Novel Evolutionary and Neural Computation for Classification

by

Xiaoxiao Qian

A dissertation

submitted to the Faculty of Engineering

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Engineering

University of Toyama

Gofuku 3190, Toyama-shi, Toyama 930-8555 Japan

2020

(Submitted March 7, 2020)

Abstract

With the development of society, people have encountered more and more complex problems in different fields of technology, commerce and finance. These problems have contributed to the rapid rise of computer applications. As one of the fastest growing science and technology in the world, computer technology has gradually evolved from the original mainframe to the current mega, miniaturization, intelligence and networking. The continuous development of computer technology is the result of the joint efforts of researchers from all over the world. The development of technology is ultimately to provide more convenience for people's life, study and work. The problems in practical applications have become more and more complicated, and the traditional calculation methods have been unable to meet the requirements, and at the same time face problems such as computational complexity and computational cost. In order to balance calculation accuracy and cost, more and more heuristic computational intelligence algorithms have been proposed. These algorithms are constructed by mimicking the evolutionary process of the organism, the way of thinking, the language, or the memory process. It is hoped that the wisdom of nature can be used to find a feasible solution in an acceptable time. The natural heuristic algorithm is just an imitation, and does not pursue complete agreement with the imitated person, or even guarantee reasonableness. Successful algorithms in such applications include ant colony algorithms, neural networks, and the like. In fact, computer intelligence has many branches, the main purpose is to meet the needs of different application areas.

This paper introduces several computational intelligence methods and gives a brief overview of their characteristics. At the same time, the differences and commonalities of various computational intelligence algorithms are discussed. In addition, the application of the computational intelligence method to the classification problem is given. The feasibility and practicability of the research are discussed. The results of the simulation experiment are provided and compared with other popular algorithms.

Firstly, the research process, analysis and results of ant colony optimization algorithm for high accuracy and low computational cost requirements are given. By constructing a list of IF-THEN rules, the traditional ant colony optimization (ACO) has been successfully applied on data classification with not only a promising accuracy but also a user comprehensibility. However, as the collected data to be classified usually contain large volumes and redundant features, it is challenging to further improve the classification accuracy and meanwhile reduce the computational time for ACO. This paper proposes a novel hybrid mutual information based ant colony algorithm (mr^{2}AM+) for classification. First, a maximum relevance minimum redundancy feature selection method is used to select the most informative and discriminative attributes in a dataset. Then, we use the enhanced ACO classifier (i.e., AM+) to perform the classification. Experimental results show that the proposed mr^{2}AM+ outperforms other seven state-of-art related classification algorithms in terms of accuracy and the size of model.

Secondly, the research process, analysis and results of the dendritic neuron model with plastic mechanism are given. By employing a neuron plasticity mechanism, the original dendritic neuron model (DNM) has been succeeded in the classification tasks with not only an encouraging accuracy but also a simple learning rule. However, the data collected in real-world contain a lot of redundancy, which causes the process of analyzing data by DNM becomes complicated and time-consuming. This paper proposes a reliable hybrid model which combines a maximum relevance minimum redundancy (Mr^2) feature selection technique with DNM (namely, Mr^{2}DNM) for classifying the practical classification problems. The mutual information-based Mr^2 is applied to evaluate and rank the most informative and discriminative features for the given dataset. The obtained optimal feature subset is used to train and test the DNM for classifying five different problems arisen from medical, physical and social scenarios. Experimental results suggest that the proposed Mr^{2}DNM outperforms DNM and other six classification algorithms in terms of accuracy and computational efficiency.

Both research results are optimized for practical application problems. First, the input

data is analyzed and processed, and the most streamlined and important data is transmitted to the algorithm or model, so that it can exert potential computing power. These studies have led us to believe that computing intelligence has many unexplored potentials, waiting for researchers to discover.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 An Outline of Computational Intelligence

In a wide range of different technologies, businesses, and finances, complex issues have spurred a need for computer applications that can exhibit ‘intelligent behavior”. These applications are expected to provide decision making, control processes, and identify and interpret patterns, or manipulate machines or robots in an unknown environment. Novel methods, tools, and programming environments have been developed to accomplish these tasks. The mechanisms and processes involved in these intelligent behaviors are already in the field of artificial intelligence research. Like other computer sciences, computational intelligence includes both theory and application.

As technology advances, the problems encountered in engineering practice become more and more complex. Using traditional calculation methods to solve these problems faces problems such as high computational complexity and long calculation time, especially for some non-deterministic polynomial hard (NP-hard) problems. Traditional algorithms cannot be solved within the tolerable time range. For example, like the classic Travel Saleman Problem (TSP) problem [1, 2], the branch and bound method can only solve up to 20 nodes. For a given problem, the strategy commonly employed in computational intelligence is to apply approximation techniques and methods to find rough, incomplete, or partially effective solutions. Therefore, in order to strike a balance between solution time and solution accuracy, computer scientists have proposed many heuristic computational intelligence algorithms, that is, simple sub-functions lead to complex, self-organizing behavior through interaction. These algorithms either mimic the evolutionary processes of the biological world, or mimic the physiological structures and bodily functions of living things, or imitate the behavior of animals, or mimic the characteristics of human thought, language, and memory processes, or mimic the physical phenomena of nature. It is hoped that the optimal solution of the problem can be realized by simulating the wisdom of nature and human beings, and an acceptable solution can be solved in an acceptable time. Naturally inspired methods are usually just imitations, not necessarily exacting, or even biologically justified. These successful applications include neural networks [3] and evolutionary algorithms, ant colony algorithms. In addition, the overall problem of uncertainty, fuzzy and uncertain knowledge problem solving strategies, this part of the guiding thinking is very valuable for human imperfect knowledge. Therefore, it is desirable to have a computer work with this knowledge without degrading accuracy and knowledge. Partially successful methods capable of dealing with fuzzy and uncertain knowledge are fuzzy systems and Bayesian networks. These algorithms together form a computational intelligence optimization algorithm.

1.2 The Definition of Computational Intelligence

The intersection, mutual penetration and mutual promotion of information science and life science is a distinctive feature of the development of modern science and technology. Computational intelligence involves areas such as neural networks [3], fuzzy logic [4], evolutionary computation [5], and artificial life. Its research and development reflect the important development trend of multidisciplinary integration and integration of contemporary science and technology. It may not be appropriate to classify a neural network as artificial intelligence, and classifying it into computational intelligence is more telling the essence of the problem. Some of the topics of evolutionary computation [5], artificial life, and fuzzy logic systems [4] are also classified in computational intelligence. The definition of computational intelligence was proposed by Bezdek in 1992 [6]. He believes that computational intelligence depends on the numerical data provided by the manufacturer and does not depend on knowledge. On the other hand, artificial intelligence applies knowledge tidbits. Although the boundaries between computational intelligence and artificial intelligence are not very obvious, it is useful to discuss their differences and relationships. Marks mentioned the difference between computational intelligence and artificial intelligence in 1993 [7], while Bezdek is concerned the relationship between pattern recognition (PR) and biological neural networks (BNN) or artificial neural networks (ANN), or computational neural networks (CNN), and the relationship between pattern recognition and other intelligence [8, 9]. Neglecting the difference between ANN and CNN may lead to confusion, misunderstanding, misrepresentation and misuse of the neural network model in pattern recognition.

Bezdek gave certain symbols and brief descriptions or definitions of these related terms (e.g. ABC) in order to describe computational intelligence and artificial intelligence:

- A-Artificial, which means artificial (non-biological), that is, artificial.
- B-Biological, representing physical + chemical + (??) = biological.
- C-Computational, representing math + computer.

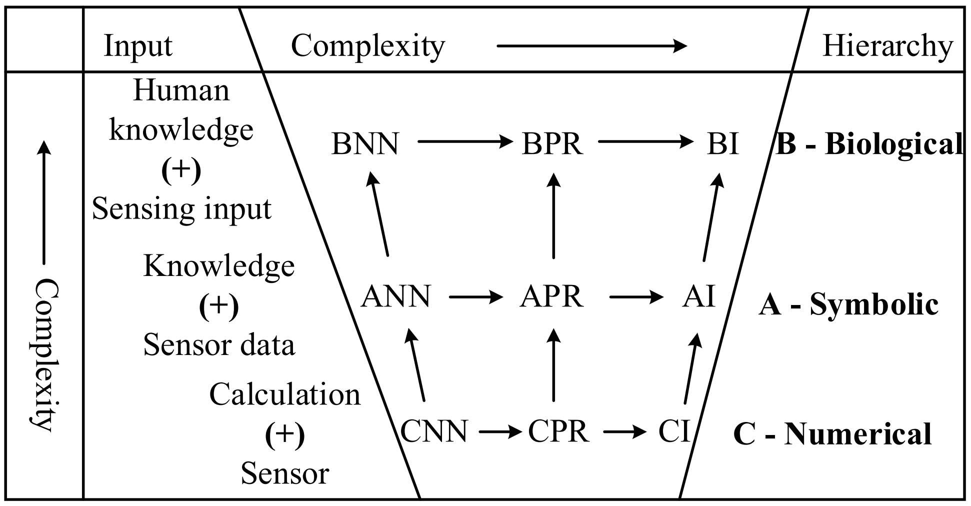


Figure 1.1: The interaction diagram of ABC.

Fig. 1.1 and Table 1.2 represent ABC and its relationship with neural networks (NN), pattern recognition (PR) and intelligence (I).

Table 1.1: The difference between Artificial Intelligence and Computational Intelligence.

	Classical Artificial Intelligence	Computational Intelligence
Object	Knowledge	Data
Method	Rigorous	Probability

- Horizontal: NN->PR->I (Neural Network-> Pattern Recognition->Intelligence)
- Portrait: C->A->B (value->symbol->biological)

Computational intelligence is a low-level cognition of intellectual means, which the difference between it and artificial intelligence is that the cognitive level falls from the middle to the lower level. The middle system contains knowledge (tidbits) and the low level system does not. Table 1.1 repsents the difference between Artificial Intelligence and Computational Intelligence.

When a system involves only numerical (lower) data, it contains a pattern recognition part, does not apply knowledge in the sense of artificial intelligence, and can present:

- Computational adaptability
- Computational fault tolerance
- Speed close to people
- Error rate is close to people

Then the system is a computing intelligence system.

When an intelligent computing system adds knowledge (tidbits) values in a non-numerical manner, it becomes an artificial intelligence system.

Since the advent of computers, artificial intelligence has been one of the goals pursued by computer scientists. As an important field of artificial intelligence, computational intelligence has good adaptability and strong global search ability because of its intelligence (Including the adaptive and self-organizing of the algorithm, the algorithm does not depend on the characteristics of the problem itself, and has universality.), parallelism (The algorithm basically solves the problem in a group collaboration way, which is very suitable for large-scale parallel processing.) and robustness (The algorithm has good fault tolerance and is not sensitive to initial conditions. It can find the optimal solution under different conditions.). It has good adaptability and strong global search capabilities and received extensive attention from many researchers. At present, many breakthroughs have been made in algorithm theory and algorithm performance, and it has been widely used in various fields, playing an important role in scientific research and production practice.

- Computational intelligence is motivated by the idea of bionics, which simulates natural intelligence based on the structure, evolution and cognition of the biological nervous system.

- Computational intelligence is a natural intelligent simulation method based on models (calculation models, mathematical models) and characterized by distributed and parallel computing.

- Although there is a coincidence between artificial intelligence and computational intelligence, computational intelligence is a new subject area. Whether it is bio-intelligence or machine intelligence, computational intelligence is its core part, while artificial intelligence is the outer layer.

From the perspective of computing intelligent systems, if a system only processes low-level numerical data, contains pattern recognition components, does not use knowledge in the sense of artificial intelligence, and has four characteristics: computational adaptability, computational tolerance, near-human calculation speed, and human error rate, then it is computational intelligence. From the perspective of subject, computational intelligence is a unified subject concept based on the relatively mature development of neural networks (NN), Evolutionary Computation (EC) and Fuzzy System (FS) [10].

Table 1.2: The Definition of ABC and its related fields.

BNN	Human Intelligence Hardware: Brain	Human sensor input processing
ANN	Middle model: CNN + knowledge tidbits	Middle-level treatment in the brain
CNN	Low-level, bio-incentive model	Brain-based sensor data processing
BPR	Search for human sensor data structures	Identification of structures in a human perception environment
APR	Middle model: CPR+ knowledge tidbits	Intermediate value and syntax processing
CPR	Search for sensor data structures	All CNN+ fuzzy, statistical and deterministic models
BI	Human Intelligence Software: Intelligence	Human cognition, memory and function
AI	Middle model: CI+ knowledge tidbits	Middle-level cognition
CI	Low-level algorithm for computational reasoning	Low-level cognition

Chapter 2

Computational Intelligence Algorithms

Computational Intelligence is based on the revelation of the laws of nature (biological world), according to its laws, to design an algorithm to solve the problem. The phenomena and laws of the disciplines of physics, chemistry, mathematics, biology, psychology, physiology, neuroscience and computer science may become the basis and source of ideas for computational intelligence algorithms.

Computational intelligence algorithms mainly include three parts: neural computing, fuzzy computing and evolutionary computing. As shown in Fig. 2.1, typical computational intelligence algorithms include artificial neural network algorithms in neural computing, fuzzy logic [4] in fuzzy computing, genetic algorithm in evolutionary computation, ant colony optimization algorithm [11–14] , particle swarm optimization algorithm [15, 16], immune algorithm, distribution estimation algorithm, Memetic algorithm, and single-point search technology such as simulated annealing algorithm, Tabu search algorithm, etc.

All of the above computational intelligence algorithms have a common feature of simulating human intelligence by mimicking one (some) aspect of human intelligence, realizing the computer programming of biological intelligence and natural laws, and designing the optimization algorithm. However, these different research fields of computational intelligence have their own characteristics. Although they have the commonality of imitating human and other biological intelligence, there are some differences in specific methods. For example: artificial neural network imitates the physiological structure of human brain and the process of information processing, simulating human wisdom; fuzzy logic (fuzzy system) [4] imitates the concept of ambiguity in human language and thinking, simulating

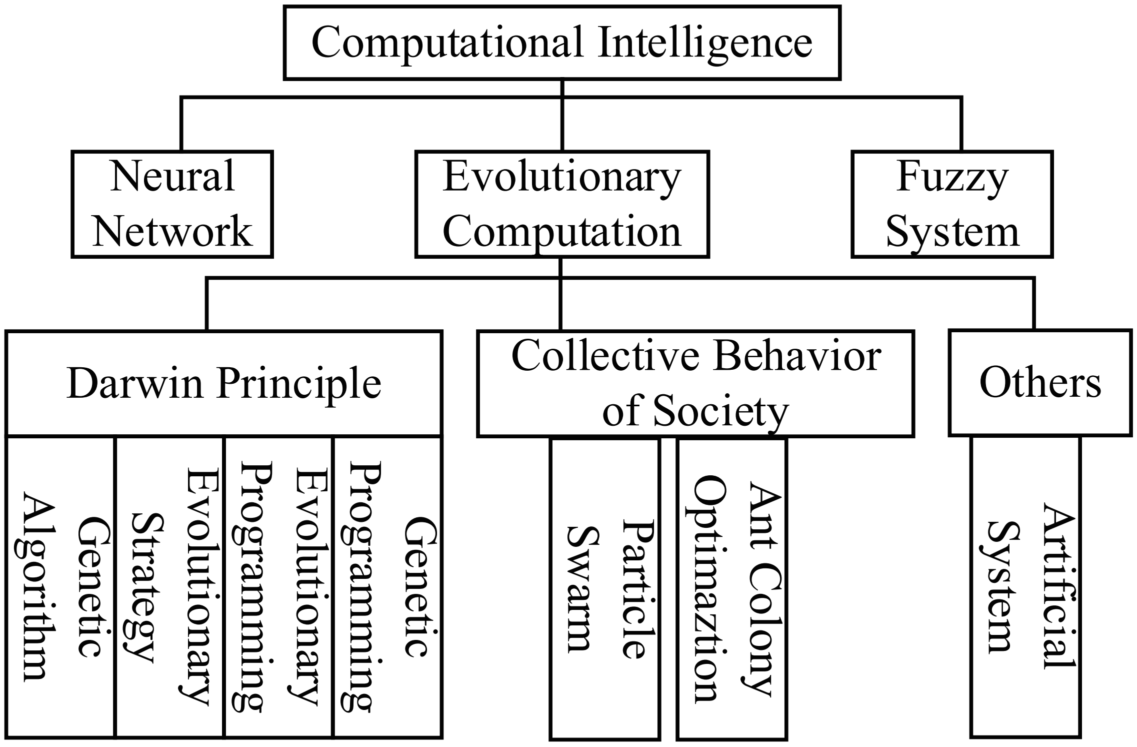


Figure 2.1: Computational intelligent algorithm.

human wisdom; evolutionary computation mimics the process of biological evolution and the process of swarm intelligence, simulating the wisdom of nature [17–19].

However, at this stage, the development of computational intelligence is also facing severe challenges. One of the important reasons is that computing intelligence still lacks a solid mathematical foundation. It is not possible to use mathematical tools to solve their computational problems as freely as physics, chemistry, astronomy, etc. Although neural networks have a relatively complete theoretical foundation, important computational intelligence techniques such as evolutionary computation have no perfect mathematical foundation. The analysis and proof of the stability and convergence of computational intelligent algorithms are still in the research stage. It is an important method to study the computational intelligence algorithm by numerical experiments and specific application methods to verify the effectiveness and efficiency of computational intelligent algorithms.

2.1 Artificial Neural Network

2.1.1 Progress in artificial neural network research

The artificial neural network system is a complex network system formed by a large number of simple processing units, that is, neurons are widely connected. In artificial neural networks, computation is done by the flow of data in the network. During the flow of data, each neuron receives an input data stream from its connected neurons, processes it, and transmits the result as an output data stream to other neurons connected to it. The topology of the network and the connection weight W_i between the neurons are determined by the corresponding learning algorithm. The algorithm continually adjusts the structure of the network and the connection weights between neurons until the neural network produces the required output. Through this learning process, artificial neural networks can continuously acquire knowledge from the environment and store this knowledge in the network in the form of network structure and connection weights.

Artificial neural networks have good self-learning, self-adaptation and self-organization capabilities, as well as human-scale parallel, distributed information storage and processing. This makes it ideal for dealing with incomplete, inaccurate information processing problems that require multiple factors to be considered simultaneously. At present, artificial neural networks have been highly valued by the academic community and have been widely used in many fields. However, it should be noted that in the design process of the neural network, the setting of various parameters and the determination of the network structure are highly empirical, and there is no complete theory to follow. Its scale is far from the scale of the billions of neurons that the human brain has. Moreover, the artificial neural network is based on the brain model, and its research is limited by the results of brain science research. It is difficult to truly simulate the human brain without a clear understanding of the thinking and cognitive processes of the human brain.

In 1960, Widlow and Hoff took the lead in using neural networks for automatic control research. From the late 1960s to the mid-1980s, neural network control was at the same low level as the entire neural network study. Since the late 1980s, with the recovery and development of artificial neural network research, the research on neural network control is also very active. The research progress in this area is mainly in neural network adaptive control and fuzzy neural network control and its application in robot control. Characteristics of artificial neural networks:

- Parallel distributed processing: The neural network has a high degree of parallel structure and parallel implementation capability, so it can have better fault tolerance and faster overall processing capability.
- Nonlinear mapping: Neural networks have inherent nonlinear characteristics due to their ability to approximate arbitrary nonlinear mapping (transformation).
- Learning through training: The neural network is trained through past data records of the system under study. A properly trained neural network has the ability to summarize all data.
- Adaptation and integration: The neural network is able to adapt to online operations and can perform both quantitative and qualitative operations. The strong adaptation and information fusion capabilities of the neural network enable the network process to simultaneously input a large number of different control signals, solve the problem of complementarity and redundancy between input information, and realize information integration and fusion processing.
- Hardware implementation: The neural network can implement parallel processing not only by software but also by software. In recent years, some VLSI implementation hardware has been introduced and is commercially available.

2.1.2 Artificial neural network structure

The structure of the neural network is determined by the basic processing unit and its interconnection method. The neuron unit shown in Fig. 2.2 consists of multiple inputs $x_i, i = 1, 2, \ldots, n$ and an output y. The intermediate state is represented by the weight sum

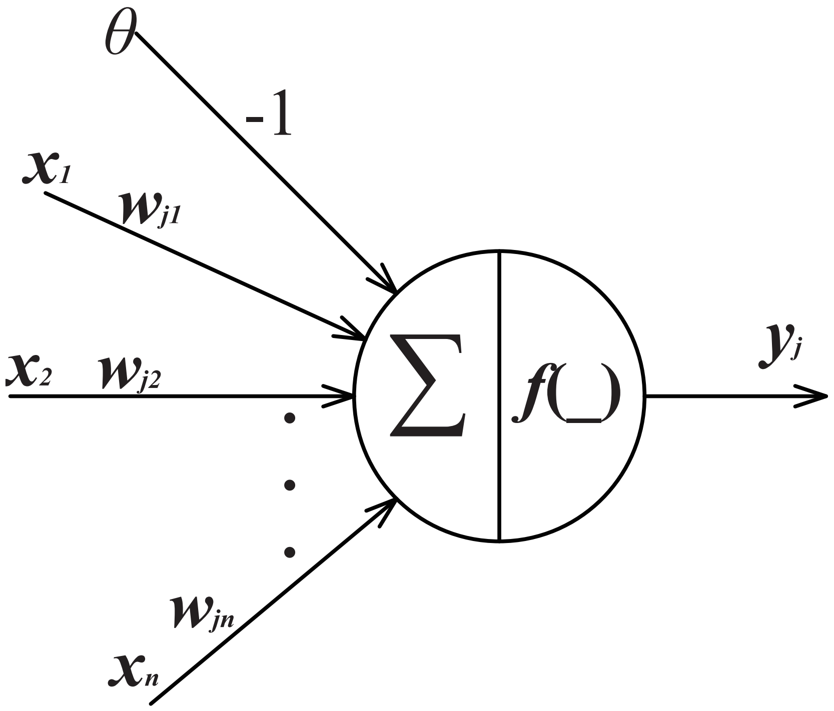


Figure 2.2: Neuron model.

of the input signal, and the output is:

$$y_j(t) = f(\sum_{i=1}^{n} w_{ji}x_i - \theta_j), \tag{2.1}$$

where θ_j is the bias of the neuron unit (threshold), w_{ji} is the connection weight coefficient (for the excited state, w_{ji} takes a positive value), n is the number of input signals, y_j is for neuron output, t is time, $f(\_)$ is an output transformation function, sometimes called an excitation function, often using 0 and 1 binary or sigmoid functions, as shown in Fig. 2.3, these three functions are continuous and non-linear. A binary function can be represented by the following formula:

$$f(x) = \begin{cases} 1, & x \geq x_0 \\ 0, & x < x_0 \end{cases}, \tag{2.2}$$

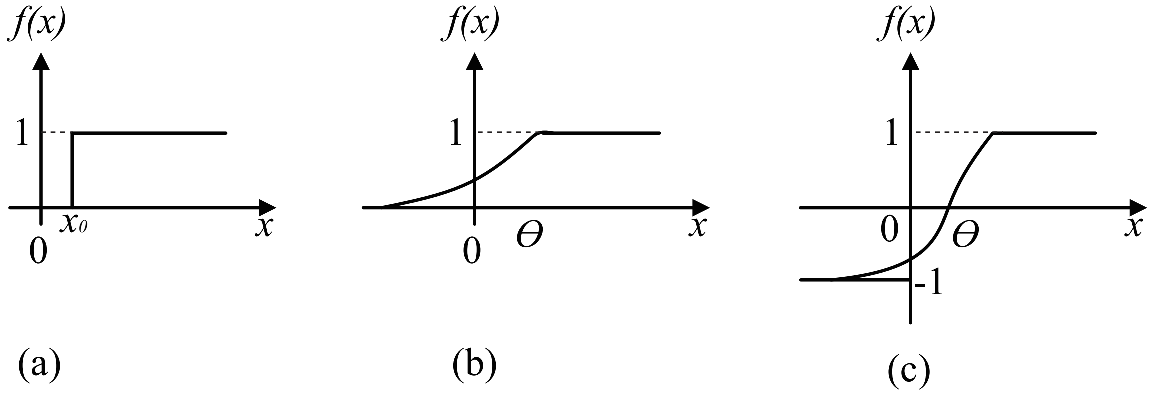


Figure 2.3: Some transform (excitation) functions in neurons.

as shown in Fig. 2.3. A conventional sigmoid function is shown in Fig. 2.3(b), which can be expressed by:

$$f(x) = \frac{1}{1 + e^{-\theta x}}, \ \ 0 < f(x) < 1. \tag{2.3}$$

Since the hyperbolic tangent function has a wider range of output values relative to the sigmoid function, the former output value can be positive or negative, so a hyperbolic tangent function is usually used instead of the sigmoid function. Fig. 2.3(c) is a hyperbolic tangent function. The commonly used hyperbolic tangent function formula is given below.

$$f(x) = \frac{1 - e^{-\theta x}}{1 + e^{-\theta x}}, \ \ -1 < f(x) < 1. \tag{2.4}$$

The artificial neural network is composed of a neuron model; this information processing network composed of many neurons has a parallel distribution structure. Each neuron has a single output and can be connected to other neurons; there are many (multiple) output connection methods, one for each connection method. Strictly speaking, an artificial neural network is a directed graph with the following characteristics:

- For each node i,there is a state variable x_i;
- From node j to node i, there is a connection right system number $w_(ji)$;
- For each node i, there is a threshold θ_i;

- For each node i, define a transformation function $f_i(x_i, w_{ji}, \theta_i), i \neq j$; for the most general case, this function takes the form $f_i(\sum_j w_{ji}x_j - \theta_i)$.

The structure of artificial neural networks is basically divided into two categories: recursive (feedback) networks and feedforward networks.

(1) Recursive network

In a recursive network, multiple neurons are interconnected to organize an interconnected neural network, as shown in Fig. 2.4. The output of some neurons is fed back to the same or anterior neurons. Therefore, the signal can flow from the forward and reverse directions. The Hopfield network, the Elmman network and the Jordan network are representative examples of recursive networks. A recursive network is also called a feedback network. In Fig. 2.1, V_i represents the state of the node, x_i is the input (initial) value of the

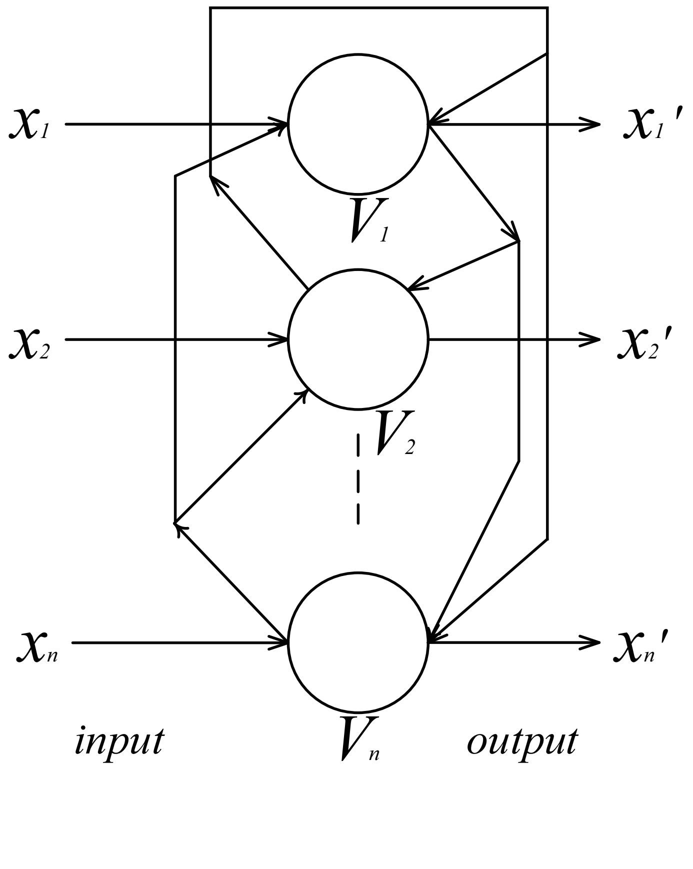


Figure 2.4: Recursive (feedback) networks.

node, and $x_i^{'}$ is the output value after convergence, $i = 1, 2, \ldots, n$.

(2) Feedforward network The feedforward network has the characteristics of disconnected neurons in the same layer, and its signal is unidirectional from the input layer to the output layer. As shown in Fig. 2.5, it is a hierarchical hierarchical structure. We use solid lines to indicate the flow of signals and dashed lines for back propagation.

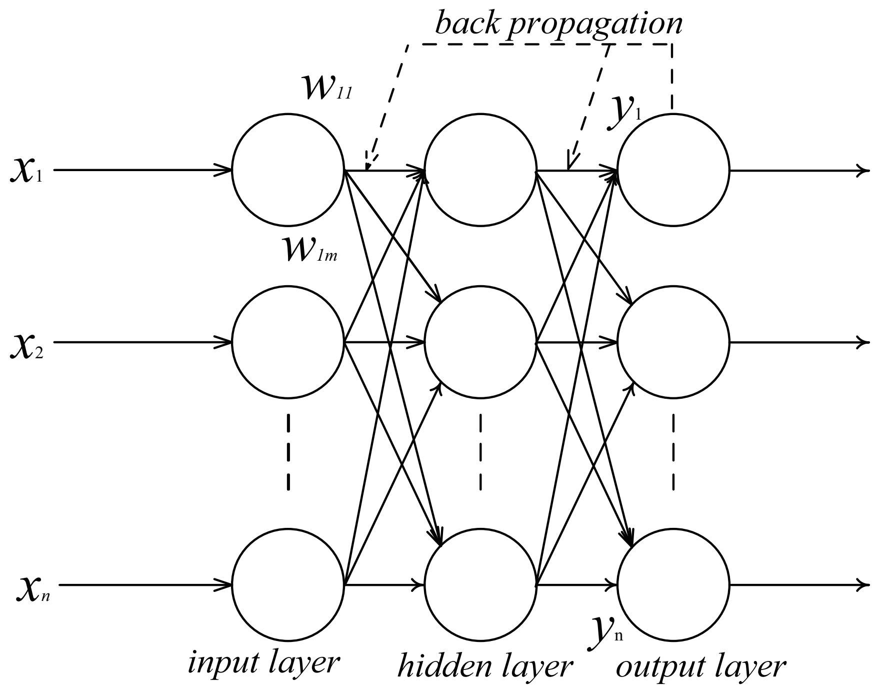


Figure 2.5: Feedforward (multilayer) network.

2.1.3 The main learning algorithm of artificial neural network

The neural network mainly uses a guided (with teacher) learning algorithm and a non-directed (no teacher) learning algorithm. In addition, there is a third learning algorithm, namely the reinforcement learning algorithm; it can be regarded as a special case of teacher learning.

(1) Teacher Learning (supervised algorithm)

A teacher learning algorithm can adjust the strength or weight of connections between neurons based on the difference between the desired and actual network output (corresponding to a given input). Therefore, a teacher needs to have a teacher or tutor to provide a desired or target output signal. Examples of learned algorithms include Delta rules, generalized Delta rules or backpropagation algorithms, and LVQ algorithms.

(2) No-teacher Learning (unsupervised learning)

The no-learning algorithm does not need to know the expected output. During the training process, as long as the input mode is provided to the neural network, the neural network can automatically adapt to the connection rights to group the input patterns into groups according to similar features. Examples of non-teacher learning algorithms include the Kohonen algorithm and the Carpenter-Grossberg Adaptive Resonance Theory (ART).

(3) Reinforcement Learning

As mentioned earlier, reinforcement (enhancement) learning is a special case of teacher learning. It does not require the teacher to give the target output. The reinforcement learning algorithm uses a "commenter" to evaluate the goodness (quality factor) of the neural network output corresponding to a given input. An example of a reinforcement learning algorithm is the genetic algorithm (GA).

The representation method of knowledge based on neural network system is completely different from the method used in traditional artificial intelligence system (such as production, framework, semantic network, etc.), the method used in traditional artificial intelligence systems is the explicit representation of knowledge, while the knowledge representation in neural networks is an implicit representation. Here, knowledge is not represented as each rule independently as in a production system, but rather a number of knowledge of a problem is represented in the same network.

Neural network based reasoning is achieved through network computing. The initial evidence provided by the user is used as an input to the network, and the output is finally obtained through network calculation.

In general, network inference has forward network inference, and the steps are as follows:

- Enter known data into each node of the network input layer.
- The output function is used to calculate the output of each layer in the network. In the calculation, the output of the previous layer is used as the input of the relevant

node in the next layer, and is calculated layer by layer until the output value of the output layer is calculated.

- The output of the output layer is determined by a threshold function to obtain an output result.

2.2 Evolutionary Computation

In the evolution of billions of years, organisms in nature have developed an intrinsic mechanism to optimize their structures, which can continually learn from the environment to adapt to changing environments. The survival process of biological groups generally follows Darwin's natural selection and evolutionary rules for the survival of the fittest; organisms adapt to the natural environment through individual choice, crossover, and variation [20]. The biological chromosome is represented by a mathematical or computer method as a series of numbers, still called chromosomes, sometimes called individuals; the ability to adapt is measured by the value of a corresponding chromosome; the choice or elimination of chromosomes is based on the maximum or minimum problem. For most organisms, this process is done through natural selection and sexual reproduction. Natural selection determines which individuals in the population can survive and reproduce: Sexual reproduction ensures the mixing and recombination of offspring genes [21–23]. Inspired by this evolutionary process of nature, evolutionary computation starts with the evolution of biological processes in the natural world, and explores the laws of the development and evolution of certain intelligent behaviors from the level of genes to solve the problem of how intelligent systems learn from the environment [5].

The theoretical basis of evolutionary computation is Darwin's theory of evolution, which is a new computational method formed by the combination of computer science and biological genetics [20]. Evolutionary computation uses simple coding techniques to represent complex structures and guides learning and determining search directions through simple genetic manipulation of a set of coded representations and natural selection strategies for survival of the fittest. Academic research is carried out through genetic manipulations such as replication, hybridization and mutation of the population. Evolutionary algorithms can search multiple points in different regions of the solution space. It can find global optimal solutions with great probability and is not easy to fall into local optimal conditions. Especially in the fields of system identification, fault diagnosis, machine learning and neural network design, evolutionary computing has shown its charm. However, as a new, interdisciplinary research topic, the theoretical research of evolutionary computation needs to be further improved, including basic theory, coding mechanism, selection strategy of control parameters, convergence analysis and so on.

2.2.1 Evolutionary algorithms

Since the 1960s, how to imitate organisms to build powerful algorithms and apply them to complex optimization problems has become a research hotspot. Evolutionary algorithms are born in this context. Evolutionary algorithms include genetic algorithms (GA), evolution strategies, evolutionary programming, and genetic programming.

Evolutionary algorithms are not a specific algorithm, but an "algorithm cluster". The inspiration of evolutionary algorithms draws on the evolutionary operations of organisms in nature. It generally includes basic operations such as gene coding, population initialization, cross mutation operators, and operational retention mechanisms. Compared with traditional optimization algorithms such as calculus-based methods and exhaustive methods, evolutionary computation is a mature global optimization method with high robustness and wide applicability. With self-organizing, self-adaptive, self-learning features, it can effectively deal with complex problems that traditional optimization algorithms are difficult to solve (such as NP-hard optimization problems) without being limited by the nature of the problem. The basic structure of the evolutionary algorithm is as follows:

```
{Randomly generate an initial population, calculate the fitness value of each individual in the population;
while Does not meet the termination criteria do
{Apply genetic manipulation (replication, hybridization, mutation, etc.) to generate
```

next-generation populations: Calculate the fitness values of each individual in the population}}.

The evolutionary algorithm knows nothing about the problem itself, but as long as the presentation scheme, adaptation function, genetic operator, control parameters, termination criteria, etc. are given. The algorithm can effectively search the unknown space in a way that does not depend on the problem itself, and finally find the solution to the problem.

In addition to the above advantages, evolutionary algorithms are often used in the optimization of multi-objective problems. We generally call such evolutionary algorithms evolutionary multi-objective optimization algorithms (MOEAs). At present, related algorithms for evolutionary computation have been widely used in parameter optimization, industrial scheduling, resource allocation, and complex network analysis.

2.2.1.1 Genetic algorithm

Genetic Algorithm (GA) is one of the most basic evolutionary algorithms. It is an optimization model for simulating Darwin's biological evolution theory [20]. It was first proposed by Professor J. Holland in 1975 [24]. In the genetic algorithm, each individual of the population is a feasible solution in the solution space. By simulating the evolution process of the organism, the optimal solution is searched in the solution space.

The structure of many application problems is complex, but can be turned into a simple bit string form coding representation. The process of transforming the problem structure into a bit string form code representation is called encoding; conversely, the process of transforming a bit string formatted representation into the original problem structure is called decoding or decoding. The bit string form code is called a chromosome, sometimes called an individual. The algorithmic process of GA is briefly described below. First, take a group of points in the solution space as the first generation of genetics. Each point (gene) is represented by a string of binary digits, the degree of which is measured by the fitness function.

The most common coding method for genetic algorithms is binary coding. One of the biggest drawbacks of binary encoding is the large length, which may be more advantageous for many problems with other primary encoding methods. Other coding methods mainly include: floating point number coding method, Gray code, symbol coding method, multi-parameter coding method, etc.

In order to reflect the adaptability of chromosomes, a function that can measure each chromosome in the problem is introduced, called the fitness function. The goal of TSP is to minimize the total length of the path [25, 26]. Naturally, the total length of the path can be used as a fitness function for the TSP problem. The fitness function should effectively reflect the gap between each chromosome and the optimal solution chromosome of the problem. The value of the fitness function has a great relationship with the meaning of solving the problem object.

There are three main genetic operations of simple genetic algorithms: selection, crossover, mutation. Improved genetic algorithms have greatly expanded genetic manipulation to achieve higher efficiency. (1) The selection operation, also called the reproduction operation, determines whether it is eliminated or inherited in the next generation according to the degree of the individual's fitness function. (2) The simple way of the crossover operation is to exchange the two partial individuals P1 and P2 as parent individuals and exchange the partial code values of the two. (3) The simple way of mutation operation is to change the number at a certain position of the digital string. The simple mutation operation represented by the binary code is to interchange 0 and 1: the 0 variation is 1, and the 1 variation is 0. The basic operation of the genetic algorithm can be described by Fig. 2.6.

After the individual coding mode is determined, the specific description of the operation of Fig. 2.6 is as follows:

Step1 Population initialization: design appropriate initialization operations according to the characteristics of the problem (initialization should be as simple as possible, time complexity is not too high) to initialize the N individuals in the population;

Step2 Individual evaluation: calculating the fitness value of the individual in the population according to the optimized objective function;

Step3 Iterative setting: set the maximum number of iterations of the population g_{max}, and make the current iteration number $g = 1$;

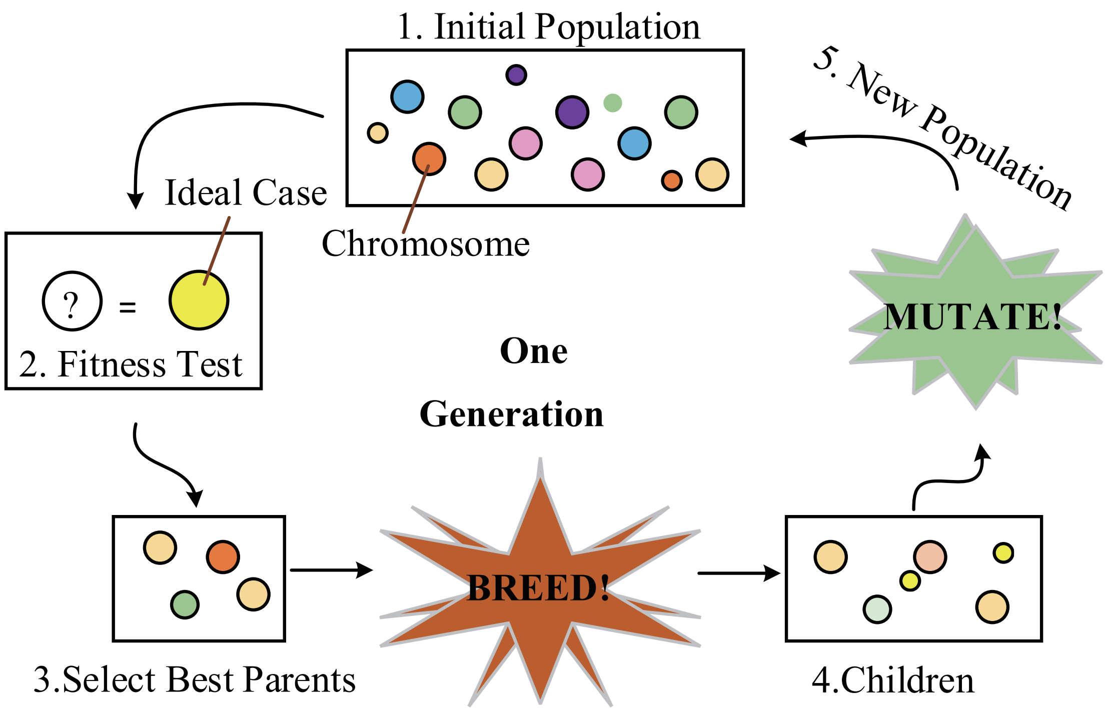


Figure 2.6: Basic operation of genetic algorithm.

Step4 Individual selection: Design a suitable selection operator to select the population $P(g)$individuals, and the selected individuals will enter the mating pool to form the parent population $FP(g)$ for cross transformation to generate new individuals. The selection strategy is based on individual fitness values. If the problem to be optimized is to minimize the problem, the probability that individuals with smaller fitness values are selected should be correspondingly larger. Common selection strategies include roulette selection, tournament selection, and more.

Step5 Crossover operator: Determine whether the parent individual needs to perform crossover operation according to the crossover probability pm (pre-specified, generally 0.9). The crossover operator is designed according to the characteristics of the problem being optimized. It is the core of the whole genetic algorithm. The quality of the design will directly determine the performance of the whole algorithm.

Step6 Mutation operator: According to the mutation probability pc (pre-specified, generally 0.1) to determine whether the parent individual needs to perform the mutation operation. The main role of the mutation operator is to maintain the diversity of the population and prevent the population from falling into local optimum, so it is generally designed as a random transformation.

After the cross mutation operation, the parent population $FP(g)$ generates a new child population $P(g + 1)$, and the population iteration number $g = g + 1$, and the next iteration operation (jump to Step 4) until the number of iterations reaches the maximum number of iterations.

In order to better illustrate the role of cross-operation, we use Fig. 2.7 as an example to understand the role of cross-operation:

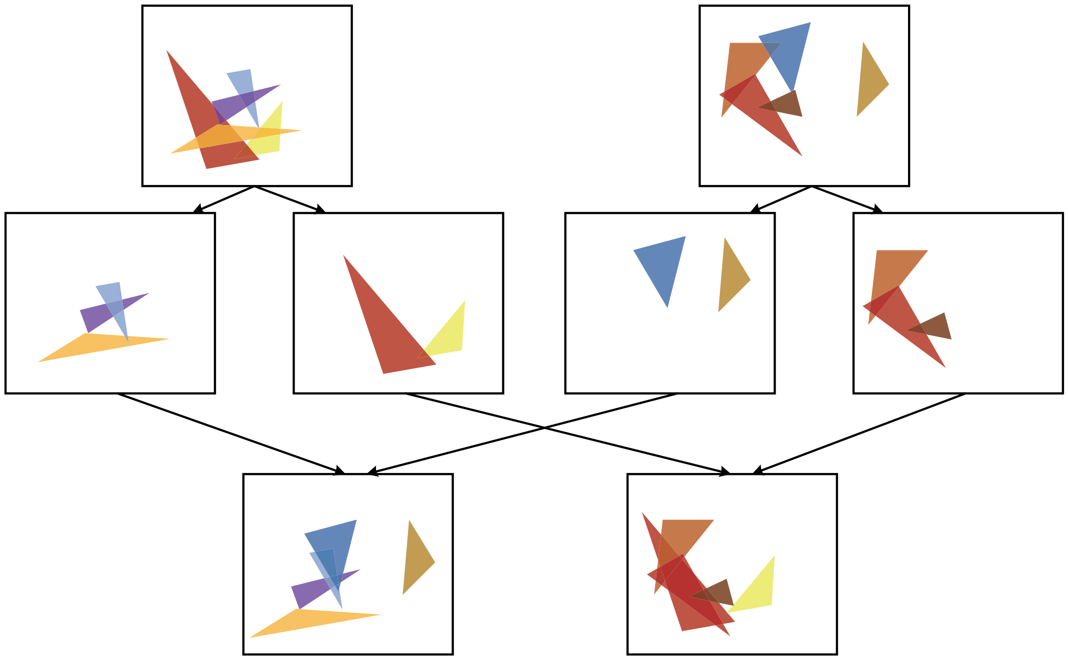


Figure 2.7: The role of cross-operation in genetic algorithms.

Through the crossover operation, the original two individual combinations generate two new individual combinations, which is equivalent to searching in the solution space, and each individual is a feasible solution to the solution space.

Genetic algorithm is a space-based search algorithm that simulates the natural evolution process to find the solution to the problem through natural selection, genetics, mutation and other operations and Darwin's theory of survival of the fittest [10, 20]. The genetic algorithm has the following characteristics: (1) The genetic algorithm is the encoding of the parameter set rather than the parameter itself; (2) The genetic algorithm starts with the code group of the problem solution and does not start with a single solution; (3) Genetic algorithms use the information of the fitness of the objective function instead of using derivatives or other ancillary information to guide the search; (4) Genetic algorithms use operators such as selection, intersection, mutation, etc. instead of using deterministic rules for random operations. A block diagram of a simple genetic algorithm is shown in Fig. 2.8. The simplest stopping conditions of the algorithm are as follows: (1) The completion of the predetermined evolutionary algebra is stopped; (2) The optimal individuals in the population do not improve over successive generations or the average fitness ceases when there are substantially no improvements in successive generations.

The main steps of the general genetic algorithm are as follows: (1) An initial population consisting of a characteristic string of a determined length is randomly generated. (2) Perform the following steps (a) and (b) on the string population iteration until the stop criteria are met: (a) Calculating the fitness value of each individual string in the population; (b) Calculating the fitness value of each individual string in the population; (3) The best individual string that appears in the offspring is specified as the result of the execution of the genetic algorithm, and this result can represent a solution to the problem.

2.2.1.2 Evolutionary programming

Evolutionary programming is a finite state machine evolution model proposed by L. J. Fogel in artificial intelligence research in the 1960s, in which the state of the machine is compiled based on the law of distribution [27]. D. B. Fogel expanded EP thinking in the 1990s to handle real-space optimization problems and introduced normal-distribution mutation operators in mutation operations, so that EP became an optimized search tool, and it has been applied in many practical problems [28]. EP simulates the evolution of biological population levels, so in the process of evolution, it mainly emphasizes the behavioral relationship of biological populations, that is, emphasizes the evolution of behavior at the population level to establish a behavioral chain between parents and children. This means that good offspring are eligible to survive, regardless of their parents, suitable for selecting

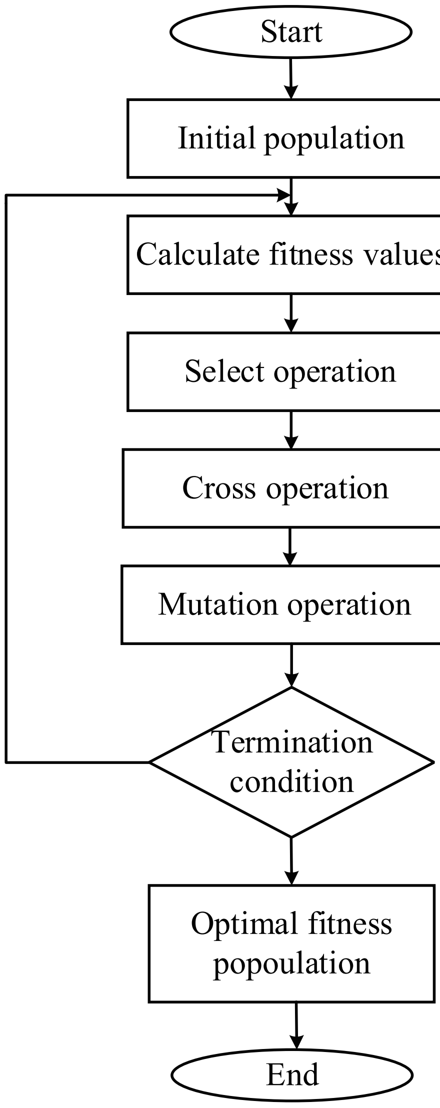


Figure 2.8: Simple genetic algorithm block diagram.

offspring. Algorithm 1 is the pseudo code of EP, the entire process can be explained as follows:

(1) Individual phenotype X : $X = (x_1, x_2, \ldots, x_n, \sigma_1, \sigma_2, \ldots, \sigma_n)$, where $x_1, \ldots, x_n$ is the component of individual performance, $\sigma_1, \ldots, \sigma_n$ is a variation parameter of the individual's performance component;

(2) Group size N: is the number of individuals included in the algorithm;

(3) Randomly generate initial populations: in the individual phenotype X, the individual

Algorithm 1: The pseudo code of EP.

```
Initialization (individual phenotype $X$, population size $N$, number of iterations $G$,
  etc)
Randomly generate initial population and calculate fitness values (containing $N$
  individuals)
while (not done) do
    for ($i = 1;\ i < N;\ i++$) do
        Mutating $X_i$ to get $X_i^{`}$
        Feasibility check on $X_i$
        Calculate the fitness value of $X_i$
    Select $N$ individuals from $2N$ individuals (use $q-$competition algorithm)
Output
```

x_i is initialized to a random value within the range of the component, σ_i is generated according to $N(0, 1)$ and calculates the fitness value of the individual;

(4) Variation of X_i to obtain $X_i^{`}$: can be mutated according to the following formula;

(5) Conduct a feasibility check on $X_i^{`}$ and calculate the fitness value: judging each component in $X_i^{`}$ within the range of values. (If it does not meet the value range, the processing method can be seen in the next section. If it is met, calculate the fitness value of the individual.);

(6) Select N individuals from $2N$ individuals: the method uses a random $q-$competition method.

The process of evolutionary programming can be understood as searching for individual computer programs with high fitness from the space formed by all possible computer programs. In evolutionary programming, there may be hundreds or thousands of computer programs involved in genetic evolution. Evolutionary programming was originally started by a randomly generated group of computer programs. These computer programs consist of functions that are appropriate for the problem space domain. Such functions can be standard arithmetic operations, standard programming operations, logic functions, or functions specified by the realm. Each computer program individual in the population is evaluated by fitness, which is related to a particular problem domain. The basic process of evolutionary programming is shown in Fig. 2.9.

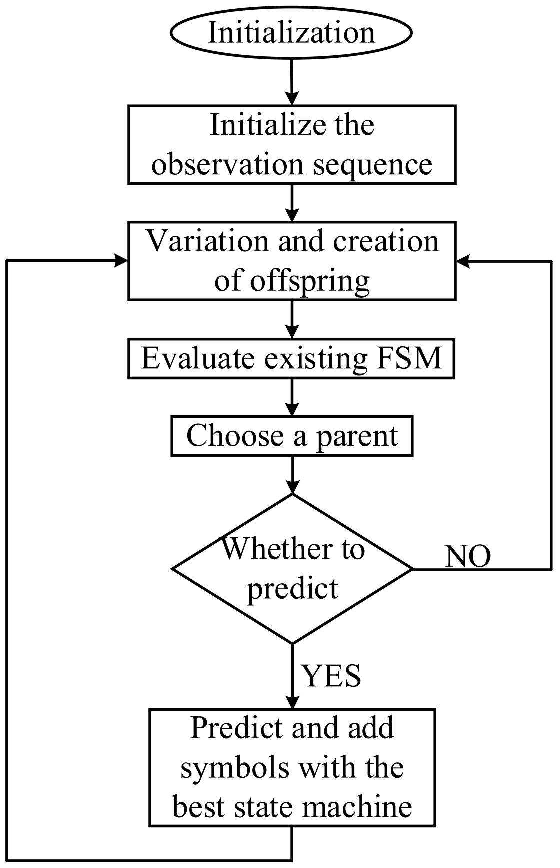

Initialization
Initialize the observation sequence
Variation and creation of offspring
Evaluate existing FSM
Choose a parent
Whether to predict
NO
YES
Predict and add symbols with the best state machine

Figure 2.9: Basic process of evolutionary programming.

2.2.1.3 Evolutionary strategy

Evolutionary strategy and genetic algorithm Evolutionary strategy (ES) is another optimization model that uses evolutionary theory. In addition to research and application areas, evolutionary strategies and genetic algorithms have the following differences: (1) Evolutionary strategies and genetic algorithms represent individuals in different ways. Evolutionary strategies run on floating-point vectors, while genetic algorithms typically run on binary vectors. (2) The selection process of evolutionary strategy and genetic algorithm is different. (3) Unlike the replication parameters of genetic strategies and genetic algorithms, the replication parameters of genetic algorithms (the possibility of crossover and mutation) remain constant during evolution, while evolutionary strategies change them from time to time. With the development of technology, the differences between evolutionary strategies and genetic algorithms are becoming less and less obvious.

Table 2.1: The difference between evolutionary strategy and genetic algorithm.

	Evolutionary Strategy	Genetic Algorithm
Reproduction	Breed first, then choose good children	Choose good parents, then breed
DNA	Usually DNA is a real number, such as 1.221	Usually use binary coded DNA
Variation	Variation of DNA by normal distribution	Perform mutation DNA by randomly changing 1 to 0

(1+1)-ES (1+1)-ES is a form of ES evolution strategy, and it is also a convenient and effective method in many forms. If you use a sentence to summarize (1+1)-ES: a war between a father and a child, which is, Have a father → According to Dad, a baby is mutated → Choosing the best one among Dad and baby becomes the next generation dad.

In kill_bad, we choose more suitable, whether it is dad or child, as long as it is suitable to leave, not suitable to kill. But there is also a point to note that in this step we have to make some changes to the intensity of the mutation. The method of change follows the 15 successful rule. The meaning of this principle is: When we are not converging, we increase the intensity of the mutation. If it is almost converging, we will reduce the intensity of variation. If one-fifth of the variation is better than the original parent, it will converge quickly.

Natural evolution strategy The natural evolution strategy (NES) is to calculate a gradient-induced method using fitness. The NES approach is actually very close to the policy gradient approach in reinforcement learning. Simply summarize their differences: In the behavioral strategy, the policy gradient is disturbing the action. The different action bring the different reward. The gradient is calculated by the reward size corresponding to the action, and the gradient is passed backwards. But ES is disturbing the parameters in the neural network. The different parameter bring the different reward. The original parameters are updated proportionally by the reward size corresponding to the parameters.

Algorithm 2: Canonical search gradient algorithm.

```
input: $f$, $\theta_{init}$
repeat
    for $k = 1 \ldots \lambda$ do
        draw sample $z_k \sim \pi(\cdot \mid \theta)$
        evaluate the fitness $f(z_k)$
        calculate log-derivatives $\nabla_\theta \log \pi(z_k \mid \theta)$
    $\nabla_\theta J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \, log \, \pi(z_k \mid \theta) \cdot f(z_k)$
    $\theta \leftarrow \theta + \eta \cdot \nabla_\theta J$
until stopping criterion is met
```

2.2.1.4 Genetic programming

Genetic programming itself is an optimization algorithm based on Darwin's theory of evolution, adopting the strategy of survival of the fittest and survival of the fittest [20]. Different from the optimal solution of genetic algorithm, genetic programming is generally used to solve the optimal solution structure, and the structure of the solution generally uses graph or tree structure. Unlike each individual in the genetic algorithm, which is a piece of chromosome coding, its individual is a computer program. The genetic plan was formally presented in a doctoral thesis published by Professor John R. Koza in 1990 [29]. The earliest application of genetic programming was symbolic regression. For example, in order to fit $z = f(x, y)$, it generates an initial function $g(x, y)$, and then uses the correlation coefficient of $g(x, y)$ and z as the fitness to select cross mutation, finally, get the best function g' and do another linear regression. The goal of the planning is to get the best computer program, the degree of freedom is the highest in machine learning algorithms, and can handle almost any problem. Of course, being able to deal with the problem does not mean that it can be solved, and solving the problem still requires precision to meet certain requirements.

The mathematical expression of genetic programming: Definition $\{(x_i, y_i);\ x_i \in X, y_i \in Y, i \in I\}$ is a given set of input and output, where X and Y are subsets in a finite dimensional space, I is the indicator set; $C(X)$ is the total of all continuous functions on X, F is a subset on $C(X)$, and ρ is the distance on the product space $\Pi(i \in I)$. Then the genetic programming is to solve $f^{`} \in F$, so that any $f \in F$ satisfies the formula $\rho(\{f^{`}(x_i)\}, \{f^{`}(y_i)\}) \leq \rho(\{f(x_i)\}, \{f(y_i)\})$. The algorithm flow chart is shown in Fig. 2.10. The specific algorithm steps are as follows:

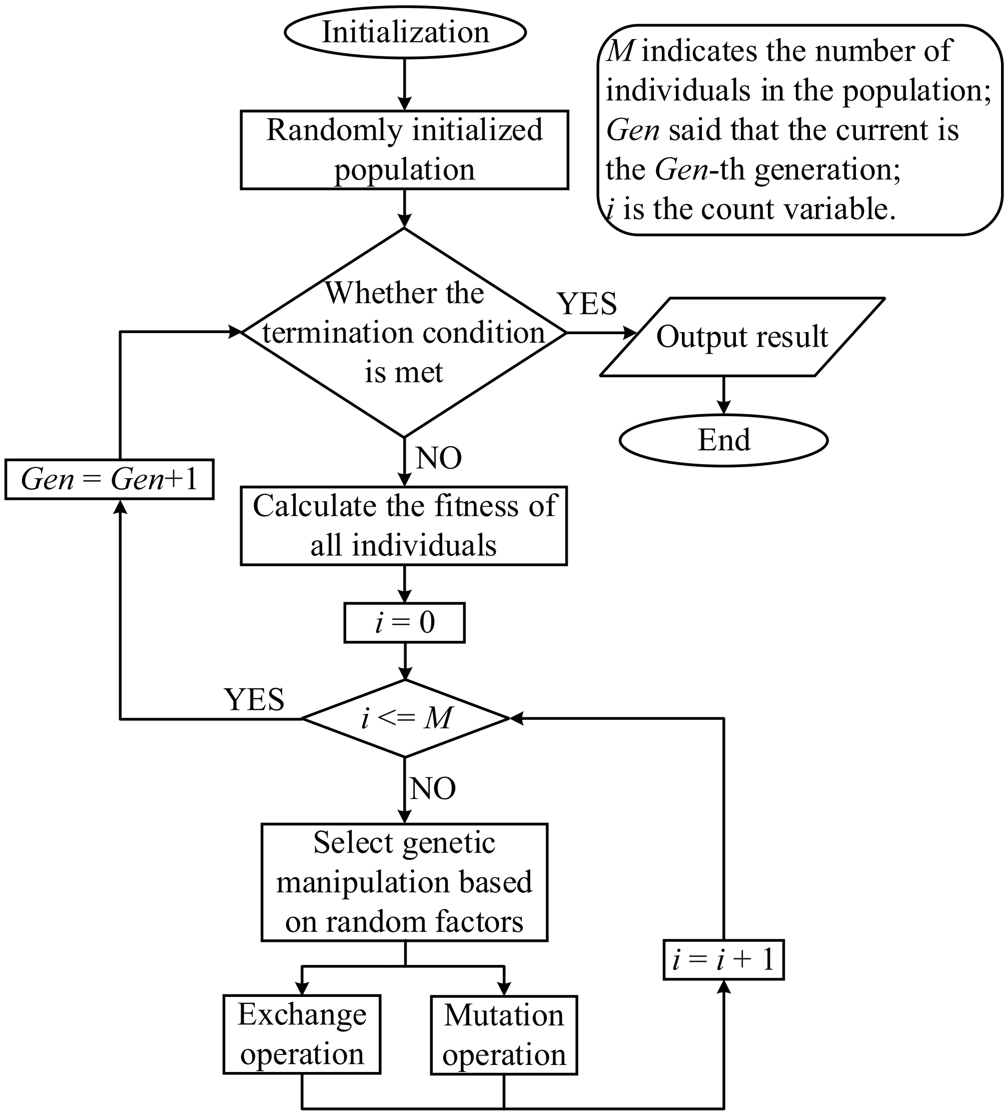

Initialization
Randomly initialized population
M indicates the number of individuals in the population; Gen said that the current is the Gen-th generation; i is the count variable.
Whether the termination condition is met
YES
Output result
End
NO
Gen = Gen+1
Calculate the fitness of all individuals
i = 0
YES
i <= M
NO
Select genetic manipulation based on random factors
i = i + 1
Exchange operation
Mutation operation

Figure 2.10: Genetic programming algorithm flow chart.

(1) Initialization: Randomly generate multiple individuals to initialize the population.

(2) Evaluation: Determine the appropriate fitness function and evaluate all individuals.

(3) Selection: Selecting the next generation of individuals in the population by fitness function and random factor.

(4) Crossover: Randomly select the subtrees of two individuals for exchange.

(5) Mutation: Randomly select one node of the individual, and replace the subtree rooted at the node with a randomly generated mutation tree.

(6) Terminal Criterion: Repeat steps 2 ~ 5 until the termination condition is met. Termination condition, the occurrence is the individual whose fitness meets the demand or the number of cycles.

Application in the Renascence architecture The Renascence architecture uses genetic programming algorithms to dynamically determine the formula of the ADF. But with the GP algorithm described above, there is a big difference: (1) In the Renascence architecture, each function has its own input and output type, and the output type of the child node must match the input type of the parent node. (2) The implementation of the computer program generated by Renascence depends on the underlying function library, not a complete computer program. Because of these differences, both the generation and mutation algorithms need to first find all possible solutions using the backtracking algorithm, and then randomly select them, which is much more complicated than the traditional genetic programming algorithm.

The tree implementation is the easiest to implement, and the tree representation is currently used in the Renascence architecture. Because the crossover operator is cumbersome to implement and easy to cause expansion, the crossover operator is discarded, and the algorithm of Evolution Strategy is used for optimization. The structural optimization algorithms used in the current architecture actually have a lot of room for optimization.

2.2.2 Swarm Intelligence

Swarm Intelligence (SI) refers to the decentralized, decentralized self-organizing behavior that is expressed at the collective level [17]. For example, the complex social system consisting of ant colonies and bee colonies, the migration of birds, fish populations to adapt to air or seawater, and the collective intelligence of microorganisms and plants in adapting to the living environment. Cluster intelligence refers to cluster intelligence, which is cluster intelligence, if there are many unintelligent individuals in a certain group, and they exhibit intelligent behavior through simple cooperation with each other [18, 19]. The term SI was first proposed by Gerardo and Jing Wang in 1989, when it was proposed for the self-organization of cell robots on computer screens [30]. The most well-known cell robot system has Langton's ants [31] and Conway's life game [32]. Depending on the simple motion rules of each cell, the movement of the cell collection can exhibit extraordinary intelligent behavior. Swarm Intelligence is not a simple collection of multiple bodies, but a higher-level performance that transcends individual behavior. The evolution from individual behavior to swarm behavior is often extremely complex and unpredictable. Millonas [33] proposed in 1993 that swarm intelligence should follow five basic principles, namely:

(1) Proximity Principle:the swarm is able to perform simple space and time calculations;

(2) Quality Principle: the swarm is able to respond to quality factors in the environment;

(3) Principle of Diversity Response: the scope of action of the swarm should not be too narrow;

(4) Stability Principle: the swarm should not change its behavior every time the environment changes;

(5) Adaptability Principle: at a time when the cost is not too high, the group can change its behavior at the appropriate time.

These principles suggest that intelligent agents that implement swarm intelligence must be able to demonstrate intelligent characteristics such as autonomy, responsiveness, learning, and adaptability in the environment. However, this does not mean that every individual in the group is quite complicated, and the fact is exactly the opposite. Just as a single ant is not intelligent, each individual that makes up a group has only simple intelligence, and they exhibit complex intelligent behavior through cooperation with each other. It can be said that the core of group intelligence is that a group of simple individuals can achieve a certain function and accomplish a certain task through simple cooperation with each other. Among them, "simple individuals" means that a single individual has only simple abilities or intelligence. “Simple cooperation” refers to the simple direct communication between an individual and an individual adjacent to it or indirectly communicating with other individuals through changing the environment, so that they can interact and coordinate with each other.

Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) are the two most widely known “cluster intelligence” algorithms. At a basic level, these algorithms use multiple agents. Each agent performs very basic actions, which together are more complex and more immediate actions that can be used to solve problems. Ant colony optimization is different from particle swarm optimization. The purpose of both is to perform immediate actions, but in two different ways. Similar to real ant colonies, ACO uses information hormones to guide individual agents to the shortest path. Initially, random information hormones are initialized in the problem space. A single agent begins to traverse the search space and sprinkles information hormones while walking. The information hormone decays at a certain rate in each time step. A single agent determines the path to traverse the search space based on the strength of the information hormone in front. The greater the intensity of the information hormone in one direction, the more likely the agent is to move in this direction. The global optimal solution is the path with the strongest information hormone.

PSO is more concerned with the overall direction. Multiple agents are initialized and proceed in a random direction. In each time step, each agent needs to make a decision on whether to change direction. The decision is based on the direction of the global optimal solution, the direction of the local optimal solution, and the current direction. The new direction is usually the optimal ”trade-off” result for the above three values.

2.2.2.1 Ant colony optimization algorithms

It is not an accident that the ant colony can build a body pontoon across the gap. An ant colony may have built more than 50 ant bridges at the same time, each ranging from 1 ant to 50 ants. Ants can not only build bridges, but also effectively assess the balance between cost and efficiency of bridges. For example, on a V-shaped road, the ant colony will automatically adjust to the appropriate position to build the bridge, neither near the

V vertex part nor the largest part of the V opening. The surface of the algorithm used by biologists to study ant colony bridges. Each ant does not know the overall shape of the bridge. They are only following two basic principles: (1) If there are other ants on my body, then I will stay still; (2) If the frequency of ants passing through me is below a certain threshold, I join the march and not serve as a bridge.

Dozens of ants can form a raft together to cross the water. When ant colonies migrate, the entire raft may contain tens of thousands or more ants. Every ant does not know the overall shape of the raft, nor the direction in which the raft will drift. But the ants are very cleverly connected to each other to form a three-dimensional structure that is breathable and impervious, even if the ants at the bottom of the water are completely submerged. This structure also allows the entire raft to contain more than 75% of the air volume, so it can float smoothly on the water surface.

Ant colonies often form very complex routes for food and food handling on the ground. It seems that the whole group is always able to find the best food and the shortest route, but each ant does not know how this intelligence is formed. Each ant follows only two basic rules: (1) Ants looking for food leave a stronger bio-pheromone on a higher quality route; (2) Ants tend to join a stronger pheromone route and feedback with other ants during constant round-trips, allowing shorter routes to be reinforced. However, the use of mothballs on the route that ants pass through can cause ants to get lost because the strong smell of camphor severely interferes with the identification of ant bio pheromones. Scientists have inspired the ant colony dependent information to obtain the optimal path, and created the ant colony optimization algorithm (ACO). The ant colony optimization algorithm was first proposed by M.Dorigo. in 1991 and was inspired by the social behavior of ant colonies in nature [13, 34]. It simulates the process of actual ant colony searching for food. In nature, ant colonies are always able to find a shortest path from the nest to the food source. This is because ants can leave a substance called ‘pheromone” on the path they travel during exercise. The substance can be perceived by later ants and will gradually evaporate over time. Each ant directs its direction of movement based on the strength of the pheromone on the path and tends to move in the direction of the strength of the substance. Therefore, if there are more ants walking on a certain path, the more pheromones accumulated, the greater the intensity, and the greater the probability that the path will be selected by other ants in the next time. Since the shorter the path is accessed by more ants in a certain period of time, as the above process proceeds, the entire ant colony will eventually find the shortest path from the ant colony to the food [14].

The ant colony optimization algorithm uses this feature of the biological ant colony to solve the problem. Since the process of ant feeding is very similar to the solution of Traveling Salesman Problem (TSP), the earliest application of ant colony optimization algorithm is the solution of TSP problem. At present, the ant colony optimization algorithm has been applied in the combination optimization problem solving, the scheduling and distribution of various resources such as vehicles, shops, personnel, as well as power, communication, chemical, transportation, robotics, metallurgy and other fields. Here is the basic idea of the ACO algorithm:

- Set up multiple ants according to specific problems and search in parallel.
- After each ant completes a trip, the pheromone is released on the way, and the amount of pheromone is proportional to the quality of the solution.
- The choice of the ant path is based on the pheromone intensity (the initial pheromone is set equal), taking into account the distance between the two points, using a random local search strategy. This makes the distance of the shorter side, the amount of pheromone on it is larger, and the probability that the later ant chooses the side is also larger.
- Each ant can only take the legal route (passing once per city and only once), and set a taboo table to control this.
- All ants search once and iterate once. Once every iteration, they do a pheromone update for all the edges. The original ants die and the new ants perform a new round of searching.
- Updating pheromones includes the evaporation of the original pheromone and the increase in pheromones on the path that passes.

- When the predetermined number of iteration steps is reached, or stagnation occurs (all ants choose the same path, the solution no longer changes), the algorithm ends, and the current optimal solution is taken as the optimal solution of the problem.

The transition probability $P_{ij}^k(t)$ and the heuristic factor η in ACO are defined as follows:

$$P_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}(t)]^\beta}{\sum\limits_{s \in J_k(i)} [\tau_{is}(t)]^\alpha \cdot [\eta_{is}(t)]^\beta}, & \text{if j} \in \text{J}_\text{k}\text{(i)} \\ 0, & \text{otherwise} \end{cases}, \tag{2.5}$$

where α indicates the relative importance of pheromone; $\beta isrelativeimportanceofheuristicfactors$; $J_k(i)$ represent the city collection of ants k next allowed to choose.

$$\eta_{ij} = \frac{1}{d_{ij}}, \tag{2.6}$$

when all ants complete a trip, the pheromone on each path is:

$$\tau_{ij}(t+n) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}, \tag{2.7}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k, \tag{2.8}$$

$$\Delta\tau_{ij}^k = \begin{cases} \dfrac{Q}{L_k}, & \text{If ant k passes through this side of the tour (i, j)} \\ 0, & \text{otherwise} \end{cases}, \tag{2.9}$$

where Q is a positive constant, L_k is the length of the path taken by ant k in this tour. At the beginning, let $\tau_{ij}(0) = C$. Fig. 2.11 provides an ant colony optimization algorithm flow chart An explanation of the flow chart steps for the ant colony optimization algorithm is given below:

(1) Initialization parameters: The amount of pheromone on each side is equal at the beginning. $\tau_{ij}(0) = C, \Delta\tau_{ij}(0) = 0$.

(2) Place each ant on each vertex, and the taboo table is the corresponding vertex.

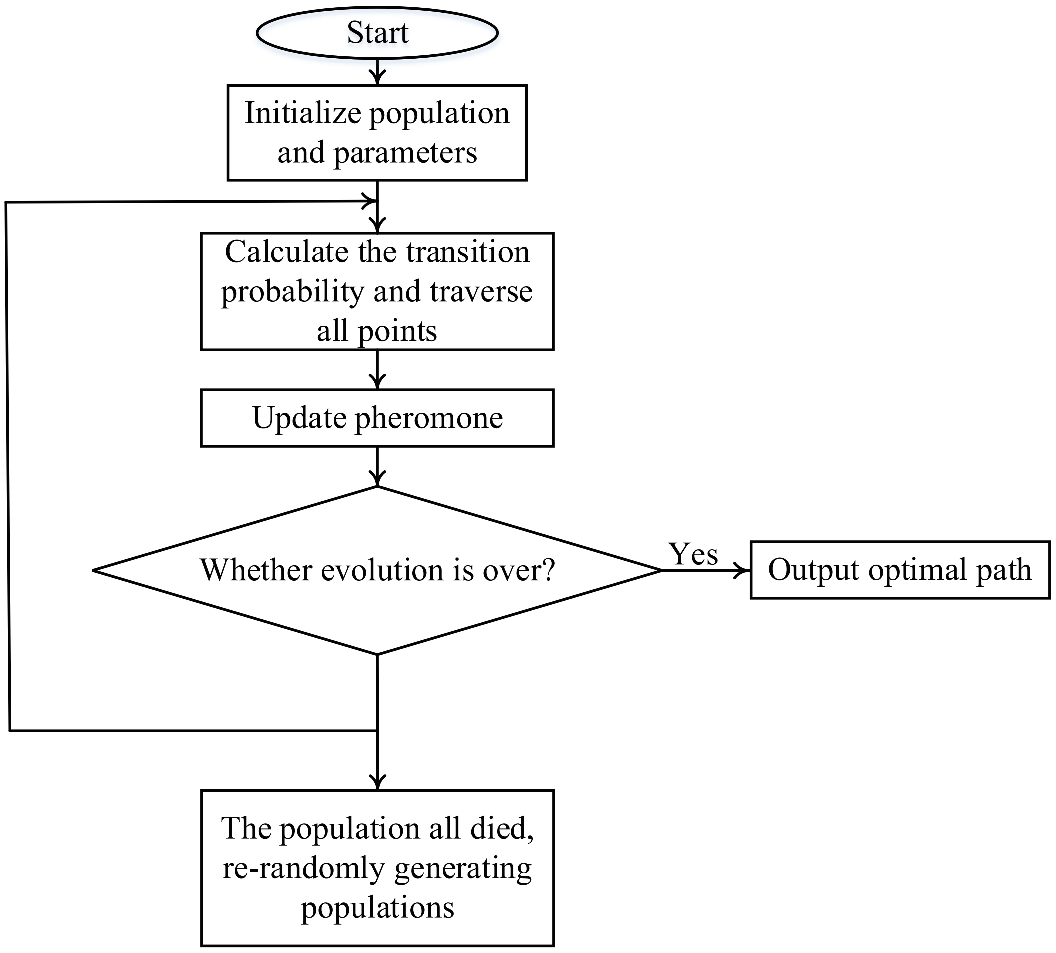

Start
Initialize population and parameters
Calculate the transition probability and traverse all points
Update pheromone
Whether evolution is over?
Yes
Output optimal path
The population all died, re-randomly generating populations

Figure 2.11: Flow chart of ant colony optimization algorithm.

(3) Take one ant, calculate the transition probability $P_{ij}^{k}(t)$, select the next vertex according to the way of roulette, and update the taboo table. Then calculate the probability, select the vertex, and then update the taboo table until you traverse all the vertices once.

(4) Calculate the amount of pheromone $\Delta\tau_{ij}^{k}$ left by each ant on each side, and the ant is no longer executed.

(5) Repeat step(3) step(4) until m ants are gone around.

(6) Calculate the pheromone increment $\Delta\tau_{ij}$ and the pheromone amount $\tau_{ij}(t+n)$ on each side.

(7) Record the path of this iteration, update the current optimal path, and clear the taboo table.

(8) Determine if the predetermined number of iteration steps is reached, or if there is a stagnation. If yes, the algorithm ends and the current optimal path is output; otherwise, go to step(2) and proceed to the next iteration.

Although the individual behavior of ants is extremely simple, the ant colony composed of individuals constitutes a highly structured social organization. The members of the ant society have division of labor and mutual communication and information transmission. During the foraging process, the ant colony always finds the shortest path from the nest to the food source without any prompts; when obstacles appear on the passing route, the new optimal path can be quickly found. Ants release pheromones on their path as they seek food. Ants have basically no vision, but they can detect the trajectory of the same pheromone in a small range, thereby determining where to go and tend to move in the direction of high pheromone intensity. The more ants that pass on a path, the more pheromones remain (evaporating a portion over time), and the higher the probability that the ant chooses the path.

2.2.2.2 Particle swarm optimization algorithm

Particle Swarm Optimization (PSO) was first proposed by Eberhart and Kennedy in 1995 [15]. Its basic concept stems from the study of the foraging behavior of birds. Imagine a scene where a group of birds are searching for food randomly, there is only one piece of food in this area, and all birds don't know where the food is, but they know how far the current location is from the food. The simplest and most effective strategy is to search for individuals who are closest to the food in the flock. PSO algorithm is inspired by this biological population behavior and used to solve optimization problems [35].

A particle is used to simulate the above-mentioned individual birds. Each particle can be regarded as a search individual in the N-dimensional search space. The current position of the particle is a candidate solution to the optimization problem, and the flight process of the particle is the individual's search process. The flight speed of particles can be dynamically adjusted according to the optimal position of the particle history and the optimal position of the population history. Particles have only two properties: speed and position, speed represents the speed of movement, and position represents the direction of movement [36]. The optimal solution for each particle to search separately is called the individual extremum, and the optimal individual extremum in the particle swarm is taken as the current global optimal solution. Constant iteration, update speed and location. Finally, an optimal solution that satisfies the termination condition is obtained. The standard PSO algorithm flow chart is shown in Fig. 2.12:

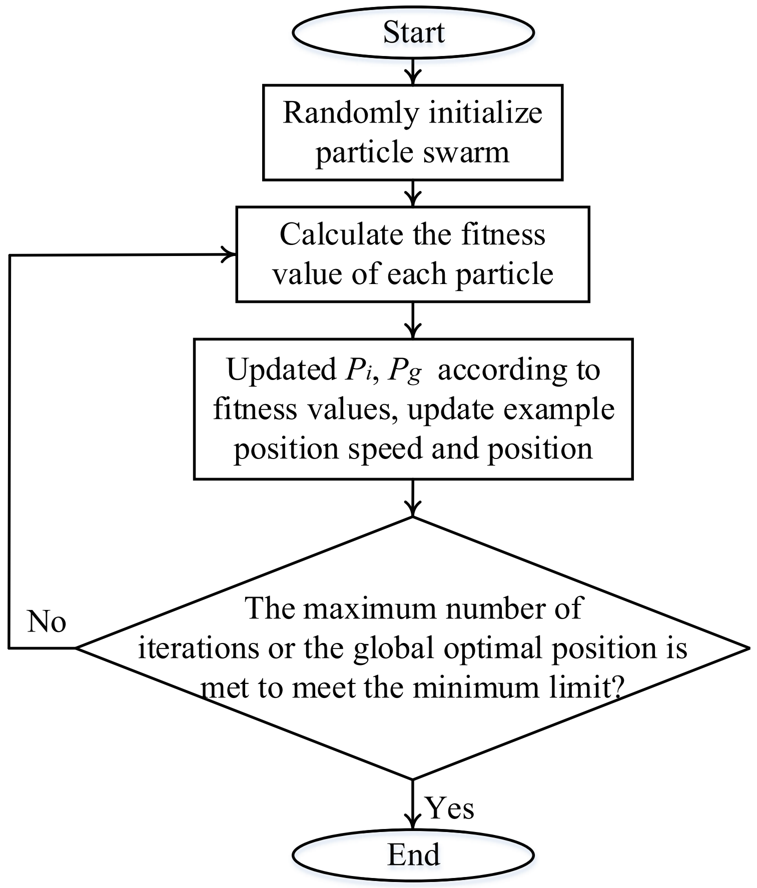

Start
Randomly initialize particle swarm
Calculate the fitness value of each particle
Updated P_i, P_g according to fitness values, update example position speed and position
The maximum number of iterations or the global optimal position is met to meet the minimum limit?
No
Yes
End

Figure 2.12: Flow chart of particle swarm optimization algorithm.

First, we set the maximum number of iterations, the number of arguments of the objective function, the maximum speed of the particle, and the location information for the entire search space. We randomly initialize the velocity and position in the velocity interval and the search space, set the particle swarm size to M, and each particle randomly initializes a flying speed. Define the fitness function, the individual extremum is the optimal solution found by each particle, and find a global value from these optimal solutions, called the global optimal solution. Update with the historical global optimal. The termination conditions are: (1) the set number of iterations is reached; (2) the difference between the algebras satisfies the minimum bound. The formula for updating the speed and position is as follows:

$$V_{id} = \omega V_{id} + C_1 random(0,1)(P_{id} - X_{id}) + C_2 random(0,1)(P_{gd} - X_{id}), \tag{2.10}$$

$$X_{id} = X_{id} + V_{id}, \tag{2.11}$$

where ω is called the inertia factor and its value is non-negative. When it is large, the overall optimization ability is strong, and the local optimization ability is strong; when it is small, the global optimization ability and the local optimization ability are weak. By adjusting the size of ω, the global optimization performance and local optimization performance can be adjusted. C_1 and C_2 are individual learning factors and social learning factors for each particle, respectively, which are called acceleration constants. Generally, $C_1 = C_2 \in [0,4]$, usually takes a value of 2. $random(0,1)$ represents the random number in the interval $[0,1]$, P_{id} represents the dth dimension of the individual extremum of the i-th variable, P_{gd} represents the dth dimension of the global optimal solution.

Like other cluster intelligent algorithms, there is always a contradiction between the diversity of the population and the convergence speed of the algorithm in the optimization process of the PSO algorithm. Improvements to the standard PSO algorithm, whether it is the selection of parameters, the adoption of niche technology or the integration of other technologies with PSO. The purpose is to maintain the diversity of the population while maintaining the local search ability of the algorithm, and to prevent the premature convergence of the algorithm while fast convergence.

PSO each particle in a particle swarm represents a possible solution to a problem. Through the simple behavior of the individual particles, the information interaction within the group realizes the intelligence of the problem solving. Due to its simple operation and fast convergence, PSO has been widely used in many fields such as function optimization, image processing, and geodetic survey. With the expansion of the application scope, the PSO algorithm has problems such as premature convergence, dimensionality disaster, and easy to fall into local extremes. The main development directions are as follows.

- Adjust the parameters of the PSO to balance the global detection and local mining capabilities of the algorithm. For example, Shi and Eberhart [35] introduce inertia weights for the velocity term of the PSO algorithm, and linearly (or non-linearly) dynamically adjust the inertia weights according to the iterative process and particle flight conditions to balance the globality and convergence speed of the search. In 2009, based on the stability analysis of the positional expectation and variance of the standard particle swarm optimization algorithm, Zhang [37] studied the influence of the acceleration factor on the position expectation and variance, and obtained a set of better acceleration factor values.

- Designing different types of topologies and changing particle learning patterns to increase population diversity, Kennedy [16] studied the effects of different topologies on SPSO performance. In view of the shortcomings of SPSO, such as easy premature convergence and low precision, a more explicit particle swarm algorithm was proposed in 2003: the bare bones particle swarm algorithm (BBPSO) [38].

- Combine PSO with other optimization algorithms (or strategies) to form a hybrid PSO algorithm. For example, Zeng Yi [39] embedded the pattern search algorithm into the PSO algorithm, which realized that the local search ability of the pattern search algorithm is complementary to the global search ability of the PSO algorithm.

- Adopt niche technology. Niche is a bionic technique that simulates ecological balance and is suitable for optimization of multimodal functions and multi-objective functions. For example, in the PSO algorithm, by constructing a niche topology, the population is divided into several sub-populations, and a relatively independent search space is dynamically formed to realize synchronous search for multiple extreme regions. Therefore, it can avoid the premature convergence phenomenon when the algorithm solves the multi-peak function optimization problem. Parsopoulos proposed a multi-group PSO algorithm based on the idea of 'divide and conquer" [40]. The core idea is to decompose the high-dimensional objective function into multiple low-dimensional functions, and then each low-dimensional sub-function is optimized by a sub-particle swarm. This algorithm provides a better idea for solving high-dimensional problems.

Different development directions represent different application fields, some need to continuously perform global detection, some need to improve the precision of optimization, some need the balance between global search and local search, and some need to solve high-dimensional problems. There is no such thing as good or bad in these directions, only the difference between the most appropriate method for solving different problems in different fields.

2.2.3 Artificial Immune Systems

The biological immune system is an adaptive, self-organizing, distributed system that is a complex defense system that can withstand foreign pathogens [10]. The artificial immune system is an emerging algorithm inspired by the biological immune system to solve problems in the computer field [41–43]. The immune system is a defense system against mammals against foreign viruses. Animals may encounter various injuries during their life, and the immune system plays an important role in their normal activities. A major feature of the immune system is the effective response to a large number of virus intrusions with a limited variety of resources. Inspired by this feature, a new algorithm for multi-peak search and global optimization of multi-peak functions has been designed [44–46]. This algorithm is called the Immune Algorithm (IA). The immune algorithm is based on the humoral immune process of the organism. The mechanism of biological humor immunity: the immune system can recognize antigens and generate different plasma cells according to the characteristics of different antigens to produce antibodies [47, 48]. If the affinity of the produced antibody with the antigen is high, it will remain, otherwise it will be screened off. B cells differentiate into plasma cells and memory cells, and the memory cells preserve antibody information with high affinity [49]. Plasma cells that produce high affinity antibodies are promoted, and vice versa. The next generation of antibodies is produced by cross mutation. The immune algorithm has the following features: cloning and mutation of the antibody helps to generate new antibodies; the convergence speed is fast, that is, the time required to produce the optimal solution satisfying the requirement is short [50–52].

Table 2.2: Correspondence between biological immunity and immune algorithm concepts.

Biological disposable system	Immune algorithm
Antigen	Optimization
Antibody	A feasible solution to the optimization problem
Affinity	Quality of feasible solution
Cell activation	Immune selection
Cell Differentiation	Individual clone
Affinity mature	Variation
Clonal inhibition	Excellent individual choice
Dynamic steady state maintenance	Population refresh

The objective function and various constraints are input to the immune algorithm as antigens of the immune algorithm, and the affinity function is selected. The antibody is produced in a random manner in the solution space. The affinity between the antigen and the antibody was calculated and ordered separately. An antibody having high affinity for an antigen is added to a memory unit, and an immunological operation is performed. Antibodies entering the next generation are produced by crossover and mutation and population refresh. The generation and selection of memory cells is terminated when the specified threshold is reached. When applying the immune algorithm to solve the actual problem, the affinity between the antigen and the antibody is often corresponding to the objective function of the optimization problem, the optimization solution, and the matching degree between the solution and the objective function [53]. A flow chart of an artificial immune algorithm based on Euclidean distance is given in Fig. 2.13.

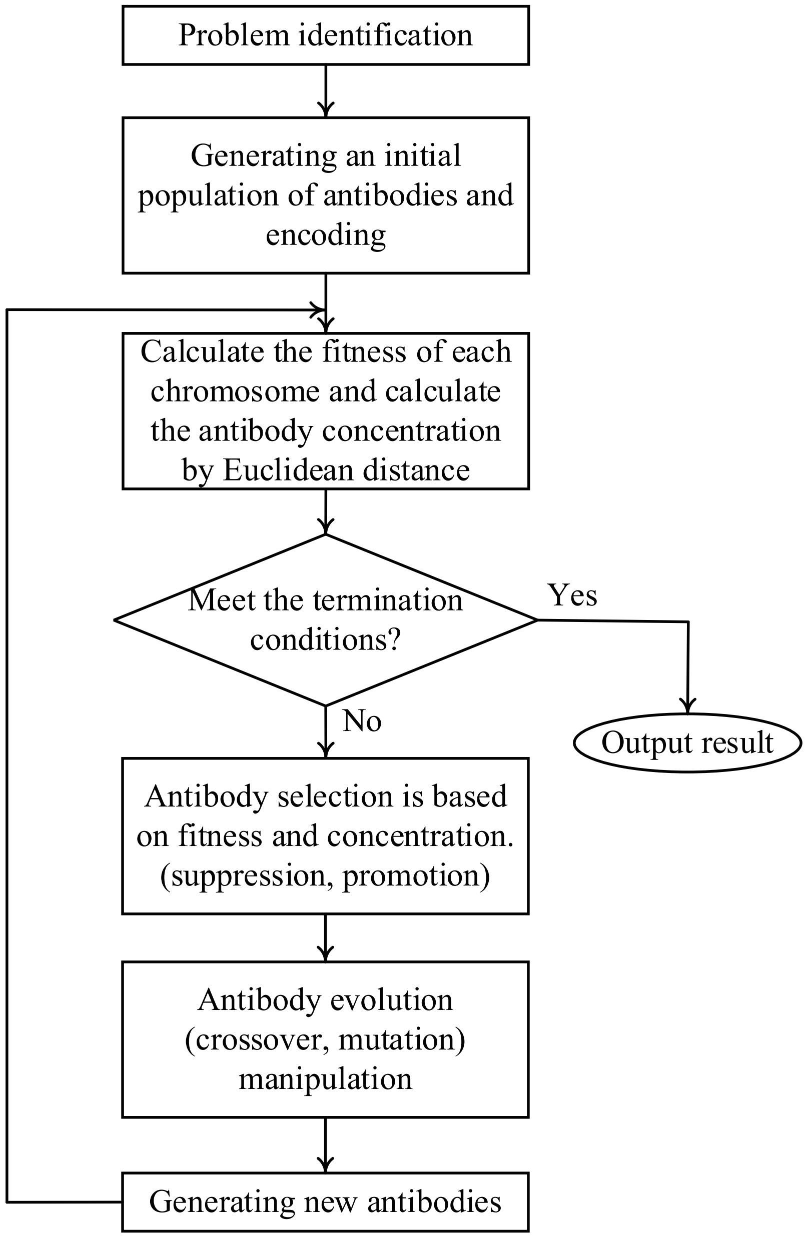

Problem identification
Generating an initial population of antibodies and encoding
Calculate the fitness of each chromosome and calculate the antibody concentration by Euclidean distance
Meet the termination conditions?
Yes
No
Output result
Antibody selection is based on fitness and concentration. (suppression, promotion)
Antibody evolution (crossover, mutation) manipulation
Generating new antibodies

Figure 2.13: Flow chart of artificial immune algorithm based on Euclidean distance.

2.2.4 Fuzzy Systems

In order to represent and deal with many inaccuracies and uncertainties in the real world, Zadeh proposed fuzzy set theory in 1965 [4]. In the fuzzy set, the boundary of the set is not clear, and the qualification of the set member is not affirmative or negative. It uses the membership function to describe the intermediate transition of the phenomenon difference, thus breaking through the absolute relationship in the classical set that belongs to or does not belong. In a fuzzy set, each individual is assigned a value to indicate the extent to which it belongs to the set. This value reflects the degree to which the individual approximates the concept represented by the fuzzy set: the greater the membership, the greater the degree of belonging to the set, and vice versa [54, 55]. The fuzzy system is based on fuzzy set theory and fuzzy logic reasoning. It attempts to simulate the human brain's ability to represent and solve inexact knowledge from a higher level. In the fuzzy system, knowledge is stored in the form of rules. It uses a set of fuzzy IF THEN rules to describe the characteristics of the object, and solves the uncertainty problem through fuzzy logic reasoning. The fuzzy system is good at describing the knowledge in the subject area and has strong reasoning ability. In the past 10 years, fuzzy systems have been widely used in expert systems, intelligent control, fault diagnosis and other fields, and have achieved some exciting results. However, it needs further research on the automatic extraction of fuzzy rules and the automatic generation of membership functions [10].

Chapter 3

Novel Evolutionary Computation Proposal

3.1 Introduction

Classification tasks are almost ubiquitous from human decision making to real-life problem solving. Classification means to allocate an unknown class target to a predefined class based on collected features (or attributes) of the target. For some professional fields with specificity (e.g., medical treatment or credit risk evaluation), it is crucial for a classification algorithm to possess both accuracy and understandability. Although neural networks (NNs) [56–60] and support vector machines (SVMs) [61] have achieved high classification accuracy, they are rather incomprehensible and opaque to humans since the resultant classifiers are described by complex mathematical functions rather than simple rules [62].

To address the pellucidity problem in NNs and SVMs, Parpinelli [63] for the first time proposed an ant colony optimization [64–66] based data miner (Ant-Miner), possessing both deducibility and understandability. By discovering a list of IF-THEN rules in the form of "IF $< term_1 >$ AND $< term_2 >$ AND ... $< term_n >$ THEN $< class >$", the classification results obtained by Ant-Miner can be represented via natural languages and therefore have satisfying comprehensibility for human decision making. Ant-Miner was further developed in [62, 67–69]. Martens [62] modified Ant-Miner by using a better performing and well defined Max-Min ant system, together with an automatic parameter controlling technique. Their proposed classifier called AM+ showed superior accuracy than its previous Ant-

Miner versions. By combining Ant-Miner with a decision tree induction algorithm, an Ant-Tree-Miner (ATM) [67] was proposed where decision trees (instead of IF-THEN rules) are generated to further enhance the practicability of the classification results. Otero [68] proposed the $\mathrm{cAM}_{\mathrm{pb}}$ by introducing a new sequential covering strategy in ACO [11–14] to alleviate the rule interaction problem during the rules construction process. Most recently, two new versions ($\mathrm{AM}_{\mathrm{mbc}}$ and $\mathrm{AM2}_{\mathrm{mbc}}$) [69] were proposed by adopting multiple rule sets to implement the rule discovery task in ACO. Their experimental results also exhibited a very promising performance.

Although several improvements have been achieved to further develop Ant-Miner, these works were devoted to modifying the inherent mechanisms in Ant-Miner and few works focus the interaction between the classification model and data. As in a data classification task, the data are originally collected for reasons (e.g. data archives are numbered for management) rather than mining the data itself, and thus redundant or irrelevant features usually exist in a dataset. Those redundant and irrelevant features might cause significant estimation errors due to finite sample size effects [70], over-fitting problem [71, 72], heavy computational burden [73, 74], etc [75–78]. Therefore, it is essential and challenging to consider a feature selection process to be incorporated into the classification model.

Based on this motivation, we propose a novel mutual information based ant colony classifier, called mr^2AM+, in this paper. First, we use a maximum relevance minimum redundancy feature selection method to select the most informative and discriminative attributes in datasets. Then, based on the reduced number of attributes, AM+ is implemented to perform the classification. The performance of the proposed mr^2AM+ is investigated on eight widely used datasets. Experimental results suggest that mr^2AM+ outperforms other seven state-of-art related classification algorithms in terms of accuracy and the size of model.

3.2 The Proposed mr^2AM+ Method

The proposed mr^2AM+ is a hybrid two-step method based on the mutual information and ant colony classifier. By utilizing the mutual information (i.e. the statistical characteristics) of databases, the strongly (or weakly) relevant features are always (or possible) included in the learning process of a classifier, while irrelevant features are removed from the learning. By doing so, the reduced features can speed up the learning process, alleviate the over-fitting of training data, and improve the generalization capacity of the classification model [79]. To realize this, the maximal relevance minimal redundancy (mr^2) mutual information criterion [80] is used in this study. The mr^2 criterion can be expressed as:

$$\max \Phi(D, R), \quad \Phi = D - R, \tag{3.1}$$

where D is used to calculate the maximal relevance of a feature subset S and it is defined as the average value of all mutual information values between single features $x_i \in S$ and the target class c, shown as:

$$D = \frac{1}{|S|} \sum_{x_i \in S} I(x_i; c). \tag{3.2}$$

It is widely accepted that removing one of the two highly dependent features will not change the class-discriminative power [79]. Thus, a redundancy measure R is added into Eq. (3.1) to find a feature subset S that have the largest dependency on the target class c and simultaneously have the minimal redundancy in this selected subset. R is defined as:

$$R = \frac{1}{|S|^2} \sum_{x_i, x_j \in S} I(x_i, x_j). \tag{3.3}$$

In Eqs. (3.2) and (3.3), $I(x, y)$ is the mutual information. The probabilistic density functions for continuous and discrete cases are defined as follow, respectively.

$$I(x; y) = \int \int p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right) dxdy, \tag{3.4}$$

$$I(x; y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right). \tag{3.5}$$

Then a fast incremental search algorithm [80] whose computational complexity is $O(|S| \cdot M)$ is utilized to rank the importance of all M features in the dataset by optimizing the mr^2 criterion Φ. Thereafter, the features are ranked as $V_1(i_1), V_2(i_2), ..., V_M(i_M)$, where the first attribute V_1 in the dataset is ranked i_1-th importance, and e.g, $i_1 = M$ means that this feature is the most informative and discriminative attribute, while $i_1 = 1$ suggests that this feature is the most redundant attribute and should be removed from the learning of the classifier firstly. Based on the ranked features, AM+ is implemented for the best compromise solution that maximizes the classification accuracy and minimizes the number of features simultaneously.

The high-level pseudo-code of proposed mr^{2}AM+ method is illustrated in Algorithm 1 [62,69].

```
begin
    Input: training samples;
    Apply the incremental search algorithm to obtain the importance list of all
      features;
    for $i = 1, 2..., M$ do
        Remove the $i$-th most redundant feature $V(i)$ from the attribute set;
        Rule_set = $\phi$;
        while $|Samples| >$ maximum uncovered do
            Initialize pheromone(), heuristic information, rule$_{global\_best}$;
            $t = 0$;
            while $t <$ maximum iterations do
                for $n = 1, 2, ..., ant\_size$ do
                    rule$_n$ = Create_Rule(Samples);
                    Prune(rule$_n$);
                    Evaluate(rule$_n$);
                    rule$_{iteration\_best}$ = rule$_n$;
                Update_Pheromone(rule$_{iteration\_best}$);
                Evaluate(rule$_{iteration\_best}$);
                rule$_{global\_best}$ = rule$_{iteration\_best}$;
                $t = t + 1$;
            Samples = Samples - Covered(rule$_{global\_best}$, samples);
            Rule_set= Rule_set + rule$_{global\_best}$;
    Output: optimal feature subset, discovered list of IF-THEN rules (rule$_{global\_best}$),
      and classification accuracy;
```

To verify the performance of the proposed mr^{2}AM+, eight widely used benchmark datasets are tested. They are from UCI machine learning repository. The full name of the

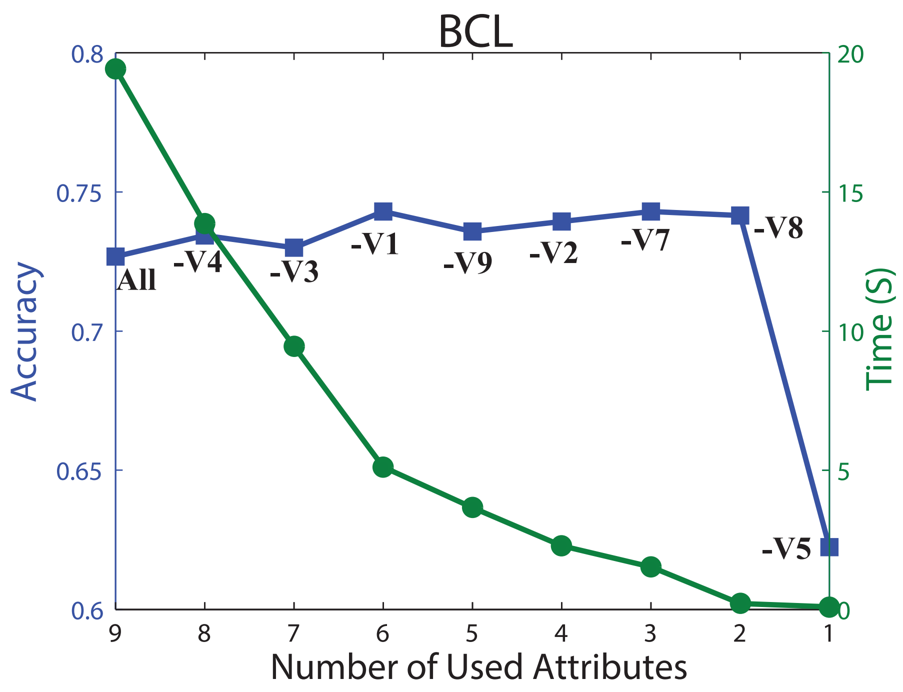


Figure 3.1: Effects of the number of used attributes on classification accuracy and computational time for the dataset BCL, respectively.

eight data sets and the corresponding abbreviations are shown in Table 3.3. Table 3.8 lists the characteristics of these datasets.

The experimental results of the proposed mr^{2}AM+ are summarized in Table 3.9, where the optimal number (#) of the feature sets, the corresponding used attributes, the average accuracy based on tenfold cross-validation of 10 independent runs, the average number of IF-THEN rules, the average number of terms in the rules, and the computational times for each classification are listed. From this table, it is clear that the optimal classification accuracy is achieved by a reduced set of features.

3.3 Experimental Results

To give some further evidences of the effects of mr^2 on the AM+ classifier, Figs. 3.1 and 3.3 illustrate the influence of the number of used attributes on classification accuracy and computational time for the dataset BCL and IO, respectively. Generally, along with the reduction of features, the computational time also becomes smaller. The redundant features

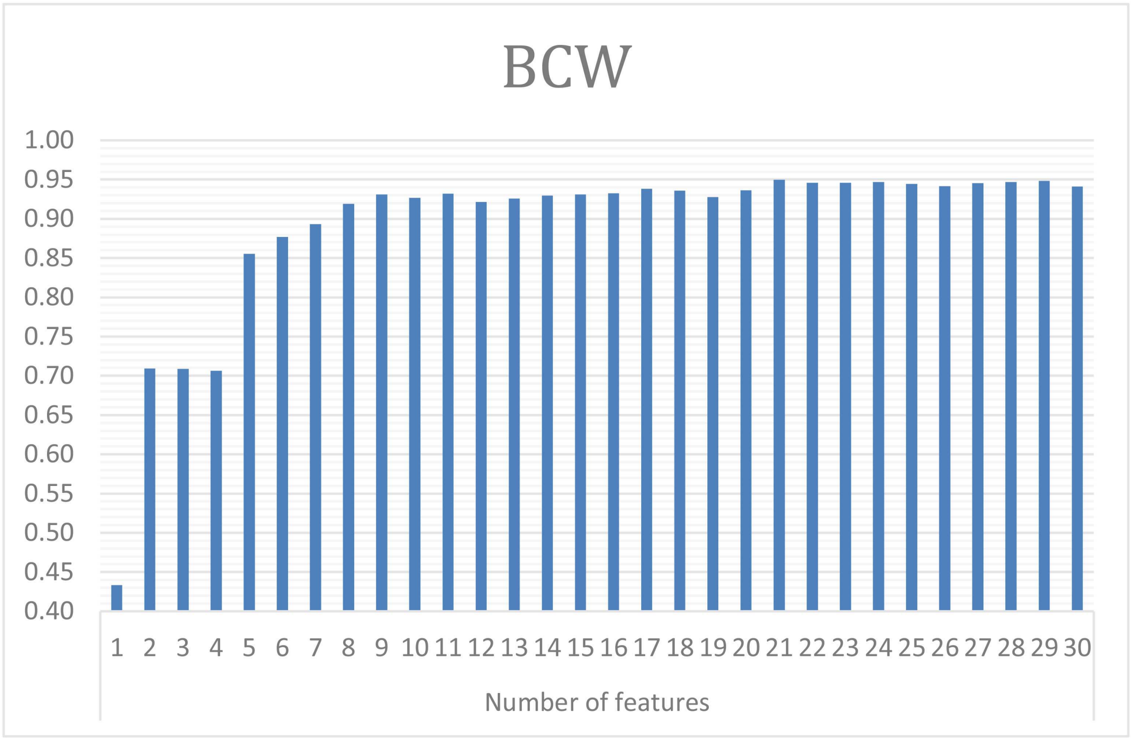


Figure 3.2: An example between the number of features and accuracy.

are sequentially removed from the feature subset, making the classification accuracy varies. However, compared with that all features are used, a reduced feature clearly attributes to a higher accuracy and meanwhile a lower computational burden. On the other hand, excessively small feature vector significantly worsens the classification accuracy. As a result, the best compromise solution that maximizes the classification accuracy and minimizes the number of features simultaneously is output as the result of mr^{2}AM+.

Furthermore, the performance of mr^{2}AM+ is compared with other seven state-of-art related classification algorithms, including cAM$_{pb}$ [68], ATM [67], C4.5 [81], JRip [82], AM$_{mbc}$ [69], AM2$_{mbc}$ [69], and AM+ [62]. Tables 3.4 and 3.5 show the comparative results of the classification accuracy and the size of the constructed classification model, respectively. It is observed that there is no single algorithm that can outperform the others on all tested datasets. However, the average ranks (A.Rank) suggest that the proposed mr^{2}AM+ averagely outperforms the others in terms of classification accuracy and the size of model.

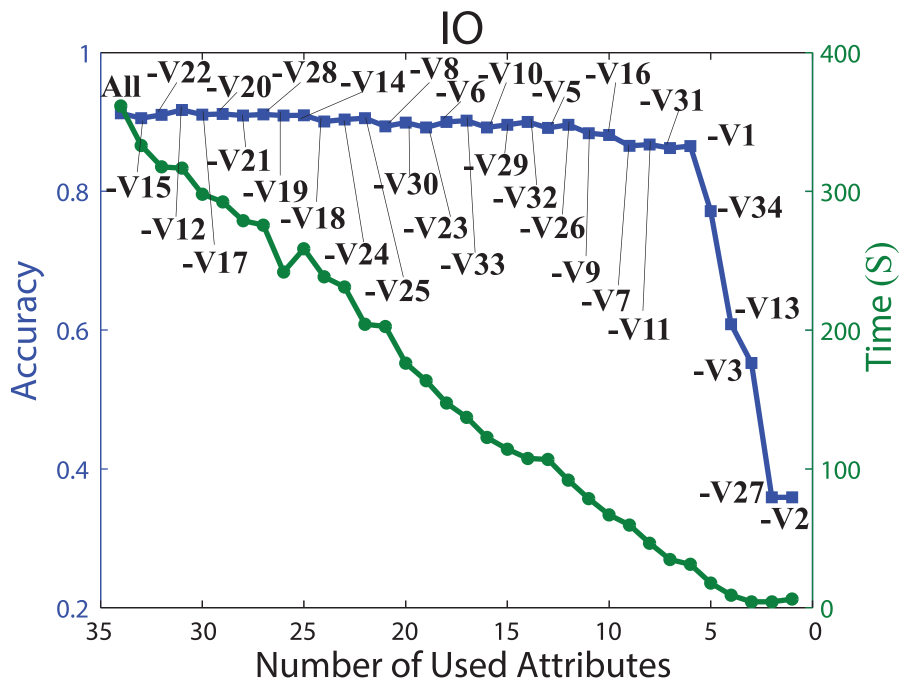


Figure 3.3: Effects of the number of used attributes on classification accuracy and computational time for the dataset IO, respectively.

3.4 Conclusions

The contributions of this study come from the following aspects: (1) an efficient mutual information based ant colony classifier (mr^{2}AM+) was proposed; (2) we empirically demonstrated that a feature reduction processing should be incorporated into an ACO classifier; (3) the use of feature selection technique to be combined with ACO-based classifiers, to the best of our knowledge, is a research area still unexplored. It thus enables this research to give more potential insights into the related community; and (4) this work also encourages people to combine mr^2 with other versions of ACO classifiers, e.g. ATM, to achieve higher classification accuracies.

Table 3.1: Overview of AntMiner versions.

	AntMiner	**AntMiner2**	**AntMiner3**	**AntMiner+**
η_{ij}	$\frac{\log_2(k) - Info(T_{ij})}{\sum_{i=1}^{n} x_i \sum_{j=1}^{p_i} (log_2(k) - Info(T_{ij}))}$	$\frac{\|T_{ij}\ \&\ CLASS = maj.class(T_{ij})\|}{\|T_{ij}\|}$	$\frac{\|T_{ij}\ \&\ CLASS = maj.class(T_{ij})\|}{\|T_{ij}\|}$	$\frac{\|T_{ij}\ \&\ CLASS = classant\|}{\|T_{ij}\|}$
$\tau_{ij}(t=0)$	$\frac{1}{\sum_{i=1}^{n} p_i}$	$\frac{1}{\sum_{i=1}^{n} p_i}$	$\frac{1}{\sum_{i=1}^{n} p_i}$	τ_{max}
τ update rule	$\frac{\tau_{ij}(t) + \tau_{ij}(t) \cdot Q}{\sum_{\forall ij \in rule} \tau_{ij}}$	$\frac{\tau_{ij}(t) + \tau_{ij}(t) \cdot Q}{\sum_{\forall ij \in rule} \tau_{ij}}$	$(1 - \rho') \cdot \tau_{ij}(t) + (1 - \frac{1}{1+Q}) \cdot \tau_{ij}(t)$	$\rho \cdot \tau_{(v_{i,j} v_{i+1,k})}(t) + \frac{Q^{+}_{best}}{10}$
$P_{ij}(t)$	$\frac{\tau_{ij}(t) \cdot \eta_{ij}}{\sum_{k=1}^{n} x_k \sum_{l=1}^{p_k} (\tau_{kl}(t) \cdot \eta_{kl})}$	$\frac{\tau_{ij}(t) \cdot \eta_{ij}}{\sum_{k=1}^{n} x_k \sum_{l=1}^{p_k} (\tau_{kl}(t) \cdot \eta_{kl})}$	$\frac{\tau_{ij}(t) \cdot \eta_{ij}}{\sum_{k=1}^{n} x_k \sum_{l=1}^{p_k} (\tau_{kl}(t) \cdot \eta_{kl})}$ if $q_1 \leq 0.4$, with $q_{1,2}$ random $\in [0, 1]$ loop if $q_2 \leq \sum_{j \in J_i} P_{ij}$ then choose term$_{ij}$ end loop else choose term with max P_{ij}	$\frac{[\tau_{(v_{i-1,k}, v_{i,j})}(t)]^{\alpha} \cdot [\eta_{v_{i,j}}(t)]^{\beta}}{\sum_{l=1}^{p_i} [\tau_{(v_{i-1,k}, v_{i,l})}(t)]^{\alpha} \cdot [\eta_{v_{i,l}}(t)]^{\beta}}$
Pruning	Yes(based on Q)	Yes(based on Q)	Yes(based on Q)	Yes(based on $confidence$)

Table 3.2: The fact used attribute and accuracy (average±standard error) in the experiments.

Data set	Used attributes	Dimensionality reduction percentage	Accuracy
BCL	3	0.6667	74.30±0.47
BCW	21	0.3000	94.99±0.20
CMC	8	0.1111	48.74±0.84
AUS	11	0.2143	85.87±0.31
GL	1	0.9000	97.80±0.17
IRIS	2	0.5000	95.07±0.25
IO	31	0.0882	91.71±0.34
TAE	3	0.4000	37.28±0.73

Table 3.3: The full name of the eight data sets and the corresponding abbreviations.

Full name of dataset	Abbreviation
Breast Cancer Data-ljubljana	BCL
Wisconsin Diagnostic Breast Cancer	BCW
Contraceptive Method Choice	CMC
Australian Credit Approval	AUS
Glass	GL
Iris Pants Database	IRIS
Ionosphere	IO
Teaching Assistant Evaluation	TAE

Table 3.4: Average classification accuracy (in%) obtained by using tenfold cross-validation and each validation is implemented by 10 independent runs for all compared algorithms.

Data set		mr^2AM+	cAM_{pb} [68]	ATM [67]	C4.5 [81]	JRip [82]	AM_{mbc} [69]	$AM2_{mbc}$ [69]	AM+ [62]
BC-L	Accuracy	74.30±0.47	72.32±0.31	73.52±0.18	72.93±2.31	69.26±2.04	74.02±0.26	73.64±0.44	72.67±0.48
	Rank	1	7	4	5	8	2	3	6
BC-W	Accuracy	94.99±0.20	94.29±0.16	94.00±0.31	94.15±0.75	93.66±1.42	94.73±0.13	94.60±0.11	94.11±0.17
	Rank	1	4	7	5	8	2	3	6
CMC	Accuracy	48.74±0.84	51.73±1.02	51.05±0.73	46.62±1.56	52.41±1.02	47.07±1.63	46.66±1.40	47.98±0.98
	Rank	4	2	3	8	1	6	7	5
AUS	Accuracy	85.87±0.31	85.68±0.15	85.90±0.10	85.80±1.08	85.80±1.10	86.37±0.83	86.01±0.55	85.43±0.19
	Rank	4	7	3	5.5	5.5	1	2	8
GL	Accuracy	97.80±0.17	95.38±1.57	97.16±0.42	96.73±0.74	95.33±1.42	96.36±0.87	96.31±0.74	96.87±0.18
	Rank	1	7	2	4	8	5	6	3
IRIS	Accuracy	95.07±0.25	93.33±1.72	96.23±0.18	93.91±1.60	96.00±1.09	88.87±1.95	87.53±1.29	94.67±0.28
	Rank	3	6	1	5	2	7	8	4
IO	Accuracy	91.71±0.34	90.59±1.66	90.81±0.29	90.64±1.19	89.74±2.68	92.42±0.39	91.37±0.61	91.25±0.44
	Rank	2	7	5	6	8	1	3	4
TAE	Accuracy	37.28±0.73	44.42±5.99	46.38±0.97	52.98±2.34	39.07±2.68	32.12±2.92	38.48±2.76	36.56±1.05
	Rank	6	3	2	1	4	8	5	7
A.Rank		**2.75**	5.38	3.38	4.94	5.56	4.00	4.63	5.38

Table 3.5: Average size of the model obtained by using tenfold cross-validation and each validation is implemented by 10 independent runs for all compared algorithms.

Data set		mr^{2}AM+	cAM$_{pb}$ [68]	ATM [67]	C4.5 [81]	JRip [82]	AM$_{mbc}$ [69]	AM2$_{mbc}$ [69]	AM+ [62]
BC-L	Size	1.70±0.19	10.36±0.11	10.04±0.23	44.51±0.33	3.27±0.11	3.07±0.10	2.86±0.10	2.73±0.24
	Rank	1	7	6	8	5	4	3	2
BC-W	Size	5.36±0.07	8.33±0.10	9.02±0.11	11.54±0.10	5.67±0.17	4.68±0.11	4.14±0.14	5.37±0.07
	Rank	3	6	7	8	5	2	1	4
CMC	Size	2.94±0.28	18.60±1.07	85.80±1.85	98.67±3.52	5.01±0.11	3.23±0.04	3.32±0.09	2.69±0.11
	Rank	2	6	7	8	5	3	4	1
AUS	Size	3.29±0.24	12.31±0.10	29.64±0.22	34.92±0.71	4.14±0.21	4.79±0.24	4.31±0.21	3.51±0.17
	Rank	1	6	7	8	3	5	4	2
GL	Size	5.00±0.00	6.00±0.08	20.18±0.12	26.16±0.48	8.68±0.48	6.01±0.37	4.98±0.25	5.01±0.01
	Rank	2	4	2	7	8	5	1	3
IRIS	Size	3.04±0.04	4.53±0.22	4.22±0.06	5.05±0.08	3.14±0.13	2.96±0.14	2.90±0.19	3.25±0.05
	Rank	3	7	6	5	8	2	1	5
IO	Size	11.55±0.15	8.00±0.29	12.48±0.15	14.35±0.17	3.57±0.20	5.65±0.28	6.45±0.34	11.46±0.11
	Rank	6	4	7	8	1	2	3	5
TAE	Size	1.03±0.02	10.20±0.25	34.77±0.22	60.34±2.68	3.06±0.11	10.97±1.01	10.17±1.44	1.03±0.02
	Rank	1.5	5	7	8	3	6	4	1.5
A.Rank		**2.44**	5.63	6.13	7.50	4.75	3.63	2.63	2.94

Table 3.6: The results of binary and multi-class classification cases.

	BCL				**CMC**				**AUS**				**IRIS**				**TAE**			
	inst		attr		inst		attr		inst		attr		inst		attr		inst		attr	
	277		9		1473		9		690		15		150		4		151		5	
Technique	Acc	σ_{Acc}	#R	#T/R	Acc	σ_{Acc}	#R	#T/R	Acc	σ_{Acc}	#R	#T/R	Acc	σ_{Acc}	#R	#T/R	Acc	σ_{Acc}	#R	#T/R
mr^{2}AntMiner+	74.30	1.49	**1.7**	1.9	48.74	2.64	2.9	3.1	**85.87**	0.97	**3.3**	2.1	**95.07**	0.78	**3.0**	1.4	37.28	2.32	**1.0**	**0.1**
AntMiner+	72.67	1.51	2.7	2.8	47.98	3.10	**2.7**	3.3	85.43	0.62	3.5	2.4	94.67	0.89	3.3	1.6	36.56	3.33	**1.0**	**0.1**
AntMiner	76.45	4.63	6.7	**1.6**	42.32	2.63	14.3	1.8	84.09	1.65	6.5	2.3	76.60	3.89	5.6	**1.0**	40.39	7.17	8.9	1.2
AntMiner2	75.91	5.72	7.0	**1.6**	41.49	2.95	15.5	**1.7**	84.30	0.78	6.1	**1.8**	81.80	3.22	5.4	**1.0**	43.73	5.55	8.3	1.3
AntMiner3	**78.39**	3.64	7.0	**1.6**	40.85	2.47	15.3	**1.7**	83.61	2.86	7.1	2.4	77.00	3.80	5.5	**1.0**	40.39	6.68	8.3	1.2
RIPPER	75.22	3.72	2.3	2.0	**48.94**	2.83	4.8	3.2	84.52	1.47	5.3	2.3	93.00	1.94	3.2	1.2	35.20	6.41	5.1	1.3
C4.5	74.56	2.36	7.3		46.60	2.15	79.4		84.82	0.83	15.8		93.80	2.20	3.3		47.20	5.67	64.8	
INN	72.18	2.47			42.16	1.28			80.83	2.17			91.00	2.16			50.20	7.86		
logit	76.56	3.95			47.52	1.80			84.83	2.51			93.80	2.90			**51.96**	6.74		
SVM	75.27	3.12			48.55	1.63			85.22	2.12			94.40	2.63			48.43	4.81		

Table 3.7: The average performances dealt with all 8 techniques.

Technique	Accuracy	Ranking	#R	#T/R
mr^{2}AntMiner+	68.25	4.0	**2.4**	1.7
AntMiner+	67.46	5.2	2.6	2.0
AntMiner	63.97	6.9	8.4	1.6
AntMiner2	65.45	6.6	8.5	**1.5**
AntMiner3	64.05	7.1	8.6	1.6
RIPPER	67.38	5.8	4.1	2.0
C4.5	69.40	5.3	34.1	
INN	67.27	7.4		
logit	**70.93**	**3.3**		
SVM	70.39	3.4		

Table 3.8: Datasets used in the experiment.

Datasets	Attributes		Classes	Samples
	Nominal	Continuous		
BCL	9	0	2	286
BCW	0	30	2	569
CMC	7	2	3	1473
AUS	8	6	2	690
GL	0	10	7	214
IRIS	0	4	3	150
IO	0	34	2	351
TAE	4	1	3	151

Table 3.9: Performance of the proposed mr^{2}AM+ for tested eight datasets.

	#	Used attributes	Accuracy	Rules	Terms	Time
BCL	3	V5, V6, V8	0.74	1.70	1.91	1.5
BCW	21	V2, V4, V5, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20, V22, V24, V25, V27, V29, V30	0.95	5.36	3.31	82.7
CMC	8	V1, V2, V4, V5, V6, V7, V8, V9	0.49	2.94	3.13	11.0
AUS	11	V1, V3, V4, V5, V6, V7, V8, V9, V10, V13, V14	0.86	3.29	2.14	14.4
GL	1	V1	0.98	5.00	1.41	11.7
IRIS	2	V2, V3	0.95	3.04	1.37	6.1
IO	31	V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V13, V14, V16, V17, V18, V19, V20, V21, V23, V24, V25, V26, V27, V28, V29, V30, V31, V32, V33, V34	0.92	11.55	1.81	316.7
TAE	3	V1, V3, V4	0.37	1.03	0.02	0.1

Chapter 4

Novel Neural Computation Proposal

4.1 Introduction

As a machine learning technique, a supervised learning algorithm is usually evaluated with a dataset which includes training samples and testing samples. Each sample is depicted by a certain number of features (or attributes) and a class label, e.g. for the medical diagnosis, the features might consist of the age, sex, and smoking habit of a patient, and the class label is the corresponding diagnosis result that the patient is whether or not suffering from liver disorders [83]. After learning, the classifier can obtain learning rules that can be applied to classify future samples in the same domain. However, most domains are explored with less than 40 features before 1997 [84]. It should not be tolerated that the dimension issue of the dataset leads the study to only explore on a limited scale. To explore the domains with more features, the optimization of the dataset is urgent and challenging. Regarding the feature of a dataset, the concept of "relevance" is firstly proposed by John, Kohavi and Pfleger [85] in the context of machine learning. That motivates Langley [86] to develop a relevant features selection method for assisting the learning of the classifier. However, selecting the most relevant feature through finding or ranking all the relevant features of the dataset is generally suboptimal for training a classifier, especially if the features include duplicate information, which is called redundant feature. Therefore, a maximum relevance minimum redundancy (Mr^2) feature selection framework that can eliminate most irrelevant and redundant features to reduce training samples is proposed for gene expression array analysis [87]. Generally, in a gene expression dataset which contains 6,000 ~ 60,000 samples, there are only less than 100 samples which are suitable for training and testing. Hence, the feature selection provides a good solution for developing the gene domain. The objective of the feature selection is to avoid the curse of dimensionality of the dataset, and thereafter to improve the classification performance of the classifiers. It can not only provide better classification accuracy with lower computation cost, but also give an easier understanding of the importance of the feature in the dataset. The feature selection methods have driven the classifier to explore more domains, particularly those consist of numerous features. It has been widely applied to areas of text processing of internet documents [88], combinatorial chemistry [84], etc.

To achieve the best performance of classification, in addition to the feature selection, the classifier is another crucial factor. Among hundreds of classifiers, the artificial neural networks (ANNs) occupy an important place. ANNs are inspired by biological systems with lots of interconnected simple processors [89, 90], and are widely applied for solving problems arisen from many different fields, e.g. business, industry and science [91]. The well-known mathematical neuron model called McCulloch-Pitts model (MCP) [92] defines the corresponding weights for the synapses to control the importance of the inputs. In recent years, many studies [93–95] suggest that the information processing capacity based MCP of a single neuron has not been fully developed. As the MCP-based single neuron model is too over-simplified to address nonlinearly separated problems [96, 97], it is considered that the utilization of the dendritic structure [98, 99] is promising to improve the nonlinear processing ability for a neuron. Although the Koch-Poggio-Torre model [100] considers the effects of dendrites in the neuron, it lacks the plasticity mechanism, that is, the synaptic type and dendritic structure cannot correctly classify some complex tasks [101]. Some studies [102–105] have pointed out that some pyramidal neurons possess the plasticity mechanism, which might provide inspirations for improving the Koch-Poggio-Torre model.

In our previous works, we mainly focus on the development of a single dendritic neuron model (DNM) via the nonlinear information processing ability of synapses [106]. DNM has been applied to medical diagnosis [107, 108], tourism prediction [109, 110] and financial time series prediction [56]. Besides its supervised learning ability, an unsupervised learnable DNM has been used for efficiently learning the two-dimensional multidirectional selectivity problem [111]. In addition, DNM trained by six population-based evolutionary learning algorithms also shows its prominent effects in classification, approximation, and prediction [112]. In DNM, the neuron plasticity mechanism is realized by synaptic pruning and dendritic pruning during learning. Meanwhile, the obtained simplified morphological of DNM can be implemented with hardware logical circuits [57].

To reduce the influence of redundancy feature on the dataset and save computation cost, in this paper we propose a hybrid model Mr^{2}DNM by combining Mr^2 with DNM. Mr^{2}DNM applies an optimal subset to train and generate learning rules, where the optimal subset is obtained by utilizing Mr^2 criteria to search and rank the features of the dataset, and DNM is used to evaluate the subset. Meanwhile, the unused samples of the optimal subset will be used as testing ones to verify the performance of Mr^{2}DNM. In the experiment, the proposed model is compared with other six classification models by classifying five real-world benchmark datasets, which includes three well-known medical diagnosis datasets (i.e. breast cancer, liver disorders, and diabetes), one radar dataset that returns from the ionosphere and one congressional voting records dataset. Results suggest that the proposed model outperforms its peers in terms of the classification accuracy, computational efficiency, convergence rate, and the quality of the area under the receiver operator characteristic (ROC) curve.

The remaining of this paper is organized as follows. Section 2 presents a brief introduction of the fundamental structures and functions of Mr^{2}DNM. Section 3 introduces the error back-propagation learning algorithm that is applied to train Mr^{2}DNM. Section 4 shows the experimental results of the model and performance analysis on five benchmark datasets. Finally, the conclusions are drawn in Section 5.

4.2 Proposed Model: Mr^{2}DNM

4.2.1 Mr^2

The proposed Mr^{2}DNM is a hybrid approach based on a feature selection technique and a neural network classifier, which are combined using a wrapper approach as shown in Fig.

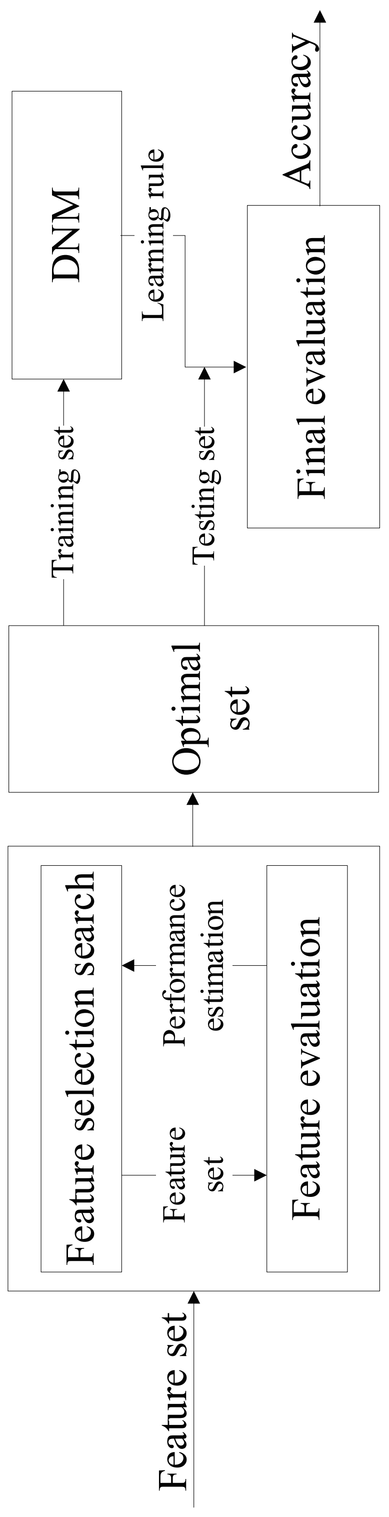


Figure 4.1: The wrapper approach to the proposed Mr^{2}DNM.

4.2.1. The feature selection is implemented via the criteria of Mr^2 based on mutual information. By calculating the mutual information of dataset, relevances of 1) feature-feature and 2) feature- target class are visually quantified. Furthermore, information overlap between features (i.e. feature-feature) is considered and defined as redundancy. The feature subset which is obtained by Mr^2 criteria includes ordered (strongly $\rightarrow$ weakly) relevance features. The relevance of the feature decides the frequency of the feature joining into the learning process of a classifier (i.e. strongly - always $\rightarrow$ weakly - possibly). Meanwhile, the irrelevant features are excluded from the optimal feature subset during the learning of the classifier. Therefore, Mr^2 feature selection combining with plasticity neurons of DNM is supposed to reduce the computational burden (e.g. learning process acceleration), avoid the over-fitting problem, and enhance the generalization capacity of Mr^{2}DNM [79,80,113]. The Mr^2 criterion based on mutual information [80] is expressed as:

$$\max \Phi(D, R), \Phi = D - R, \tag{4.1}$$

where D represents the maximal relevance of a feature set S with N features x_i. $\Phi(.)$ expresses the optimize operation which combines D and R to find an optimal feature subset. The equation of D is defined as:

$$\max\ D = \frac{1}{|S|} \sum_{x_i \in S} I(x_i; c),\ i = 1, \ldots, n, \tag{4.2}$$

where I represents the mutual information between individual feature $x_i \in S$ and the target class c. In addition, it is considered that there is redundancy in two highly dependent features. In this case, one of the two features can be removed and it will not influence the discriminative power [79]. Therefore, R is used to compute the minimal redundancy of a feature set S, shown as:

$$\min\ R = \frac{1}{|S|^2} \sum_{x_i, x_r \in S} I(x_i; x_r),\ i, r = 1, \ldots, n, \tag{4.3}$$

where $I(x; y)$ is the mutual information, both x and y are random variables. Their probabilistic density (or distribution) functions for continuous (or discrete) case is expressed as follow:

$$I(x;y) = \iint p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)dxdy, \tag{4.4}$$

$$I(x;y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right). \tag{4.5}$$

In the Mr^2 criterion, the ranking of all N features $X = x_i\{i = 1,\ldots,N\}$ in the dataset is done via selecting the features with the maximal $\Phi(.)$ in turn. Among them, the near-optimal features defined by $\Phi(.)$ can be found with an incremental search method [80]. The incremental search method is defined as follows:

$$\max_{x_r\in\{X-S_{n-1}\}}\left[I(x_r;c) - \frac{1}{n-1}\sum_{x_i\in S_{n-1}} I(x_r;x_i)\right], \tag{4.6}$$

where S_{n-1} is the feature set which there is $n-1$ features. The computational complexity of the incremental search method is $O(|S|\cdot N)$.

Additionally, the features are defined as $F_1(i_1), F_2(i_2), \ldots, F_N(i_N)$, where F_N represents the given mark of the feature in the dataset, i_N is the ranking of the feature which is obtained by the Mr^2 criterion, and e.g. $i_N = 1$ indicates that the feature F_N ranks the first one in the dataset and should be the most important feature, which has the maximal relevance with the target class c and the minimal redundancy in comparison with the other features, while $i_N = N$ means the feature F_N can be firstly excluded from the learning of the classifier to speed up the calculation efficiency. The DNM model combines with the ranked features to achieve the optimal compromised solution between classification accuracy rate and dataset dimension.

4.2.2 DNM

In DNM, the dendrites and synapses are formed via initial user-defined parameters in the primary neuron system. The initial structure is allowed to possess superfluous number of dendrites and synapses. The superfluous parts are screened, meanwhile, the useful parts are strengthened and fixed to form the ripened structure of the neuron model during learning. Four basic rules are used to define the DNM, shown as follows:

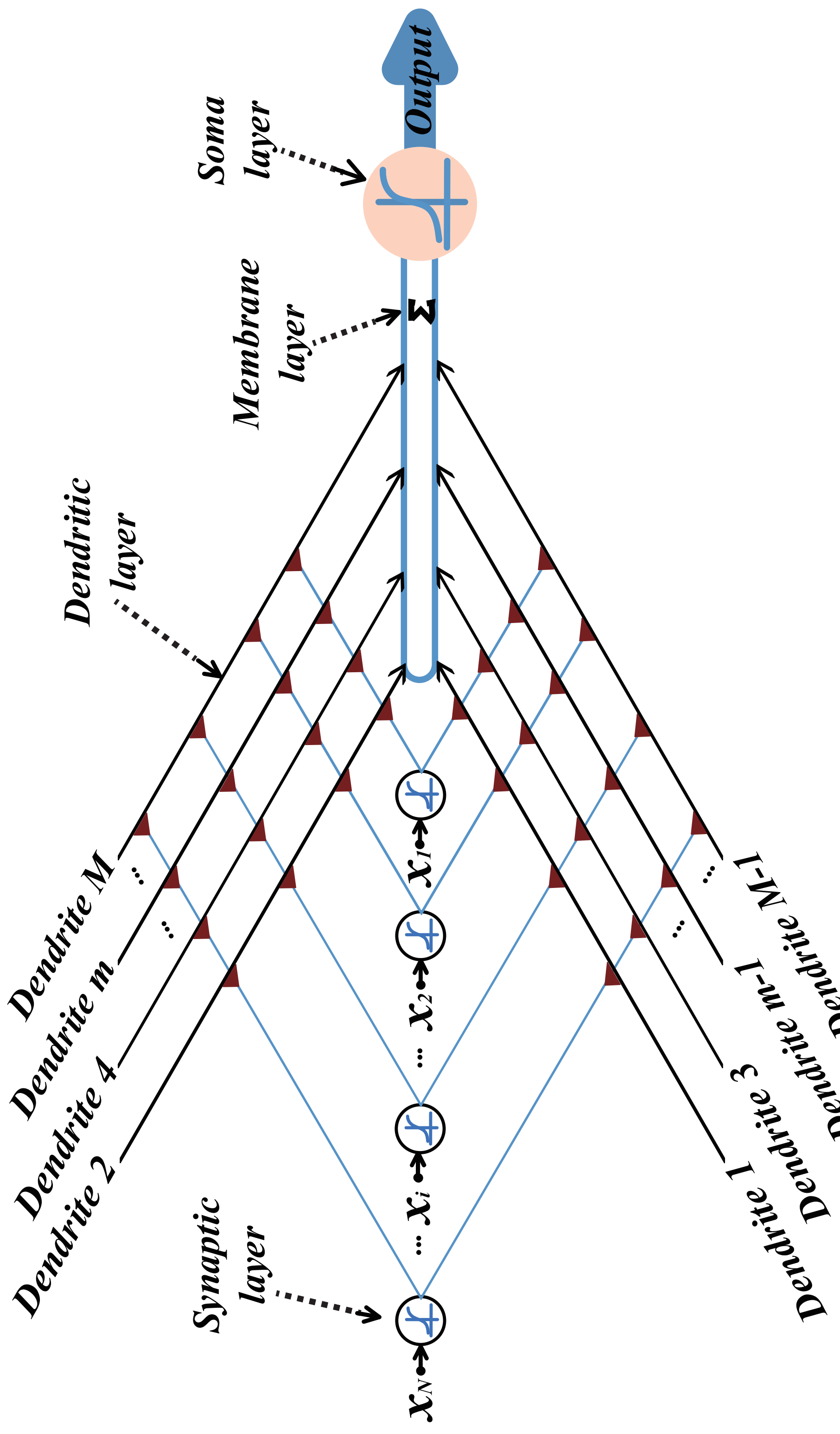


Figure 4.2: The structure of the DNM.

- The model allows initial number of dendrites and synapses which can be arbitrarily defined.
- The interaction exists among all synapses in the same dendrite layer.
- The ripened dendrites and synapses are decided by learning.
- The synapses can only be defined as one of the four specific connection states.

In Fig. 4.2.2, the transmission process of signals in the model during learning is illustrated. It can be summarized as follows:

- The input signals for one specific task are transferred to synapses via sigmoid functions and output to dendritic branches.
- The results from synapses on the same dendritic branch are calculated by applying a multiplication operation.
- The signals from all dendritic branches are collected in the membrane layer and summed to the soma layer.
- The signal is determined in the soma layer whether it exceeds the threshold or not.

4.2.2.1 Synaptic layer

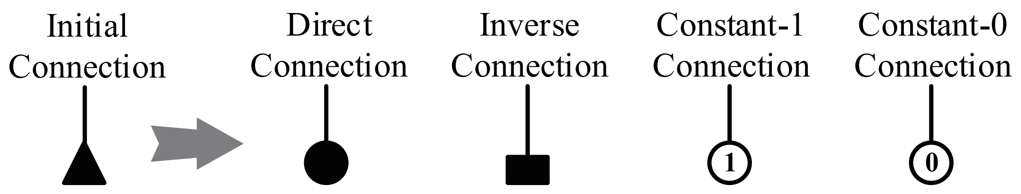


Figure 4.3: Four connection types in the synaptic layer.

A synapse is produced by the contact of two neurons. Its duty is to transmit information within two neurons. In the synaptic layer of our model, the synapse can be defined as the specific one of the four connection types, while as an input to interact with the

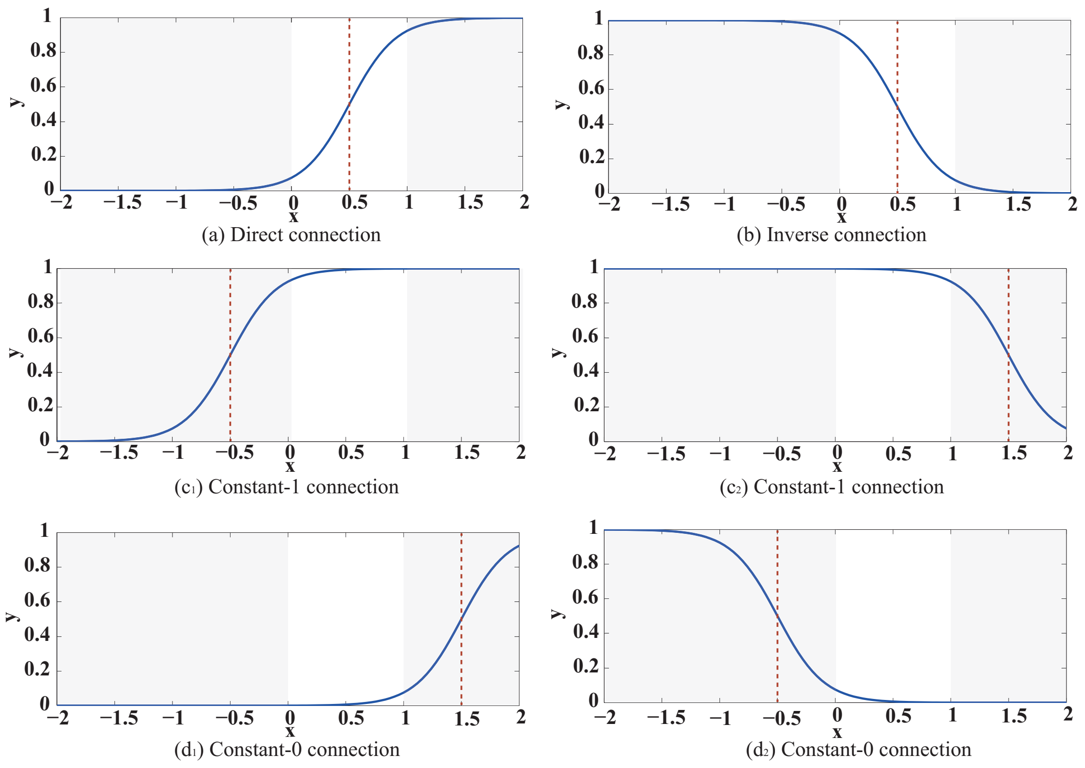


Figure 4.4: Six function cases of the synaptic layer.

dendritic branch. The four connection types include the direct connection, inverse connection, constant-0 connection, and constant-1 connection, which can be expressed by sigmoid functions. The four connection types are illustrated in Fig. 4.3. The changes in the postsynaptic potential caused by ion can be used to decide whether the input is an excitation synapse or an inhibition one [114]. The node function that connecting ith ($i = 1, 2, 3, \ldots, N$) input to the jth ($j = 1, 2, 3, \ldots, M$) synaptic layer is expressed as follows:

$$Y_{i,j} = \frac{1}{1 + e^{-k(\omega_{i,j} x_i - q_{i,j})}}, \tag{4.7}$$

where $Y_{i,j}$ indicates the output of the synaptic layer. $x_i \in [0, 1]$ denotes the input of the synapse. k represents a user-defined parameter, whose optimal setting will be given in the experiment. The weight parameters $\omega_{i,j}$ and $q_{i,j}$ in the synapses need to be trained by learning algorithms. The following equation is used to compute the threshold $\theta_{i,j}$ of the synaptic layer:

$$\theta_{i,j} = \frac{q_{i,j}}{\omega_{i,j}}. \tag{4.8}$$

The four connection types for synaptic connections are calculated and determined based on $\omega_{i,j}$ and $q_{i,j}$. Fig. 4.4 gives specific information on the four types of connections, which are divided into six cases:

- Type 1: Direct Connection

  Case (a): $0 < q_{i,j} < \omega_{i,j}$, e.g. $\omega_{i,j} = 1.0$ and $q_{i,j} = 0.5$.

  $$Y_{i,j} = \begin{cases} 1 & \text{if } x_i > \theta_{i,j} \\ 0 & \text{if } x_i \leq \theta_{i,j} \end{cases}, \tag{4.9}$$

  In this case (Fig. 4.4 (a)), when the input x_i value exceeds the threshold $\theta_{i,j}$, the output $Y_{i,j}$ is 1, which means the signals will be passed and output smoothly. Otherwise, the signals will be blocked.

- Type 2: Inverse Connection

  Case (b): $\omega_{i,j} < q_{i,j} < 0$, e.g. $\omega_{i,j} = -1.0$ and $q_{i,j} = -0.5$.

  $$Y_{i,j} = \begin{cases} 0 & \text{if } x_i > \theta_{i,j} \\ 1 & \text{if } x_i \leq \theta_{i,j} \end{cases}. \tag{4.10}$$

  where the threshold $\theta_{i,j}$ is not exceeded by the input x_i value, the output $Y_{i,j}$ is 1, which means the signal is updated as an excitatory signal and allows the information to pass, shown in Fig. 4.4 (b). The inverse connection type is considered as a logic NOT operation.

- Type 3: Constant-1 Connection

  Case (c_1): $q_{i,j} < 0 < \omega_{i,j}$, e.g. $\omega_{i,j} = 1.0$ and $q_{i,j} = -0.5$;

  Case (c_2): $q_{i,j} < \omega_{i,j} < 0$, e.g. $\omega_{i,j} = -1.0$ and $q_{i,j} = -1.5$.

  In the constant-1 connection cases (Figs. 4.4 (c_1) and (c_2)), the outputs are always 1, regardless of the inputs or the parameters change. The information will be transmitted completely.

- Type 4: Constant-0 Connection

  Case (d$_1$): $0 < \omega_{i,j} < q_{i,j}$, e.g. $\omega_{i,j} = 1.0$ and $q_{i,j} = 1.5$;

  Case (d$_2$): $\omega_{i,j} < 0 < q_{i,j}$, e.g. $\omega_{i,j} = -1.0$ and $q_{i,j} = 0.5$.

  In the two cases (Figs. 4.4 (d$_1$) and (d$_2$)), the signal is blocked and not passed, so all input values can be ignored directly.

Synaptic types are randomly assigned before the model is trained, and the weight parameters $\omega_{i,j}$ and $q_{i,j}$ are assigned a random value of -1.5 to 1.5. Once the training of the model is completed, the corresponding learning rules are generated, and the weight parameters $\omega_{i,j}$ and $q_{i,j}$ will also get the correct values to find the correct synaptic connection type.

4.2.2.2 Dendritic layer

The dendritic layer receives the signals from the synaptic layers and implements a multiplication operation. The multiplication operation approximately corresponds to a logical AND operation and is described by:

$$Z_j = \prod_{i=1}^{N} Y_{i,j}. \tag{4.11}$$

4.2.2.3 Membrane layer

The membrane layer receives the signal from the dendritic branch and adds it. This operation is most similar to the logical OR operation, and the corresponding formula is provided as follows:

$$V = \sum_{j=1}^{M} Z_j. \tag{4.12}$$

4.2.2.4 Soma layer

The soma layer is the last step of a neuronal computation and associated with a threshold. If the signal from the membrane exceeds the threshold, the transmission channel is turned on. The operation is defined as a sigmoid function and is shown as follows:

$$O = \frac{1}{1 + e^{-k_{soma}(V-\theta_{soma})}}, \tag{4.13}$$

where k_{soma} is a user-defined parameter, θ_{soma} means the threshold of the cell body and its range is [0,1]. When the signal from the membrane layer is greater than the threshold, the neuron excitation will occur, otherwise keep fired.

4.2.2.5 Neuronal pruning function

The neuronal pruning functions in the synaptic layer and dendritic layer complete the plasticity mechanism of the proposed model. Based on classification problems, the proposed model can give the specific pruning structure by applying the synaptic pruning and dendritic pruning.

Synaptic pruning: The constant-1 synaptic connection in the four connection types is considered as one of the origin of the plasticity of the neuron, which is called the synaptic pruning. The constant-1 completes a multiplication operation in the dendritic layer, since every synapse interacts with the other synapses in each dendritic layer. A value multiplied by the constant-1 is not changed, and it does not cause the output of the dendritic layer to change. Therefore, this constant-1 synaptic connection type can be neglected or pruned in the dendritic layer to simplify the neuron model without having any impact on the learning process of the proposed model.

Dendritic pruning: The constant-0 synaptic connection interacts with each dendritic layer, which is called dendritic pruning. Hence, whatever the output of the dendritic layer is, it multiplied by the constant-0 always equals 0. The outputs of all the dendritic layers are summed in the membrane layer, and any value adds zero is equal to itself. The corresponding dendrite with constant-0 can be removed without any impact, which can simplify the morphology and structure of the proposed model.

4.3 Learning Algorithm

Based on the structure of the proposed Mr^{2}DNM which is a feed-forward logic neural network, the error back-propagation (BP) algorithm is employed for training the model. The construction of the neuron model depends on an effective learning rule. Its learning rule is obtained by the *least squared error* between the real output vector O and the target output vector T, shown as follows:

$$E = \frac{1}{2}(T - O)^2, \tag{4.14}$$

The error is decreased by correcting the synaptic parameters $\omega_{i,j}$ and $q_{i,j}$ of the connection function during learning. The corrections of both parameters utilize the gradient descent learning algorithm. The equations are expressed as follows:

$$\Delta\omega_{i,j}(t) = -\eta\frac{\partial E}{\partial \omega_{i,j}}, \tag{4.15}$$

$$\Delta q_{i,j}(t) = -\eta\frac{\partial E}{\partial q_{i,j}}, \tag{4.16}$$

where η represents the learning rate, which is a user-defined parameter. However, a small learning rate might make the convergence speed slow. Thus, we set the corresponding suitable η for each classification problem as possible in the simulation. Then, the updating rules of $\omega_{i,j}$ and $q_{i,j}$ are computed as follows:

$$\omega_{i,j}(t+1) = \omega_{i,j}(t) + \Delta\omega_{i,j}, \tag{4.17}$$

$$q_{i,j}(t+1) = q_{i,j}(t) + \Delta q_{i,j}, \tag{4.18}$$

where t is the number of the learning iteration. In addition, the partial differentials of E with regard to $\omega_{i,j}$ and $q_{i,j}$ are defined as follows:

$$\frac{\partial E}{\partial \omega_{i,j}} = \frac{\partial E}{\partial O}\cdot\frac{\partial O}{\partial V}\cdot\frac{\partial V}{\partial Z_j}\cdot\frac{\partial Z_j}{\partial Y_{i,j}}\cdot\frac{\partial Y_{i,j}}{\partial \omega_{i,j}}, \tag{4.19}$$

$$\frac{\partial E}{\partial q_{i,j}} = \frac{\partial E}{\partial O}\cdot\frac{\partial O}{\partial V}\cdot\frac{\partial V}{\partial Z_j}\cdot\frac{\partial Z_j}{\partial Y_{i,j}}\cdot\frac{\partial Y_{i,j}}{\partial q_{i,j}}, \tag{4.20}$$

$$\frac{\partial E}{\partial O} = O - T, \tag{4.21}$$

$$\frac{\partial O}{\partial V} = \frac{k_{soma} e^{-k_{soma}(V - q_{soma})}}{(1 + e^{-k_{soma}(V - q_{soma})})^2}, \tag{4.22}$$

$$\frac{\partial V}{\partial Z_j} = 1, \tag{4.23}$$

$$\frac{\partial Z_j}{\partial Y_{i,j}} = \prod_{L=1 \& L \neq i}^{N} Y_{L,j}, \tag{4.24}$$

$$\frac{\partial Y_{i,j}}{\partial w_{i,j}} = \frac{k x_i e^{-k(x_i w_{i,j} - q_{i,j})}}{(1 + e^{-k(x_i w_{i,j} - q_{i,j})})^2}, \tag{4.25}$$

$$\frac{\partial Y_{i,j}}{\partial q_{i,j}} = \frac{-k e^{-k(x_i w_{i,j} - q_{i,j})}}{(1 + e^{-k(x_i w_{i,j} - q_{i,j})})^2}. \tag{4.26}$$

4.4 Experiment and Analysis

4.4.1 Experimental setup

This experiment is programmed in MATLAB (R2013b) and implemented on a computer with *Interl(R) Core i5 3.4GHz* and *RAM 16GB*. To assess the performance of the proposed Mr^{2}DNM, five widely used benchmark datasets taken from the University of California at Irvine Machine Learning Repository (UCI) are tested [115]. These datasets include Wisconsin breast cancer database (WBCD), BUPA medical research database for liver disorders (BUPA), ionosphere dataset (IONO), Pima Indians diabetes dataset (PIMA), and congressional voting records dataset (VOTE). These five datasets could be divided into categorical (WBCD, BUPA) or numerical (IONO, PIMA, VOTE) ones. Table 4.1 lists the characteristics of these datasets. To make a fair comparison, the samples which include missing value are deleted, because the used classifiers cannot handle missing value. According to our previous work, the samples of each dataset are randomly divided: 70% for training and 30% for testing [108]. In addition, the input variables are normalized from 0 to 1.0, by a min-max normalization rule:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}. \tag{4.27}$$

Table 4.1: Datasets used in the experiment.

Dataset	Feature		Sample
	Nominal	Continuous	
WBCD	9	0	683
BUPA	0	6	345
IONO	0	34	351
PIMA	0	8	768
VOTE	16	0	232

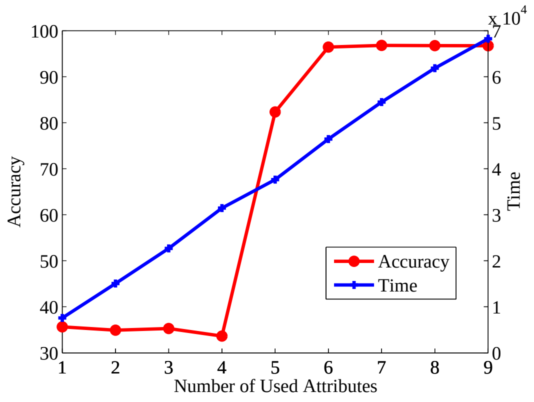
Table 4.2 provides the user-defined parameter settings to our experiment for each dataset independently. Among them, the parameter settings of five datasets are set based on the suggesting in [107, 108].

Table 4.2: Parameter setting for five datasets.

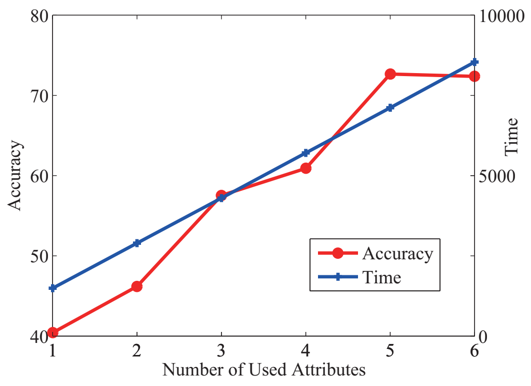
Dataset	k	k_{soma}	θ_{soma}	M	η	No. of iteration	No. of samples	
							Training	Testing
WBCD	1	10	0.5	45	0.01	1000	478	205
BUPA	3	10	0.5	10	0.005	2000	242	103
IONO	3	10	0.5	34	0.001	1000	246	105
PIMA	3	10	0.5	25	0.001	1000	538	230
VOTE	3	10	0.5	30	0.001	1000	162	70

4.4.2 Performance evaluation

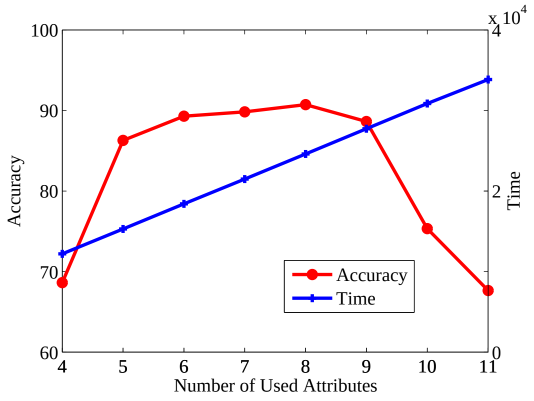
The optimal classification accuracy results of the proposed Mr^{2}DNM which adopts the reduced feature subsets are summarized in Table 4.3, where the number of features (NF) in the original dataset, the number of features in the optimal subset (#) obtained by Mr^2 criteria, the reduction rate of features of the optimal subset to the original one, corresponding feature sequence obtained by Mr^2 criteria, average accuracy based on 30 independent runs, computational time, and average area under the receiver operator characteristic curve (AUC) for five classification problems are listed. To further prove the effect of Mr^2 on the DNM classifier, Fig. 4.5 illustrates the influence of used feature size on accuracy and calculation time for classifying five datasets, respectively. It is observed that as the number of features decreases, the accuracy rate changes. Compared with the results that more features



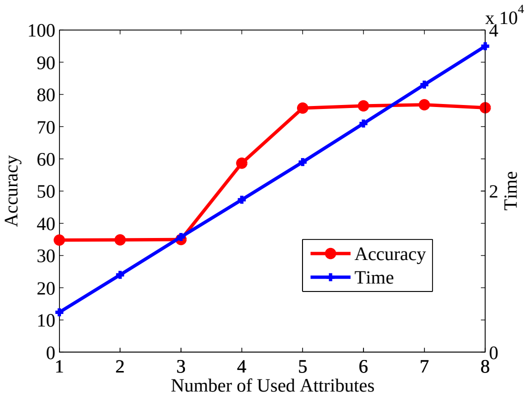
(a) WBCD



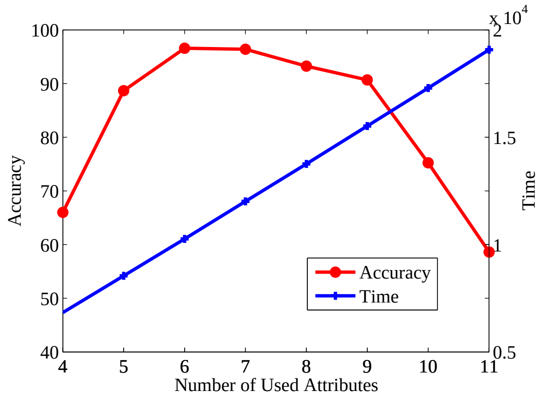
(b) BUPA



(c) IONO



(d) PIMA



(e) VOTE

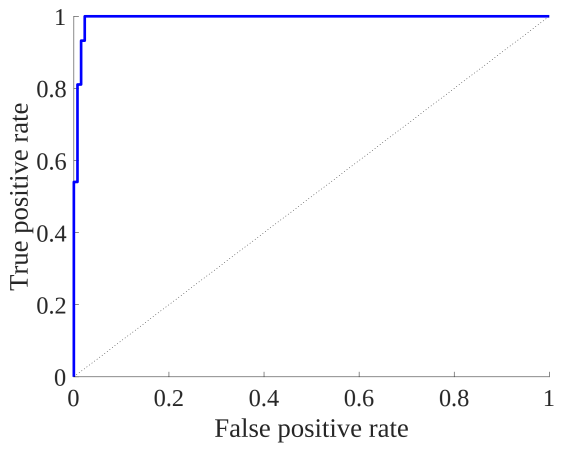
Figure 4.5: Accuracy, time and feature size for five datasets.

are used, a specific subset of features can obtain better accuracy with a lower computational cost. However, too few features will cause the accuracy rate to deteriorate significantly. In addition, the ROCs that can prove the classification quality of classifiers are shown in Fig. 4.6. AUC is the area under ROC, and its range is [0,1] [116]. It means that the classifier can perfectly classify the dataset, when the value of AUC is 1. If the AUC is equal to 0.5, it means the model is a random classifier [117]. According to Table 4.3, it can be found that Mr^{2}DNM obtains high accuracy on WBCD, IONO and VOTE, and relatively low one on BUPA and PIMA. The low accuracy is caused due to complexity of datasets, and existent literatures also obtain similar results.

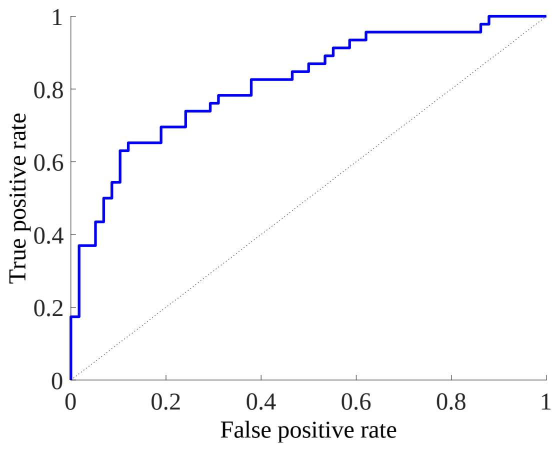
To compare the convergence speed of each feature size, the mean squared error (MSE) of Mr^{2}DNM at each iteration is calculated and illustrated in Fig. 4.7, which provides the results of 1000 iterations for five datasets. In Fig. 4.7, the number shown in the legend denotes the feature size. The curves of only eight consecutive subset sizes are shown for IONO and VOTE datasets, which contain the optimal subset size. From Fig. 4.7, it is observed that a better accuracy rate always can be obtained by removing appropriate redundancy features, and resulting in a fast convergence speed and a smooth convergence curve. Therefore, Mr^2 feature selection method is effective for DNM to deal with classification tasks.

The convergence situations of the five optimal subsets are shown in Fig. 4.8. It is clear that five datasets have all completed their own convergence within 500 iterations. Generally, the reduction of features leads to a lower calculation time. The redundant features are sequentially excluded from the feature subsets so that the classification accuracy changes. However, a reduced feature subset clearly can contribute a better accuracy with a lower calculation cost, faster and smoother convergence situation in comparison with that all features are used. It should be noted that overly small feature size conspicuously reduces the classification accuracy. For the above reasons, Mr^{2}DNM is verified to be an optimal compromised method that maximizes the classification accuracy, and synchronously minimizes the feature size and calculation time.

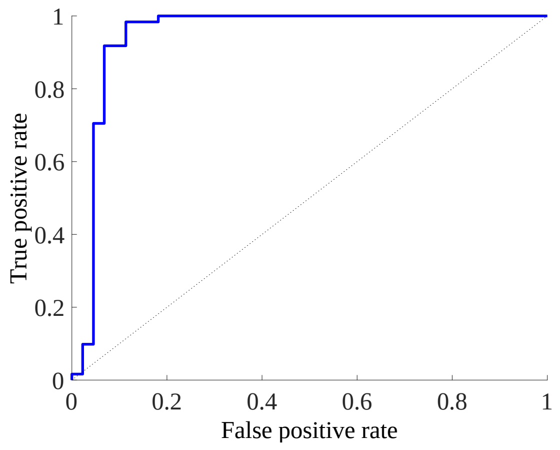
Furthermore, the performance of Mr^{2}DNM is compared with other six related classification algorithms, including standard back-propagation (Orig) [118], RENN [119], FaLKNR [120], AdaBoost [121], MultiBoost [122] and IE$_{MLP}$ [118]. Table 4.4 shows the comparative results of the classification accuracy on five benchmark datasets, and the corresponding ranks of performance are listed. The proposed Mr^{2}DNM obtains the best accuracy on three classification problems and the average rank (A.Rank) for five classification problems, which is first place among all compared methods. In fact, it can be considered that



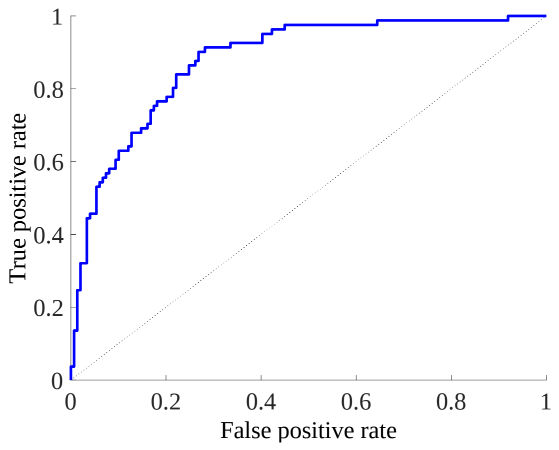
(a) WBCD



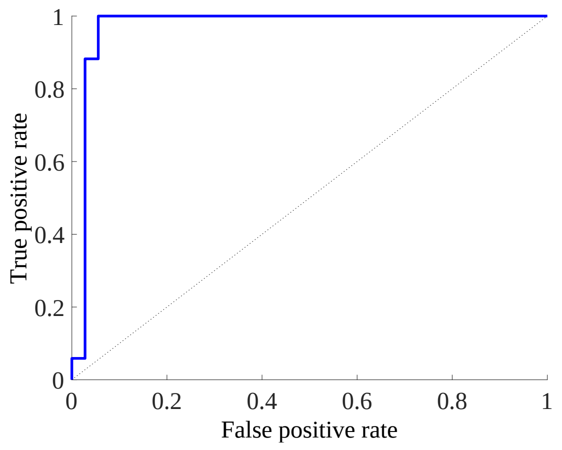
(b) BUPA



(c) IONO

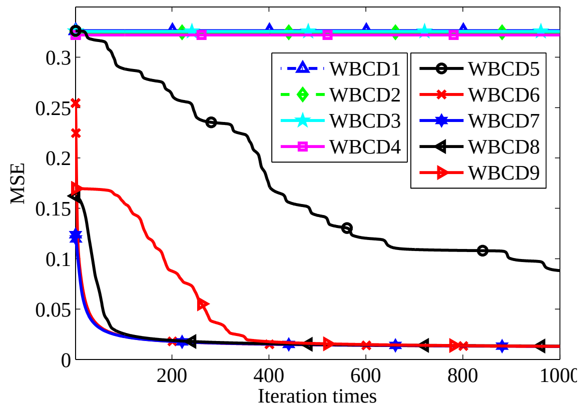


(d) PIMA

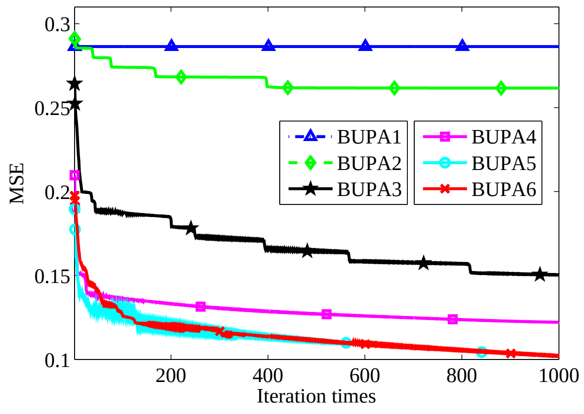


(e) VOTE

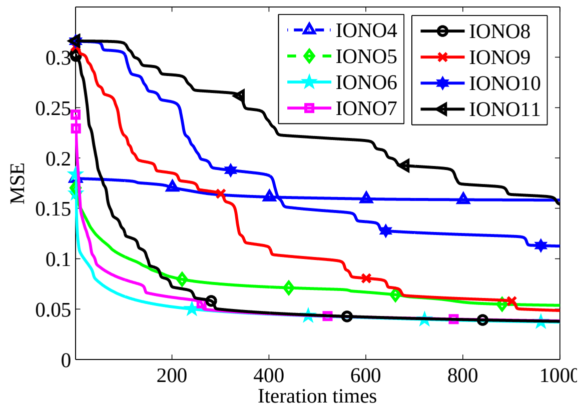
Figure 4.6: ROCs of Mr^{2}DNM that used the optimal feature subsets for five datasets.



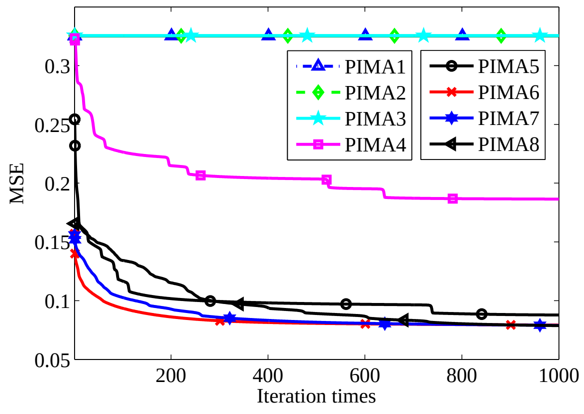
(a) WBCD



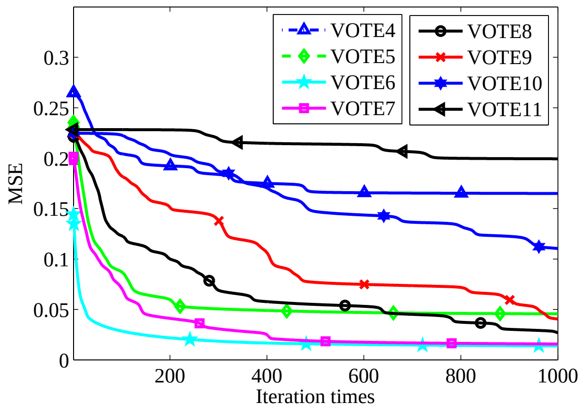
(b) BUPA



(c) IONO



(d) PIMA



(e) VOTE

Figure 4.7: MSE of each feature size for five datasets.

there is no one algorithm that always outperforms the others on all classification tasks. However, the A.Rank suggests that the performance of the proposed Mr^{2}DNM averagely outperforms the other classification techniques.

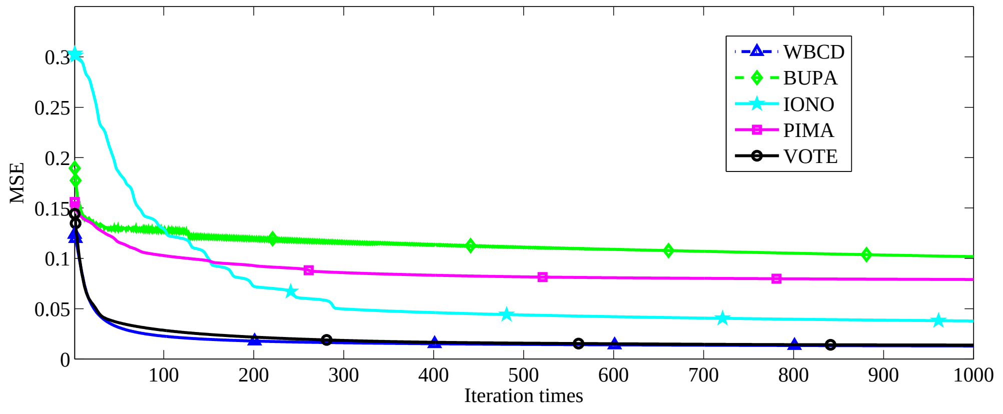
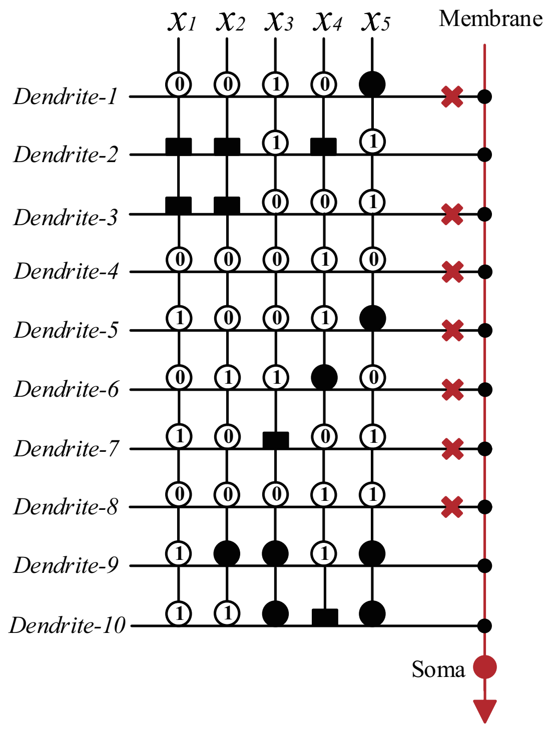


Figure 4.8: MSE of the used optimal feature sizes for five datasets.

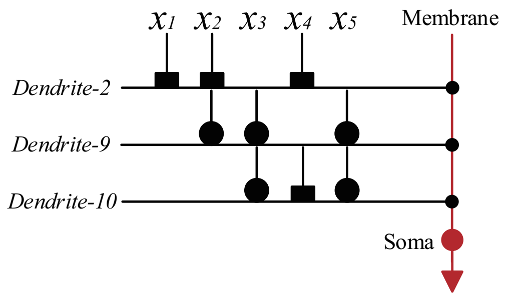
4.4.3 Simplified morphology analysis

4.4.3.1 Neuron morphology

As mentioned above, Mr^{2}DNM achieves the internal dimensional reduction via simplifying the morphology to the neuron model during learning. During learning, 1) each attribute has an input (synapse) connection on each dendritic branch; 2) an input connection is defined as one of four connection states whenever a connection action occurs; 3) four connection states are a direct connection (●), an inverse connection (▬), a constant-0 connection (⓪), and a constant-1 connection (①), respectively; 4) the same feature does not necessarily have the same connection type on each dendritic branch; and 5) all the dendritic branches are finally summed in the membrane layer. The internal dimensional reduction is implemented via ignoring the inputs (synapses) which have the constant-1 connection and removing the dendritic branches which have the input of the constant-0 connection states. An example that the neuronal morphology of BUPA is given in Fig. 4.9. Since Mr^2 is employed as the feature selection, the initial number of the feature is set as 5 at the beginning, which means that DNM reduces the calculation of 10 connection states before training the model. In addition, before training the model, there are 50 synaptic points and 10 dendritic points to perform calculation, as shown in Fig. 4.9 (a). After training, the model obtains a simplified morphology which only has 9 synaptic points and 3 dendritic points through the neuron pruning, as shown in Fig. 4.9 (b).



(a) After learning.



(b) After pruning.

Figure 4.9: The dendritic morphology of BUPA dataset.

4.4.3.2 Logic circuits morphology

The functions of Mr^{2}DNM approximately correspond to the "comparator", logical "NOT", "AND", and "OR" operation, respectively [57, 123]. Thence, the simplified neuron morphology can be replaced by the logic circuits. And the corresponding logic circuits for the five datasets are shown in Fig. 4.10(a)(b)(c)(d)(e). The comparator of the logic circuit compares the input with the corresponding threshold. If the value of the input exceeds the threshold θ, the result outputs 1; otherwise is 0. The final output of the model can be obtained by subsequent logic circuits. The implementation of the simplified model can be realized by the logic circuit in hardware so that the results are easily reproduced while decreasing the computational cost.

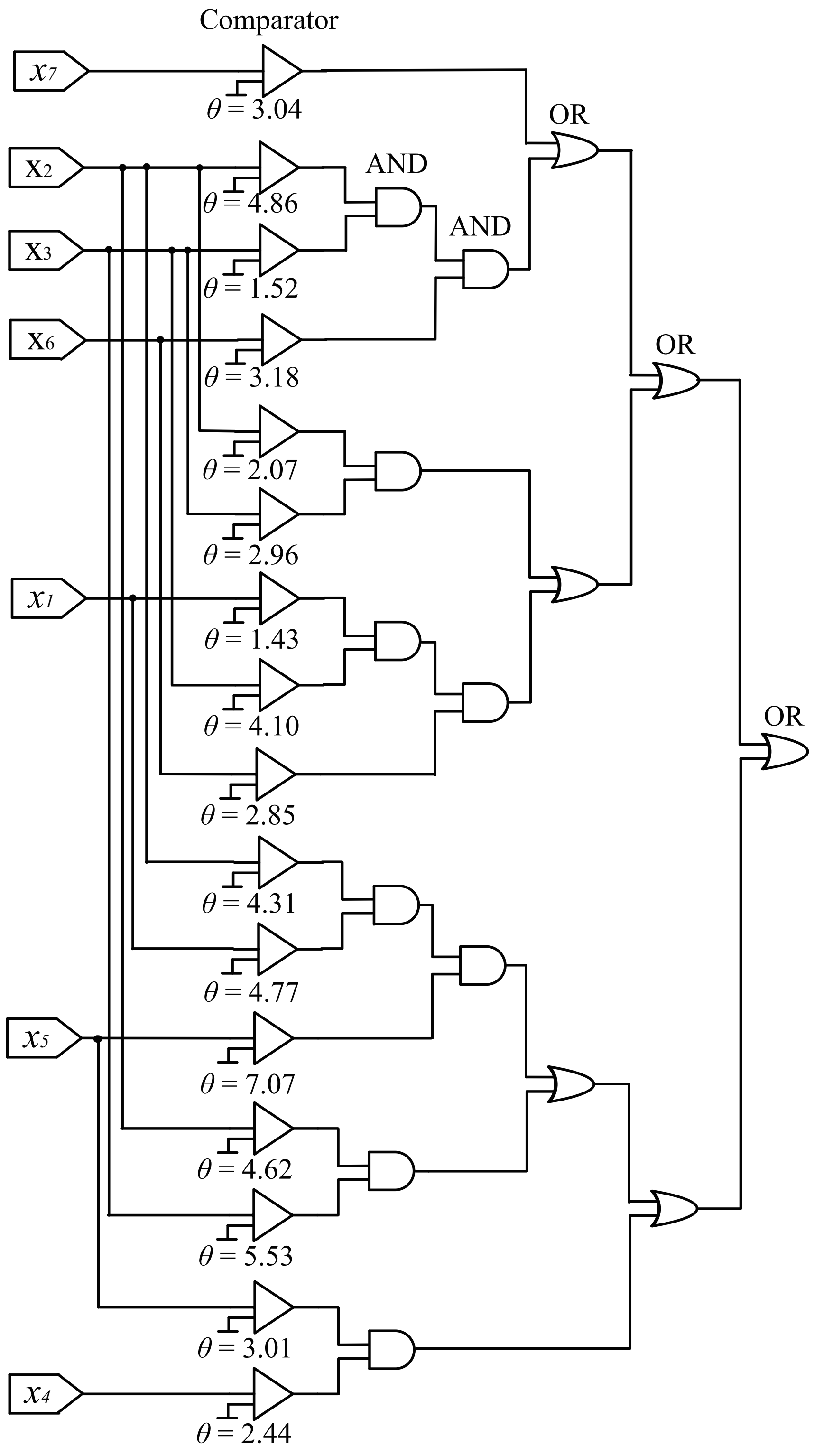


Figure 4.10: (a) Logic circuit of the simplified morphology of WBCD dataset.

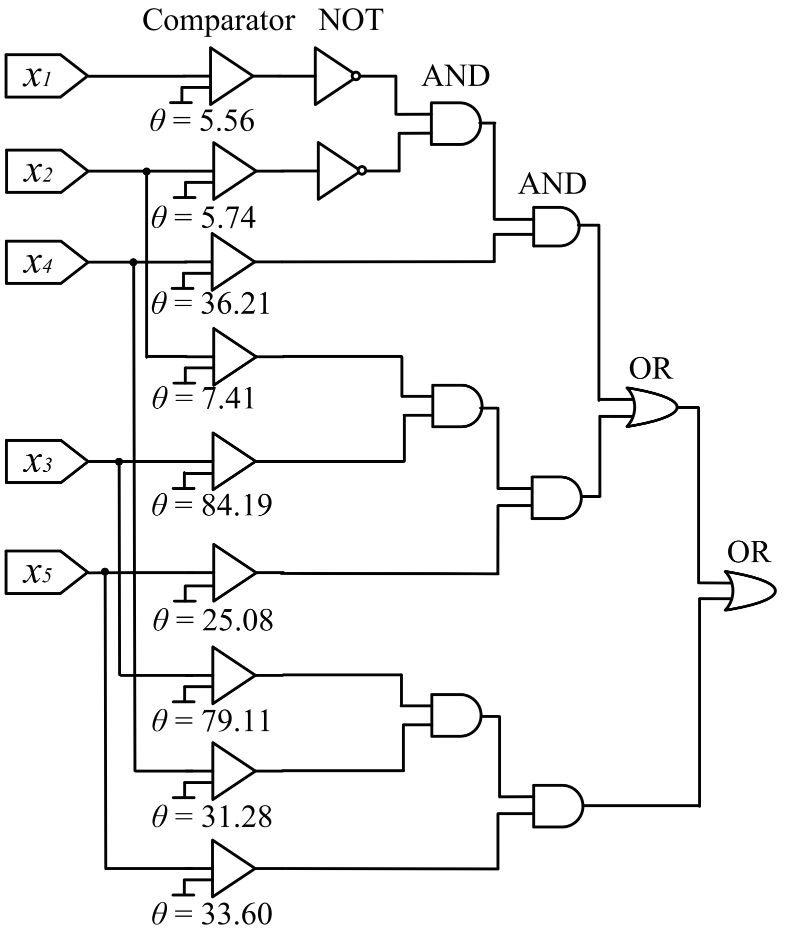


Figure 4.10: (b) Logic circuit of the simplified morphology of BUPA dataset.

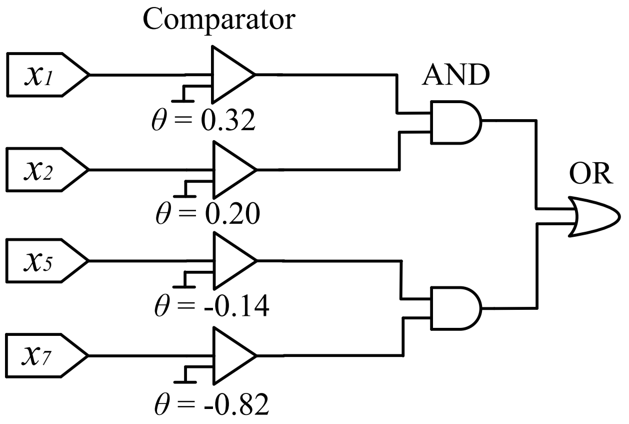


Figure 4.10: (c)Logic circuit of the simplified morphology of IONO dataset.

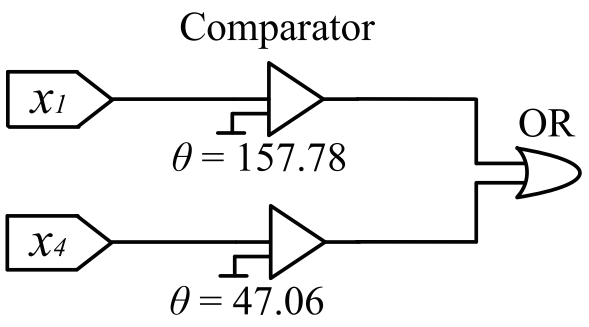


Figure 4.10: (d)Logic circuit of the simplified morphology of PIMA dataset.

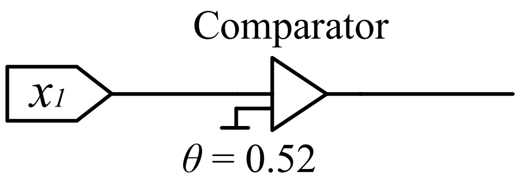


Figure 4.10: (e)Logic circuit of the simplified morphology of VOTE dataset.

4.5 Conclusion

In this paper, a hybrid model (Mr^{2}DNM) by considering the feature redundancy and non-linear interactions in a dendrite tree is used for classifying the practical problems with a low computational cost. The mutual information-based Mr^2 criterion can cut out redundant features to provide an optimal feature subset for the training of DNM. DNM trained by BP learning algorithm handles major classification work with the plastic mechanism and sigmoid functions. In addition, the simplified morphology of the proposed model obtained by training can be achieved via logic circuits to further decrease cost.

The contribution of study is summarized as follow: (1) an efficient hybrid classification model (Mr^{2}DNM) is proposed; (2) the simulation proves that a feature selection method combined with a neuron model can obtain beneficial results; (3) to our knowledge, the hybrid of feature selection method and single neuron model is a research area that still needs to be explored deeply and provides an inspiring view; and (4) meanwhile, this study advocates others to employ feature selection method to other neural network models for reaching superior classification performance, and it can be expected that such hybridization can avoid the negative impact brought by the redundancy features in the datasets and make the performance of the model fully reflected.

Table 4.3: Performance of the proposed Mr^{2}DNM for five datasets.

Dataset	NF	#	Reduction(%)	Optimal feature sequence	Accuracy (%)	Time ($\times 10^3 s$)	AUC
WBCD	9	7	22.22	F2, F6, F1, F7, F5, F3, F8	96.80	54.4	0.9942
BUPA	6	5	16.67	F5, F6, F1, F4, F3	72.66	7.1	0.7458
IONO	34	8	76.47	F5, F1, F8, F4, F3, F28, F7, F14	90.73	24.6	0.9227
PIMA	8	7	12.5	F2, F5, F8, F6, F4, F1, F3	76.80	33.2	0.8198
VOTE	16	6	62.5	F4, F5, F12, F3, F14, F8	96.57	10.2	0.9779

Table 4.4: Average classification accuracy (%) obtained by 30 runs for all compared classifiers.

Dataset		Orig	RENN	FaLKNR	AdaBoost	MultiBoost	IE$_{MLP}$	Mr^{2}DNM
WBCD	Accuracy (%)	95.28	96.14	96.28	94.99	95.85	96.62	**96.80**
	Rank	6	4	3	7	5	2	1
BUPA	Accuracy (%)	71.59	71.88	71.01	71.88	71.59	71.59	**72.66**
	Rank	5	2.5	7	2.5	5	5	1
IONO	Accuracy (%)	91.17	86.61	86.61	91.17	**91.74**	89.23	90.73
	Rank	2.5	6.5	6.5	2.5	1	5	4
PIMA	Accuracy (%)	75.39	76.69	75.91	75.26	75.13	**78.07**	76.80
	Rank	5	3	4	6	7	1	2
VOTE	Accuracy (%)	94.71	94.71	96.55	94.48	94.48	95.95	**96.57**
	Rank	4.5	4.5	2	6.5	6.5	3	1
	A.Rank	4.6	4.1	4.5	4.9	4.9	3.2	**1.8**

Chapter 5

Conclusion

Computational Intelligence (CI) is a new term that encompasses three areas: neural networks, fuzzy systems, and evolutionary computational research that have had widespread and far-reaching implications. But computational intelligence can make the three merge into an organic whole, so that the advantages can be complemented, so that the integrated system will be more effective than a single technology, and can achieve greater results. Emerging computing intelligence expands the traditional computing model and intelligent theory. It is suitable for complex systems that cannot be accurately described by mathematical models. It has been widely used in the field of mechanical engineering and has achieved certain effects. However, computational intelligence still has many shortcomings in theory and practical applications (such as learning problems of neural networks), which restricts the practicality of computational intelligence, and its improvement has yet to be further efforts by researchers.

Bibliography

[1] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: optimization by a colony of cooperating agents,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 26, no. 1, pp. 29–41, 1996.

[2] T. Stützle and H. Hoos, “Max-min ant system and local search for the traveling salesman problem,” in *Evolutionary Computation, 1997., IEEE International Conference on*. IEEE, 1997, pp. 309–314.

[3] K. Mäkisara, O. Simula, J. Kangas, and T. Kohonen, *Artificial neural networks*. Elsevier, 2014, vol. 2.

[4] L. A. Zadeh, “Fuzzy sets,” *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[5] T. Bäck, U. Hammel, and H.-P. Schwefel, “Evolutionary computation: Comments on the history and current state,” *Evolutionary computation, IEEE Transactions on*, vol. 1, no. 1, pp. 3–17, 1997.

[6] J. C. Bezdek, “What is computational intelligence?” USDOE Pittsburgh Energy Technology Center, PA (United States); Oregon State . . . , Tech. Rep., 1994.

[7] R. I. MARKS, “Intelligence: Computational versus artificial,” *IEEE Trans. Neural Networks*, vol. 4, no. 5, pp. 737–739, 1993.

[8] J. C. Bezdek, “Ieee fellows-class of 2015 [society briefs],” *Computational Intelligence Magazine, IEEE*, vol. 10, no. 2, pp. 7–17, 2015.

[9] W. Pedrycz, A. Sillitti, and G. Succi, “Computational intelligence: an introduction,” in *Computational Intelligence and Quantitative Software Engineering*. Springer, 2016, pp. 13–31.

[10] A. P. Engelbrecht, *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[11] M. Dorigo and C. Blum, “Ant colony optimization theory: A survey,” *Theoretical computer science*, vol. 344, no. 2, pp. 243–278, 2005.

[12] C. Blum, “Ant colony optimization: Introduction and recent trends,” *Physics of Life reviews*, vol. 2, no. 4, pp. 353–373, 2005.

[13] M. Dorigo, M. Birattari, and T. Stützle, “Ant colony optimization,” *Computational Intelligence Magazine, IEEE*, vol. 1, no. 4, pp. 28–39, 2006.

[14] W. Xiang and H. Lee, “Ant colony intelligence in multi-agent dynamic manufacturing scheduling,” *Engineering Applications of Artificial Intelligence*, vol. 21, no. 1, pp. 73–85, 2008.

[15] R. C. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1. New York, NY, 1995, pp. 39–43.

[16] J. Kennedy, “Particle swarm optimization,” in *Encyclopedia of Machine Learning*. Springer, 2010, pp. 760–766.

[17] G. Beni, “From swarm intelligence to swarm robotics,” in *Swarm robotics*. Springer, 2005, pp. 1–9.

[18] J. Halloy, G. Sempo, G. Caprari, C. Rivault, M. Asadpour, F. Tâche, I. Said, V. Durier, S. Canonge, J. M. Amé *et al.*, “Social integration of robots into groups of cockroaches to control self-organized choices,” *Science*, vol. 318, no. 5853, pp. 1155–1158, 2007.

[19] R. S. Parpinelli and H. S. Lopes, "New inspirations in swarm intelligence: a survey," *International Journal of Bio-Inspired Computation*, vol. 3, no. 1, pp. 1–16, 2011.

[20] C. Darwin, *The Origins of Species by Means of Natural Selection, Or the Preservation of Favoured Races in the Struggle for Life*. Kartindo. com, 1888.

[21] H.-G. Beyer, *The theory of evolution strategies*. Springer Science & Business Media, 2013.

[22] Y. Hu, K. Liu, X. Zhang, L. Su, E. Ngai, and M. Liu, "Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review," *Applied Soft Computing*, vol. 36, pp. 534–551, 2015.

[23] W. Gong, Z. Cai, and D. Liang, "Adaptive ranking mutation operator based differential evolution for constrained optimization," *Cybernetics, IEEE Transactions on*, vol. 45, no. 4, pp. 716–727, 2015.

[24] J. H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM Journal on Computing*, vol. 2, no. 2, pp. 88–105, 1973.

[25] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 4, pp. 459–472, 2006.

[26] T. Stützle and H. H. Hoos, "Max–min ant system," *Future generation computer systems*, vol. 16, no. 8, pp. 889–914, 2000.

[27] L. J. Fogel, "Artificial intelligence through a simulation of evolution," in *Proc. of the 2nd Cybernetics Science Symp., 1965*, 1965.

[28] D. B. Fogel, L. J. Fogel, and J. W. Atmar, "Meta-evolutionary programming," in *Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems & Computers*. IEEE, 1991, pp. 540–545.

[29] J. R. Koza, *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*. Stanford University, Department of Computer Science Stanford, CA, 1990, vol. 34.

[30] G. Beni and J. Wang, "Swarm intelligence in cellular robotic systems," in *Robots and biological systems: towards a new bionics?* Springer, 1993, pp. 703–712.

[31] A. Gajardo, A. Moreira, and E. Goles, "Complexity of langton's ant," *Discrete Applied Mathematics*, vol. 117, no. 1-3, pp. 41–50, 2002.

[32] L. Schulman and P. Seiden, "Statistical mechanics of a dynamical system based on conway's game of life," *Journal of Statistical Physics*, vol. 19, no. 3, pp. 293–314, 1978.

[33] M. M. Millonas, "Swarms, phase transitions, and collective intelligence," *arXiv preprint adap-org/9306002*, 1993.

[34] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, vol. 2. IEEE, 1999, pp. 1470–1477.

[35] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*. IEEE, 1998, pp. 69–73.

[36] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 3, pp. 281–295, 2006.

[37] W. Zhang, W. Shi, and J. Zhuo, "Bdi-agent-based quantum-behaved pso for shipboard power system reconfiguration," *International Journal of Computer Applications in Technology*, vol. 55, no. 1, pp. 4–11, 2017.

[38] J. Kennedy, "Bare bones particle swarms," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*. IEEE, 2003, pp. 80–87.

[39] Z. Yi, Z. Xusheng, and L. Guoyong, "Hybrid particle swarm optimization based on neighborhood space [j]," *Journal of East China Jiaotong University*, vol. 3, 2013.

[40] K. E. Parsopoulos and M. N. Vrahatis, "Particle swarm optimization method in multiobjective problems," in *Proceedings of the 2002 ACM symposium on Applied computing*. ACM, 2002, pp. 603–607.

[41] N. K. Jerne, "Towards a network theory of the immune system," in *Annales d'immunologie*, vol. 125, no. 1-2, 1974, pp. 373–389.

[42] S. Tonegawa, "Somatic generation of antibody diversity," *Nature*, vol. 302, no. 5909, pp. 575–581, 1983.

[43] P. Matzinger, "The danger model: a renewed sense of self," *Science*, vol. 296, no. 5566, pp. 301–305, 2002.

[44] D. Dasgupta, Z. Ji, F. A. González *et al.*, "Artificial immune system (ais) research in the last five years." in *IEEE Congress on Evolutionary Computation (1)*, 2003, pp. 123–130.

[45] S. A. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," *Evolutionary computation*, vol. 8, no. 4, pp. 443–473, 2000.

[46] M. J. Shlomchik, A. Marshak-Rothstein, C. B. Wolfowicz, T. L. Rothstein, and M. G. Weigert, "The role of clonal selection and somatic mutation in autoimmunity," *Nature*, vol. 328, no. 6133, pp. 805–811, 1987.

[47] P. K. Harmer, P. D. Williams, G. H. Gunsch, and G. B. Lamont, "An artificial immune system architecture for computer security applications," *Evolutionary computation, IEEE transactions on*, vol. 6, no. 3, pp. 252–280, 2002.

[48] L. N. De Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 3, pp. 239–251, 2002.

[49] J. Timmis, M. Neal, and J. Hunt, "An artificial immune system for data analysis," *Biosystems*, vol. 55, no. 1, pp. 143–150, 2000.

[50] S. GAO, H. DAI, G. YANG, and Z. TANG, "A novel clonal selection algorithm and its application to traveling salesman problem," *IEICE Trans. on fundamentals of electronics, communications and computer science*, vol. 90, no. 10, pp. 2318–2325, 2007.

[51] Y. Yu, L. Cunhua, G. Shangce, and T. Zheng, "Quantum interference crossover-based clonal selection algorithm and its application to traveling salesman problem," *IEICE Trans. on Information and Systems*, vol. 92, no. 1, pp. 78–85, 2009.

[52] G. Shangce, T. Zheng, and J. ZHANG, "An improved clonal selection algorithm and its application to traveling salesman problems," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 90, no. 12, pp. 2930–2938, 2007.

[53] J. Timmis and M. Neal, "A resource limited artificial immune system for data analysis," *Knowledge-Based Systems*, vol. 14, no. 3, pp. 121–130, 2001.

[54] K. Tanaka and M. Sugeno, "Stability analysis and design of fuzzy control systems," *Fuzzy sets and systems*, vol. 45, no. 2, pp. 135–156, 1992.

[55] H. Li, S. Yin, Y. Pan, and H.-K. Lam, "Model reduction for interval type-2 takagi–sugeno fuzzy systems," *Automatica*, vol. 61, pp. 308–314, 2015.

[56] T. Zhou, S. Gao, J. Wang, C. Chu, Y. Todo, and Z. Tang, "Financial time series prediction using a dendritic neuron model," *Knowledge-Based Systems*, vol. 105, pp. 214–224, 2016.

[57] J. Ji, S. Gao, J. Cheng, Z. Tang, and Y. Todo, "An approximate logic neuron model with a dendritic structure," *Neurocomputing*, vol. 173, pp. 1775–1783, 2016.

[58] H. Chen, S. Li, Q. Shi, D. Shen, and S. Gao, "Multi-valued neural network trained by differential evolution for synthesizing multiple-valued functions," in *Information*

Science and Control Engineering (ICISCE), 2015 2nd International Conference on. IEEE, 2015, pp. 332–335.

[59] S. Gao, J. Zhang, X. Wang, and Z. Tang, “Multi-layer neural network learning algorithm based on random pattern search method,” *International Journal of Innovative Computing, Information and Control*, vol. 5, no. 2, pp. 489–502, 2009.

[60] C. Vairappan, H. Tamura, S. Gao, and Z. Tang, “Batch type local search-based adaptive neuro-fuzzy inference system (anfis) with self-feedbacks for time-series prediction,” *Neurocomputing*, vol. 72, no. 7, pp. 1870–1877, 2009.

[61] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent Systems and Their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[62] D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens, “Classification with ant colony optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 5, pp. 651–665, 2007.

[63] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, “Data mining with an ant colony optimization algorithm,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 321–332, 2002.

[64] S. Gao, Y. Wang, J. Cheng, Y. Inazumi, and Z. Tang, “Ant colony optimization with clustering for solving the dynamic location routing problem,” *Applied Mathematics and Computation*, vol. 285, pp. 149–173, 2016.

[65] Y. Wang, Z. Xu, J. Sun, F. Han, Y. Todo, and S. Gao, “Ant colony optimization with neighborhood search for dynamic tsp,” in *International Conference in Swarm Intelligence*. Springer, 2016, pp. 434–442.

[66] S. Gao, W. Wang, H. Dai, F. Li, and Z. Tang, “Improved clonal selection algorithm combined with ant colony optimization,” *IEICE transactions on information and systems*, vol. 91, no. 6, pp. 1813–1823, 2008.

[67] F. E. Otero, A. A. Freitas, and C. G. Johnson, “Inducing decision trees with an ant colony optimization algorithm,” *Applied Soft Computing*, vol. 12, no. 11, pp. 3615–3626, 2012.

[68] ——, “A new sequential covering strategy for inducing classification rules with ant colony algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 64–76, 2013.

[69] Z. Liang, J. Sun, Q. Lin, Z. Du, J. Chen, and Z. Ming, “A novel multiple rule sets data classification algorithm based on ant colony algorithm,” *Applied Soft Computing*, vol. 38, pp. 1000–1011, 2016.

[70] K. Fukunaga and R. R. Hayes, “Effects of sample size in classifier design,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 8, pp. 873–885, 1989.

[71] J. Wang, Y. Zhou, Y. Wang, J. Zhang, C. P. Chen, and Z. Zheng, “Multiobjective vehicle routing problems with simultaneous delivery and pickup and time windows: formulation, instances, and algorithms,” *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 582–594, 2016.

[72] S. Gao, C. Vairappan, Y. Wang, Q. Cao, and Z. Tang, “Gravitational search algorithm combined with chaos for unconstrained numerical optimization,” *Applied Mathematics and Computation*, vol. 231, pp. 48–62, 2014.

[73] J. Wang, J. Liao, Y. Zhou, and Y. Cai, “Differential evolution enhanced with multiobjective sorting-based mutation operators,” *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2792–2805, 2014.

[74] Y. Cai and J. Wang, “Differential evolution with neighborhood and direction information for numerical optimization,” *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 2202–2215, 2013.

[75] G. P. Zhang, “Neural networks for classification: a survey,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462, 2000.

[76] J. Cheng, J. Cheng, M. Zhou, F. Liu, S. Gao, and C. Liu, “Routing in internet of vehicles: A review,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2339–2352, 2015.

[77] J. Cheng, C. Liu, M. Zhou, Q. Zeng, and A. Ylä-Jääski, “Automatic composition of semantic web services based on fuzzy predicate petri nets,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 680–689, 2015.

[78] S. Gao, S. Song, J. Cheng, Y. Todo, and M. Zhou, “Incorporation of solvent effect into multi-objective evolutionary algorithm for improved protein structure prediction,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics, DOI 10.1109/TCBB.2017.2705094*, 2017.

[79] L. Yu and H. Liu, “Efficient feature selection via analysis of relevance and redundancy,” *Journal of Machine Learning Research*, vol. 5, no. Oct, pp. 1205–1224, 2004.

[80] H. Peng, F. Long, and C. Ding, “Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[81] J. R. Quinlan, “Improved use of continuous attributes in c4.5,” *Journal of Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.

[82] V. N. Vapnik, “The nature of statistical learning theory, ser. statistics for engineering and information science,” *New York: Springer*, vol. 21, pp. 1003–1008, 2000.

[83] R. Kohavi and G. H. John, “Wrappers for feature subset selection,” *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[84] I. Guyon and A. Elisseeff, “An introduction to variable and feature selection,” *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[85] G. H. John, R. Kohavi, and K. Pfleger, “Irrelevant features and the subset selection problem,” in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 121–129.

[86] P. Langley, “Selection of relevant features in machine learning,” in *Proceedings of the AAAI Fall symposium on relevance*, vol. 184, 1994, pp. 245–271.

[87] C. Ding and H. Peng, “Minimum redundancy feature selection from microarray gene expression data,” *Journal of Bioinformatics and Computational Biology*, vol. 3, no. 02, pp. 185–205, 2005.

[88] W. Shang, H. Huang, H. Zhu, Y. Lin, Y. Qu, and Z. Wang, “A novel feature selection algorithm for text categorization,” *Expert Systems with Applications*, vol. 33, no. 1, pp. 1–5, 2007.

[89] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[90] A. K. Jain, J. Mao, and K. M. Mohiuddin, “Artificial neural networks: A tutorial,” *Computer*, vol. 29, no. 3, pp. 31–44, 1996.

[91] G. Zhang, B. E. Patuwo, and M. Y. Hu, “Forecasting with artificial neural networks:: The state of the art,” *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[92] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[93] R. N. Yadav, P. K. Kalra, and J. John, “Time series prediction with single multiplicative neuron model,” *Applied Soft Computing*, vol. 7, no. 4, pp. 1157–1163, 2007.

[94] R. Legenstein and W. Maass, “Branch-specific plasticity enables self-organization of nonlinear computation in single neurons,” *Journal of Neuroscience*, vol. 31, no. 30, pp. 10 787–10 802, 2011.

[95] C. Weber and S. Wermter, “A self-organizing map of sigma–pi units,” *Neurocomputing*, vol. 70, no. 13, pp. 2552–2560, 2007.

[96] M. Minsky and S. Papert, “Perceptrons-expanded edition: An introduction to computational geometry,” 1987.

[97] R. P. Costa and P. J. Sjöström, “One cell to rule them all, and in the dendrites bind them,” *Frontiers in Synaptic Neuroscience*, vol. 3, p. 5, 2011.

[98] C. Koch, T. Poggio, and V. Torre, “Retinal ganglion cells: a functional interpretation of dendritic morphology,” *Phil. Trans. R. Soc. Lond. B*, vol. 298, no. 1090, pp. 227–263, 1982.

[99] ——, “Nonlinear interactions in a dendritic tree: localization, timing, and role in information processing,” *Proceedings of the National Academy of Sciences*, vol. 80, no. 9, pp. 2799–2802, 1983.

[100] C. Koch, “Computation and the single neuron,” *Nature*, vol. 385, no. 6613, p. 207, 1997.

[101] A. Destexhe and E. Marder, “Plasticity in single neuron and circuit computations,” *Nature*, vol. 431, no. 7010, p. 789, 2004.

[102] A. Artola, S. Bröcher, and W. Singer, “Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex,” *Nature*, vol. 347, no. 6288, p. 69, 1990.

[103] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann, “Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps,” *Science*, vol. 275, no. 5297, pp. 213–215, 1997.

[104] G. Bi and M. Poo, “Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type,” *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[105] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, “Rate, timing, and cooperativity jointly determine cortical synaptic plasticity,” *Neuron*, vol. 32, no. 6, pp. 1149–1164, 2001.

[106] Z. Tang, H. Tamura, M. Kuratu, O. Ishizuka, and K. Tanno, “A model of the neuron based on dendrite mechanisms,” *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 84, no. 8, pp. 11–24, 2001.

[107] Z. Sha, L. Hu, Y. Todo, J. Ji, S. Gao, and Z. Tang, “A breast cancer classifier using a neuron model with dendritic nonlinearity,” *IEICE Transactions on Information and Systems*, vol. 98, no. 7, pp. 1365–1376, 2015.

[108] T. Jiang, S. Gao, D. Wang, J. Ji, Y. Todo, and Z. Tang, “A neuron model with synaptic nonlinearities in a dendritic tree for liver disorders,” *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 12, no. 1, pp. 105–115, 2017.

[109] Y. Yu, Y. Wang, S. Gao, and Z. Tang, “Statistical modeling and prediction for tourism economy using dendritic neural network,” *Computational Intelligence and Neuroscience*, vol. 2017, 2017.

[110] W. Chen, J. Sun, S. Gao, J. Cheng, J. Wang, and Y. Todo, “Using a single dendritic neuron to forecast tourist arrivals to japan,” *IEICE Transactions on Information and Systems*, vol. 100, no. 1, pp. 190–202, 2017.

[111] Y. Todo, H. Tamura, K. Yamashita, and Z. Tang, “Unsupervised learnable neuron model with nonlinear interaction on dendrites,” *Neural Networks*, vol. 60, pp. 96–103, 2014.

[112] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, “Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, pp. 601–614, 2019.

[113] H. Yu, X. Qian, Y. Yu, J. Cheng, Y. Yu, and S. Gao, "A novel mutual information based ant colony classifier," in *Progress in Informatics and Computing (PIC), 2017 International Conference on*. IEEE, 2017, pp. 61–65.

[114] C. Koch, *Biophysics of computation: information processing in single neurons*. Oxford university press, 2004.

[115] K. Bache and M. Lichman, "Uci machine learning repository (http://archive. ics. uci. edu/ml), university of california, school of information and computer science," *Irvine, CA*, 2013.

[116] D. Pham, S. Dimov, and Z. Salem, "Technique for selecting examples in inductive learning," in *European symposium on intelligent techniques (ESIT 2000)*. Citeseer, 2000, pp. 119–127.

[117] S. H. S. A. Ubaidillah, R. Sallehuddin, and N. A. Ali, "Cancer detection using aritifical neural network and support vector machine: a comparative study," *Jurnal Teknologi*, vol. 65, no. 1, pp. 73–81, 2013.

[118] M. R. Smith, T. Martinez, and C. Giraud Carrier, "An instance level analysis of data complexity," *Machine Learning*, vol. 95, no. 2, pp. 225–256, 2014.

[119] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, pp. 448–452, 1976.

[120] N. Segata, E. Blanzieri, and P. Cunningham, "A scalable noise reduction technique for large case-based systems," in *International Conference on Case-Based Reasoning*. Springer, 2009, pp. 328–342.

[121] Y. Freund, R. E. Schapire *et al.*, "Experiments with a new boosting algorithm," in *Icml*, vol. 96. Citeseer, 1996, pp. 148–156.

[122] G. I. Webb, "Multiboosting: A technique for combining boosting and wagging," *Machine Learning*, vol. 40, no. 2, pp. 159–196, 2000.

[123] J. Ji, S. Song, Y. Tang, S. Gao, Z. Tang, and Y. Todo, “Approximate logic neuron model trained by states of matter search algorithm,” *Knowledge-Based Systems*, vol. 163, pp. 120–130, 2019.

Acknowledgements

This thesis was completed under the careful guidance of supervisor Prof. Tang and Associate Prof. Gao at University of Toyama. My supervisor Prof. Tang's profound theoretical foundation, rich practical experience, and abiding realistic attitude towards academic study have benefited me a lot. In addition, supervisor Prof. Tang's pioneering and innovative scholarly style, broad mind and sincerity are worthy of my lifelong learning. Associate Prof. Gao solved many difficult problems for me in the experimental stage. His eloquent teaching and eclectic ideas have helped me a lot.

Thanks to all the members of the Intelligent Information Systems Research Lab in University of Toyama for their help and support in my studies, experimental research and life.

I sincerely thank my parents for their most selfless and greatest love that they have always given me. Their understanding and support are the driving force behind my constant efforts and my source of strength.

On the occasion of the completion of this thesis, I would like to express my sincere gratitude to the mentor and friends who have been caring for me for three years.