定会被正确识别。这种能力在人类学习中称为“过目不忘”，在一般的神经网络方法中很难做到这一点。

(6) 同时进行多分类能力。构造性覆盖算法认为被某隐层神经元覆盖的样本属于同一类，在对空间进行划分、覆盖时，多分类问题可以一次性处理，对 m 类问题并不需要做 $m-1$ 次二分类处理，文献[24]中提出的构造性神经网络需要用多个二分类器去实现多分类问题。

(7) 动态学习。构造性覆盖算法的连接类似于大脑神经连接，某个神经元只对部分输入样本有反应，根据激励函数，判断样本与隐层神经元的权值、阈值的关系，确定激活某个隐层神经元，这种隐元与样本的非全连接特征比较适于动态学习和增量学习，当有新增样本要学习，不需要改变整个神经网络结构，只需调整与之相关的部分神经元。

(8) 三支决策。用球形领域切分有限空间，因为训练样本的有限性，并不能完全覆盖全部空间，同时某一空间可能会被多个球形领域覆盖，因此，存在一定的边界区域，对测试样本决策时，落入该区域的样本无法立刻做到判别为某类，需要进一步决策、处于延迟决策状态。这使得构造性覆盖算法和三支决策理论相互取长补短，三支决策理论拓展构造性覆盖算法的二支决策理论，构造性覆盖算法的构造过程自适应形成正域、负域和边界域，物理意义明确。

下一节我们将重点讨论隐层神经元的阈值对边界域形成的影响，并给出几种延迟决策策略。

6.3　构造性覆盖算法的三支决策

三支决策问题[25]：假设 U 是有限非空集合，C 是有限规则集。三支决策问题就是根据规则集 C 将 U 分成两两不相交的区域：正域(POS)、负域(NEG)和边界域(BND)。

6.3.1　三支决策域的形成

1. 阈值的解释与计算

由于球形领域的有界性和样本的有限性，构造性覆盖算法自身特点保证了在学习过程

Voting based Constructive Covering Algorithm

Zihan Ding
School of Computer Science and Technology, Anhui University
Hefei, China
dingzihan980720@gmail.com

Ruting Rao
School of Elactrical Engerneering and Automation, Anhui University
Hefei, China
2281532960@qq.com

Yuanting Yan
School of Computer Science and Technology, Anhui University
Hefei, China
ytyan_ahu@qq.com

Yanping Zhang
School of Computer Science and Technology, Anhui University
Hefei, China
zhangyp2@gmail.com

***Abstract*—As one of the typical neural networks from geometric perspective, constructive covering algorithm (CCA) has received extensive attention since it came out. It constructs the network through randomly initializing the cover center, which also creates the randomization of the algorithm. This paper proposes a method called voting based constructive covering algorithm (V-CCA). The main mechanism of V-CCA is introducing ensemble technique to overcome the above-mentioned shortcoming. V-CCA trains a group of classifiers each with random initialization instead of single classifier to generate a stronger learner. Numerical experiments on 16 UCI datasets demonstrate the effectiveness of V-CCA.**

Keywords—Constructive Covering Algorithm, classification, majority voting.

I. INTRODUCTION

Classification learning is one of the most important research brunches in machine learning area. As one of the traditional classification models, artificial neural network (NN) has been successfully applied to various practical applications. In the past few decades, many variants of neural networks have been proposed.

The Constructive Covering Algorithm (CCA) is one of the classical methods of neural networks with the characteristics of geometric representation [1]. CCA randomly select a sample as the cover center, then it calculates the cover radius and labels all the samples covered by the constructed cover as "learned". After all the samples have been learned. CCA trains the classification model by increasing the hidden layer neurons (each corresponds to a cover) iteratively with the cover construction process [2]. It has received extensive attention in many fields, such as pattern recognition, data mining, etc.[3]-[7].

However, CCA randomly initiating the covering center during the training process will result in differences in classification boundaries, that is, the classification result is not optimal and some samples may be misclassified. The classification accuracy of the dataset will be reduced, especially some samples near the classification boundaries.

In order to improve the accuracy of CCA and reduce the number of misclassified samples, we propose in this paper a Voting-based Constructive Covering Algorithm (V-CCA). The core idea of V-CCA is to train multiple independent classifiers instead of single one [8][9]. And then the test samples are determined by a voting strategy, which is a typical method for ensemble[10]-[12]. Compared with the traditional CCA, V-CCA can obviously optimize the classification boundary through a voting based ensemble strategy and improve the generalization ability.

The structure of this paper is as follows: In section 2, we review the basic concepts of traditional CCA and propose the problem of CCA in practical application. In section 3, we describe V-CCA in detail. The results of the comparative experiments are provided in Section 4.

II. CONSTRUCTIVE COVERING ALGORITHM

In this section, the basic concepts of CCA is reviewed in section 2.1. We analyze the problems that CCA may have encountered when applied to real-world applications in Section 2.2.

A. CCA Overview

As one of the extensions of neural network, CCA mainly contains three layers: the input layer, the hidden layer and the output layer. The input layer has *n* neurons, each neuron receives one dimension of a sample. The hidden layer has *s* neurons, each neuron is added for a corresponding spherical sphere (a cover) until all the samples are covered. The output layer has *m* neurons, the input of the neurons in output layer is a set of covers with same class, and the output is the corresponding class label [2].

CCA first projects the samples to a finite spherical space, and then constructs the cover of each category based on the samples on the hyper-sphere.

Before constructing covers on samples, the following two steps are adopted:

(1) Normalize the data to a range of 0-1 with a linear function.

$$x' = \frac{x - MinValue}{MaxValue - MinValue} \tag{1}$$

(2) Map X to (n+1)-dimensional sphere.

$$T: X \to S^{n+1}, T(x) = (x, \sqrt{R^2 - |x|^2}) \tag{2}$$

where $R \geq \max\{|x|, x \in X\}$.

After the normalization and mapping operation, covers can be constructed on the data. The process of cover constructing can be summarized as follows (where $< \cdot >$ indicates inner product):

Training (Constructing covers)

Step 1: Select an uncovered sample x_k as the center of a cover randomly.

Step 2: Compute the radius r of the cover centered on x_k :

Step 2.1: Compute minimum radius $d_1(k)$ of opposite classes.

$$d_1(k) = \min_{y_k \neq y_j} \{< x_k, x_j >\}, j \in \{1, 2, ..., m\} \tag{3}$$

Step 2.2: Compute maximum radius $d_2(k)$ of same classes.

$$d_2(k) = \min_{y_k = y_j}\{< x_k, x_j > | < x_k, x_j >> d_1(k)\},\ j \in \{1,2,...,m\} \quad (4)$$

Step 2.3: Compute the radius *r*.

$$r = (d_1(k) + d_2(k))/2 \quad (5)$$

Step 3: Construct a cover C_k centered on x_k with radius *r*.

Step 4: Samples in the cover are marked as "learned" and then removed, then go back to Step 1 until all samples are covered.

To this end, a set of covers $C = \{C_1, C_2, ..., C_p\}$ can be obtained, where $C_i = \{C_i^1, C_i^2, ..., C_i^{n_i}\}$ represents all covers of the i_{th} category samples and each cover has a corresponding cover radius.

Testing

Step 1: For a test sample *x*, calculate its distance from all covers.

$$d(x, C_i^j) = < w_i^j, x > - r_i^j \quad (6)$$

where *x* is the cover center and r_i^j is the radius.

Step 2: Determine the label of test sample *x* as the label of the cover.

$$Label = \arg\max_i (d(x, C_i^j)) \quad (7)$$

As introduced above, it is obvious that CCA has some weakness in the certainty of determining the boundaries, that is different covers' centers selected, different boundaries are constructed.

B. The Uncertainty of CCA

In CCA, the neurons of the hidden layer are iteratively generated. During the training process, CCA selects the covering center randomly, thus, different initialization may result in different classification boundaries. In other words, with different cover centers, the hidden node parameters and classification boundaries are also different. Samples located near the classification boundary may be misclassified, resulting in poor performance of the classifier.

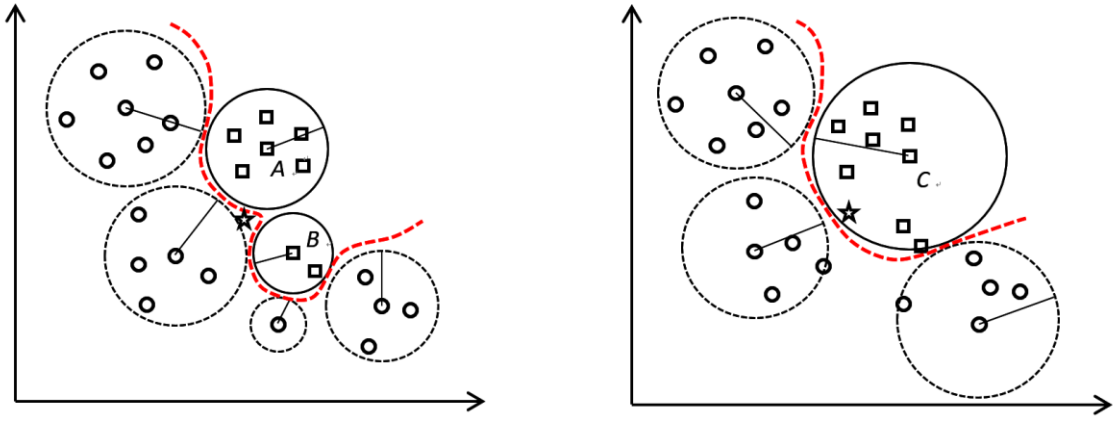
When there is single classifier (k=1), the sample near the boundary may be misclassified to another type, as shown in Fig. 1. The underlying reason for this phenomenon may be caused by the random initialization of selecting cover center. As is shown in Fig. 1, in Fig. 1(a), the cover center of the class with square shape is A and B, but in Fig. 1(b), the cover center of the class with circle shape is only C. Hence, the sample (star shape) is determined to the class with circle shape in subfigure (a), however it is determined to the class with square shape in subfigure (b).

To overcome the uncertainty as showed in Fig. 1, we introduce ensemble technique to further optimize the classification performance of CCA through combining multiple independent CCAs. Final prediction is determined through voting strategy.

III. VOTING BASED CONSTRUCTIVE COVERING ALGORITHM

As mentioned above, depending on the random selection of covers' center for each cover, the parameters of a hidden layer node of CCA are uncertain, which may result in non-optimal decision boundary. Consequently, we propose ensemble of several classifiers on the training set using various sets of random parameters where the parameters of each classifier are updated according to randomization criterion, and then making decision for testing samples through majority voting. To divided these datasets into training set and testing set, cross-validation is applied throughout the classification task. On the one hand, cross-validation scheme prohibits overfitting; on the other hand, it extends the number of predictors in the ensemble so as to guarantee a stable and accurate decision making.

In V-CCA, several individual CCAs generated from the training process can be obtained. All these individual CCAs are trained with the same dataset and the learning parameters of each CCA are randomly initialized independently. The final class label is then determined by majority voting on the results obtained by these independent CCAs. Suppose that K independent networks trained with the CCA are used in V-CCA. Then, for each testing sample x^{test}, K prediction results can be obtained based on these independent CCAs. A corresponding set $S_{K,x^{test}}$ with dimension equal to the number of class labels is used to store all these K results of x^{test}, where



(a) **(b)**

Fig. 1. Classifiers using CCA with different random initialization.

if the class label predicted by the k_{th} ($k \in [1,2,...,K]$) CCA is i, the value of the corresponding entry i in the set $S_{K,x^{test}}$ is increased by one, that is

$$S_{K,x^{test}}(i) = S_{K,x^{test}}(i) + 1 \tag{8}$$

After all these K results are assigned to $S_{K,x^{test}}$, the final class label of x^{test} is then determined by conducting a majority voting as follows:

$$c^{test} = \arg \max_{i \in \{1,...,n\}} \{S_{K,x^{test}}(i)\} \tag{9}$$

The algorithmic description of the proposed V-ELM is presented in algorithm 1.

IV. Performance Verification

In this section, we compare the performance of V-CCA with the original CCA. Simulations conducted on 16 UCI datasets are carried in the JetBrains Pycharm 2018.3.5, and running at x64 environment PC with 3.70GHZ CPU and 16GB RAM. Tenfold cross-validation is adopted for training.

Table 1 shows the specifications of datasets used in this paper, including the dataset name, number of attributes, classes and the size of the datasets. LM and ILPD are the abbreviations of datasets Library_Movement and Indian Liver Patient Dataset, respectively.

To study the relationship between K and the performance of V-CCA, experiments on 16 datasets are conducted. The independent training number K is gradually increased from 1

Algorithm 1 Voting based CCA

Input: Training set S, independent training number K.

Output: Final result of $x^{test}: c^{test}$.

Initialize: $k = 1$

While ($k \le K$) **do:**

$j \leftarrow 1$;

While $S \neq \varnothing$ **:**

Choose a sample $x_i \in S$ randomly;

Calculate radius d according to Eqs.(3) ~ (5);

Samples in the range are marked as "learned" and add learned samples to C_k^j;

Delete learned samples from S;

$j \leftarrow j + 1$;

End while

Add C_k^j to C_k

$k \leftarrow k + 1$;

End while

For every testing sample x^{test}

Set $k = 1$

while ($k \le K$)**do**

test x^{test} with CCA-k

$S_{K,x^{test}}(i) = S_{K,x^{test}}(i) + 1$

$k = k + 1$

End while

Obtaining the final result of $x^{test}: c^{test} = \arg \max_{i \in \{1,...,n\}} \{S_{K,x^{test}}(i)\}$

End for

Output: c^{test}

Fig. 2 Pseudo-code of the purposed V-CCA

to 30 one by one. For each K, 50 repetitions are conducted. To make the presentation of the experiment result clearer, we report the results of 4 datasets Breast-Cancer-Wisconsin (BCW), Car, Ionosphere, Balance-Scale (Balance). And the results of the rest 12 datasets are similar with the results reported.

Fig. 3 report the average results and gives the relationship between the independent training number K and the prediction accuracy. It can be seen from Fig. 3 that the accuracies for all the 4 datasets rise up monotonically with K. Moreover, the accuracy improvements at the beginning parts are more obvious than the accuracy improvements at the end parts. For instant, for dataset Car, when K increases from 2 to 10, the algorithm accuracy increases from 0.7845 to 0.8573, an ascension of 0.0928 (about 9.3%). But with the further increase of K from 11 to 30 the improvement of accuracy is only 0.007, less than 1%. As the training time will be increased with the increase of K with K =30 is about 3 times of the training time with K=10. We believe that it is reasonable to choose K in 10 to 18 for dataset Car. Similarly, we can see the phenomenon on the other 3 datasets. For convenience, we do not enumerate them here, and the details are given in Fig.3.

Table 2 illustrates the performance of CCA and V-CCA including accuracy, standard deviation (dev) and the difference between the two algorithms. And for V-CCA, 15 independent CCA are adopted for training and majority voting. As is shown in Table 2, V-CCA have better performance in all of the 16 datasets. The results can be roughly divided into three categories: significant improvement (more than 5%), not apparent improvement (less than 2%) and moderate improvement (from 2% to 5%). For example, on datasets Car, Lymphography, Segmentation, Balance, LM, the improvement is about 6.82%, 7.38%, 7.62%, 5.76%, 7.91% respectively. The improvement on datasets Breast-can, Glass, Horse-colic, ILPD and Iris are not apparent. The improvement on the rest 6 datasets are moderate.

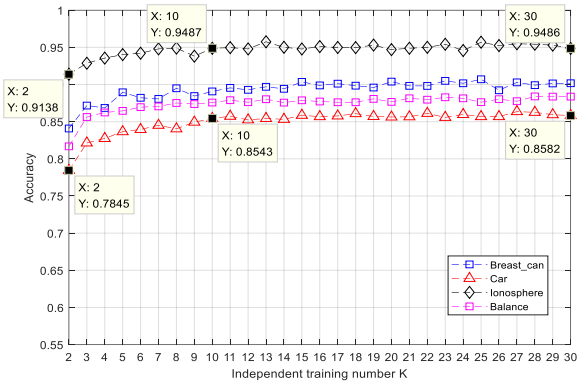


Fig. 3. Relationship between independent training number and prediction performance

Table 1 Specifications of classification datasets.

Dataset	Attributes	Classes	Size	Dataset	Attributes	Classes	Size
Haberman	3	2	306	Horse-colic	27	2	300
Breast-can	9	2	286	Lymphography	18	4	148
BCW	10	2	699	Segmentation	19	7	2310
Glass	9	7	214	ILPD	10	2	583
Ionosphere	34	2	351	Balance	4	3	625
CRX	15	2	690	LM	90	15	360
Car	6	4	1728	Kr-vs-kp	36	2	3196
Fertility	10	2	100	Iris	4	3	150

Table 2 performance comparisons with CCA

Dataset	CCA		V-CCA（k=15）		Rate Difference/%
	rate/%	dev/%	rate/%	dev/%	
Haberman	65.30	0.06	67.82	0.02	2.52
Breast-can	68.81	0.04	70.70	0.02	1.90
BCW	89.84	0.01	94.03	0.00	4.19
Glass	96.34	0.01	97.15	0.01	0.81
Ionosphere	90.77	0.02	95.10	0.00	4.33
CRX	65.44	0.02	68.48	0.01	3.04
Car	78.82	0.01	85.64	0.00	6.82
Fertility	80.48	0.11	84.40	0.02	3.92
Horse-colic	65.16	0.03	66.75	0.03	1.59
Lymphography	69.30	0.11	76.68	0.05	7.38
Segmentation	81.86	0.06	89.48	0.02	7.62
ILPD	69.24	0.03	71.02	0.01	1.78
Balance	82.13	0.02	87.89	0.00	5.76
LM	73.67	0.03	81.57	0.02	7.91
Kr-vs-kp	81.03	0.00	86.70	0.00	5.67
Iris	91.52	0.03	93.48	0.01	1.96

Table 3 Comparison of algorithm running time (s)

Dataset	CCA	V-CCA（k=15）	Dataset	CCA	V-CCA（k=15）
Haberman	0.41	8.82	Horse-colic	2.19	32.78
Breast_can	1.09	19.04	Lymphography	0.38	6.15
BCW	2.72	46.79	Segmentation	0.45	6.58
Glass	0.10	0.57	ILPD	4.62	76.23
Ionosphere	1.35	18.26	Balance	1.95	34.49
Crx	10.20	135.42	LM	6.29	75.90
Car	24.94	251.88	Kr-vs-kp	103.97	1027.17
Fertility	0.11	2.04	Iris	0.03	0.49

It is obvious in Table 2 that the standard deviations of V-CCA are much lower than CCA for all these 16 datasets. For example, standard deviation of CCA on dataset Segmentation (0.06%) is 3 times the one of V-CCA (0.02%). It indicates that V-CCA is much stable than CCA by training multiple classifiers instead of single one.

Table 3 shows the comparisons of the time consuming of CCA and V-CCA, including the training time and the testing time. It can be found in this table that the computational time of V-CCA is about 15 times than CCA on average.

V. Conclusion

CCA has received an extensive concern since it came out as a classification form of neural network. In this paper, we have proposed Voting based constructive covering algorithm to overcome the uncertainty caused by the constructive covering algorithm randomly selecting the initial covering center point. V-CCA trains multiple classifiers instead of single one to and combines them to construct a stronger classifier. Algorithm performance was verified on 16 UCI datasets. The experimental results show that the independent training number between 10 and 18 is a relatively reasonable choice, and the performance of ensemble classification are significantly better than single classifier.

References

[1] Zhang, L., & Zhang, B. (1999). A geometrical representation of McCulloch-Pitts neural model and its applications. IEEE Transactions on Neural Networks, 10(4), 925-929.

[2] Guliyev, N. J., & Ismailov, V. E. (2016). A single hidden layer feedforward network with only one neuron in the hidden layer can approximate any univariate function. Neural computation, 28(7), 1289-1304.

[3] Binkhonain, M., & Zhao, L. (2019). A Review of Machine Learning Algorithms for Identification and Classification of Non-Functional Requirements. Expert Systems with Applications.

[4] Yan, Y., Liu, R., Ding, Z., Du, X., Chen, J., & Zhang, Y. (2019). A Parameter-Free Cleaning Method for SMOTE in Imbalanced Classification. IEEE Access, 7, 23537-23548.

[5] Wang, D. (2008). Fast constructive-covering algorithm for neural networks and its implement in classification. Applied Soft Computing, 8(1), 166-173.

[6] Zhang, Y., Xing, H., Zou, H., Zhao, S., & Wang, X. (2013, October). A three-way decisions model based on constructive covering algorithm. In International Conference on Rough Sets and Knowledge Technology (pp. 346-353). Springer, Berlin, Heidelberg.

[7] Zhang, Y., Zhang, H., Wei, H., Tang, J., & Zhao, S. (2014). Multiple-instance learning with instance selection via constructive covering algorithm. Tsinghua science and technology, 19(3), 285-292.

[8] Liu, N., & Wang, H. (2010). Ensemble based extreme learning machine. IEEE Signal Processing Letters, 17(8), 754-757.

[9] Cannings, T. I., & Samworth, R. J. (2017). Random-projection ensemble classification. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79(4), 959-1035.

[10] Qingdi, W. (2015, August). Ensembling Base Classifiers to Improve Predictive Accuracy. In 2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES) (pp. 268-271). IEEE.

[11] Wei, H., Lin, X., Xu, X., Li, L., Zhang, W., & Wang, X. (2014, August). A novel ensemble classifier based on multiple diverse classification methods. In 2014 11th international conference on fuzzy systems and knowledge discovery (FSKD) (pp. 301-305). IEEE.

[12] Cao, J., Lin, Z., Huang, G. B., & Liu, N. (2012). Voting based extreme learning machine. Information Sciences, 185(1), 66-77.