University of Alberta

Modeling of Data in the Framework of Fuzzy Sets

by

Kuwen Li ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

Department of Electrical and Computer Engineering

Edmonton, Alberta

Fall 2005

0-494-09219-X

Your file Votre référence
ISBN:
Our file Notre référence
ISBN:

Abstract

Nowadays, we are continuously faced with a vast amount of information, and how to present the data in a meaningful way becomes of paramount interest. The main goal of data modeling is to offer solutions to such problems. In this thesis, we carried out research concerning three fundamental logically inclined modeling structures to improve the transparency and interpretability of data, and the ensuing models. The three modeling structures are: Reed-Muller Binary Decision Trees (RMBDT), Cascade OR/AND Fuzzy Neural Networks (COAFNN), and Parallel OR/AND Fuzzy Neural Networks (POAFNN).

Combining the optimal solution-searching techniques of both fuzzy logic and the above modeling structures, the three modeling approaches presented are demonstrated to be efficient in extracting and representing knowledge from datasets.

Acknowledgements

I would like to express my deep gratitude to my supervisors, Dr. Witold Pedrycz and Dr. Marek Reformat, for their precious advice, encouragement, continuous support and guidance throughout my study.

Thanks to my wife, Ying, for giving me all the support that I needed in my study and to my son, Tuotuo, who has missed lots of fun during this period.

Table of Content

List of Tables

List of Figures

Part I Introduction

Chapter 1 Motivation and objectives

1.1 Motivation

We live in an era of information explosion; the amount of information available is expanding day by day. How do we handle all the information we encounter in everyday life? The data may come from different sources, e.g., real estate prices, software engineering measurements, personalized Internet searching, medical diagnosis, etc. Data modeling has a close relationship to our daily life.

The approaches to developing modeling systems can be classified into two categories: those which are supposed to work autonomously, and those which are intended to be tools in the hands of the user to help him to take decisions. In the former case, the performance level may be the issue; but, in the latter, other dimensions such as comprehensibility, robustness, versatility, modifiability, and coherence with previous knowledge may be fundamental in order to allow the system to be accepted for use. The concepts of white box and black box could be introduced into the modeling approach. When achieving higher accuracy in the resulting models is the main goal, we do not want to know the details about how the models work. Models could be considered as black boxes, because they are expected to give us the results when we provide the inputs to them. When comprehensibility is the goal, we are striving for greater transparency of the models. In this case, we could say we are pursuing a white box modeling system. Primarily, we need the system to work accurately and be understandable, but can we gain these two goals together within one modeling system?

The motivation of this thesis is to give an answer to this question by creating white box modeling systems. We apply new network structures in data modeling in order to improve the accuracy of the modeling approach, and to enhance the interpretability of the knowledge.

1.2 Objectives

When we began the research, the goal was to find some new modeling structures that would give acceptable or even superb performance on data modeling or knowledge extraction. In the data modeling research area, there are two fundamental goals confronting every researcher: accuracy and interpretability. There have been many different data models researched in the past. Here, two classes of data models are to be discussed. One class is composed of the so-called "black box" models, in which the system is seen as a whole and only its interactions with the external world are considered. A black box model defines the data abstraction entirely in terms of external behavior, in transitions from stimuli to responses. Another class consists of structural or "white box" models. These models are better able to handle the situation where the inside structure of the model is essential for the users to know. In this scenario, the users are concerned about not only the results, but also the interpretable presentation of the results. We build both white box and black box modeling systems, and compare the performance of them.

In this research, two modeling structures, the Reed Muller Binary Decision Tree (RMBDT) structure and the Fuzzy Neural Network (FNN) structure, are introduced. RMBDT models are viewed as black box models, while FNN models are considered to be white box models.

With the RMBDT structure, our goal is to achieve acceptable models for prediction purposes. The RMBDTs are based on Reed Muller expansion, which uses exclusive OR operators. These models are therefore anticipated to be unable to achieve easy-to-read knowledge, but able to provide acceptable performance.

Since Fuzzy Neural Network models consist of only AND and OR operators, FNN models are considered to be easy to interpret. The OR/AND neuron is used to build the FNN models. Our goal is to investigate the suitability of the cascade OR/AND neurons to build complex structures. With the interpretability of fuzzy neurons, we may extract valuable rules from the models achieved in the training process.

The Cascade OR/AND neuron-based Fuzzy Neural Networks (COAFNNs) are applied to improve the interpretability of the models. We assume that the models using COAFNNs provide equal or slightly worse performance in predication compared with black box modeling systems, while we could extract interpretable knowledge in the form of rules from these models.

By changing the layout of the OR/AND neurons-based fuzzy neural network, the Parallel OR/AND neuron-based Fuzzy Neural Network (POAFNN) structure is applied in classification problems. In this approach, we adopted the single objective optimization approach and the multi-objective optimization technique during the training. Our goal is to make sure that we could achieve better interpretability, as well as similar performance, when we combined the knowledge and applied the knowledge on the raw data. Many actual datasets do not have even class distribution; they are referred to as skewed datasets. Most of the time, we focus on getting a higher correct classification rate on the minority class(es) in the skewed datasets. Usually, we are able to achieve a high overall classification accuracy rate, even if we just predicate all the samples as the majority class in the skewed dataset; however, the fundamental objective of adopting POAFNNs is to elevate the accuracy rate on skewed classes. The POAFNN model is mainly used as a classifier system.

Chapter 2 Organization of the thesis

This thesis is arranged into 6 parts. Part 2 provides the introduction to basic concepts in data modeling, fuzzy logic, evolutionary computation, and the datasets used in the research. Chapter 3 introduces basic concepts in the data modeling area and provides the literature review on the progress in this area. In Chapter 4, an overview of fuzzy logic is presented and the method of fuzzifying the raw datasets into fuzzy sets is explained. Evolutionary computation gets a brief introduction in Chapter 5. Two main branches in evolutionary computation, Genetic Algorithm (GA) and Evolution Strategy (ES), are explained in detail. The datasets to be used in the experiments are described in Chapter 6. These datasets include the datasets from UCI Machine Learning Repository (Boston housing dataset, Abalone dataset, Machine CPU dataset, Auto-MPG dataset, IRIS dataset) and the software metric dataset.

From Part 3 to Part 5, three data models in the framework of fuzzy sets are constructed and discussed. The fuzzification of the raw data occurs before the beginning of the training. The outputs are transferred into the form of the original raw data by defuzzification process.

In Part 3, the modeling structure using the Reed-Muller Binary Decision Tree (RMBDT) is introduced and the experimental results based on this modeling structure are presented and discussed. The fundamental concepts in Reed-Muller expansion and RMBDT are shown in Chapter 7. In Chapter 8, the modeling structure used in the experiments is described. The optimization method based on the GA used for construction of RMBDT is explained. The analysis and the discussion of the experimental results are presented.

In Part 4, we explain how to construct interpretable models using Cascade OR/AND neuron based Fuzzy Neuron Networks (COAFNN). Chapter 9 gives the overview on artificial neural networks and fuzzy neural networks, and the generic model for the fuzzy logic networks is defined. Chapter 10 describes the application of the GA to the construction of models. The statistical analysis of results is included with the rules extracted from the models. Finally, the discussion on this modeling approach is presented.

In Part 5, we use the Parallel OR/AND neuron-based Fuzzy Neuron Network (POAFNN) as the classifier. This classification system is designed to improve performance on skewed datasets. Multi-objective optimization technique is explained in Chapter 11. The Pareto-based multi-objective optimization method applied in the experiments is described in detail in this chapter. The experimental results on POAFNNs are shown in Chapter 12. Comparison of the single objective optimization with the multi-objective optimization is made and discussed. The knowledge extracted from the models is also presented.

In Part 6, the overall comparison of the three proposed approaches is made, and the conclusions and future work are addressed. Chapter 13 compares the performance in accuracy and interpretability of the three modeling approaches, while Chapter 14 concludes the work presented in this thesis and suggests potential future work along this research path.

Part II Background and Datasets

In this part, we first review concepts related to data modeling. We then review concepts in fuzzy logic, including fuzzy sets, linguistic variables, membership functions, and fuzzy modeling. Next, the evolutionary computation oriented techniques are introduced. Finally, the description of used datasets is addressed.

Chapter 3 Data modeling

3.1 Basic Concepts

The problem of characterizing a complex process can be approached from several different directions. If the process is well understood, it is possible to consider the underlying fundamental principles and arrive at a system of differential equations that describe the process. This is traditional physical modeling. For processes whose underlying principles are not well understood, or are too complex to express in terms of conservation equations, etc., without excessive simplifying assumptions, empirical modeling is often the best approach. Empirical modeling is also commonly referred to as curve fitting, data fitting, or system identification. Modeling is a very general problem and there are several tasks from a wide range of domains that can be cast into modeling tasks. The result of modeling may be a set of discrete values (known as classification), or continuous values (known as predication).

The modeling problem considered here is to learn how to achieve generalization from experience, i.e. given some examples from a domain, to learn to model new instances from the same domain. In many modeling problems explicit rules do not exist, but examples can be obtained easily. We always try to infer a solution from a (limited) set of training examples. The goal is to obtain models and learning rules to enable us to learn from the examples, and predict the value of future objects. We assume that objects are described by vectors containing a set of n real valued measurements, thus an object i is represented by the feature vector $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{in})$ $x_{ij} \in \Re$ (or shorter $\mathbf{x}_i \in \Re^n$). Each object is thus represented as one point in a feature space X.

Much research has been done to develop systems which learn by example, or use *inductive learning*. Given a training set of instances, these systems often can "learn" the relationship between the inputs and outputs well enough to be able to receive an input and produce the correct output with a high probability, even if that input was not one of the examples. This ability is called *generalization*. The process of generalization is *modeling* and the result of the generalization could be called a *model*. Learning systems which can generalize accurately have the potential of being able to solve problems for which conventional programming solutions have not been found. They may also be able to avoid the high costs involved in writing specific programs or constructing expert systems to solve many problems. In some cases, they can perform with greater accuracy as well.

All areas of human interaction with the environment involve decision-making situations. Ideally, decision-making should be based on complete knowledge of the alternatives at hand, as well as their consequences. As the complex nature of the systems we are concerned with often makes exact predictions impossible, one usually has to rely on models, which provide tractable approximations to reality. Here, system analysis plays an important role [2_4], since only a well-informed decision maker is in a position to take well-founded decisions. There is an abundance of definitions for the word 'model'. A common definition, here expressed in the words of Neelamkavil [2_19], is very illustrative. "A model is a simplified representation of a system intended to enhance our ability to understand, predict and possibly control the behavior of the system."

3.2 Previous Work

Neural networks [2_16,2_35,2_36] provide several ways to utilize inductive learning. Some styles of neural networks have been successful in accurately learning to generalize from a training set. The main idea behind an Artificial Neural Network is to use several simple computational units, connected by weighted links through which activation values are transmitted. The units normally have a very simple way to calculate new activation values, given the values received through the connections; for example, summing their inputs and feeding the results through a monotonous transfer function. This activation is then spread through the network via the connections, finally resulting in activation of the output units, which is then interpreted as the modeling result. Training of the network consists in showing the patterns of the training set to the network, and letting it adjust its connections to obtain the correct output. One of the most popular neural network architectures used for modeling problems is the Multi-Layer Perceptron. The units are organized into different layers, and the network is said to be feed-forward when the activation values propagate in one direction only, from the units in the input layer, through a number of hidden layers, to end up in the output layer. The multi-layer perceptron is usually trained with the Error Back-Propagation method. Initially the connections in the network are set randomly, then the training samples are fed one at a time into the input layer and the activity propagated through the network to the output layer. The output of the network is then compared to the desired output, and the difference gives rise to an error signal which is fed backwards through the network, causing the connections to be updated in a way which will decrease the error the next time the same pattern is presented. By going through the training set in this way, several times, the connections are gradually adjusted to minimize the output error. Many researchers have devoted a lot of effort in neural networks. The combination of neural networks with other modeling technique is also an interesting area. In [2_21], the inputting data were first processed using rough sets, then the resulted rough sets were used to design neural networks. The neural networks were optimized by genetic algorithm and final knowledge in the form of rules was extracted from the final neural networks.

The Nearest-Neighbor (NN) pattern classifier [2_8] keeps the entire training set of instances, and generalizes by using the output of the "closest" instance to a new input. The *k*Nearest-Neighbor (*k*NN) algorithm uses the most common output value of the *k* closest neighbors during generalization, and performs better in the presence of noise. The Condensed Nearest-Neighbor rule [2_13] attempts to reduce storage requirements by keeping only selected instances, but suffers from intolerance to noise. The nested generalized exemplars (NGE) approach uses single instances or axis-parallel hyperrectangles that may cover several instances [2_29]. The NGE approach reduces storage requirements as well, but there is evidence that indicates that it may not compare favorably in terms of generalization accuracy with *k*NN in many domains [2_33]. A hybrid method [2_34] improves generalization accuracy to near that of the *k*NN methods. Instance-Based Learning (IBL) algorithms [2_1] reduce the number of instances that must be saved while remaining tolerant of noise. They are also *incremental*, meaning that additional training instances can be added easily, even after the initial learning stage is already complete. Case-Based Reasoning (CBR) systems [2_26] also attempt to use previously seen cases to generalize on new inputs. However, CBR systems may modify cases rather than simply storing a list of examples. Other notable induction learning algorithms that have had some success include ID3 [2_28, 2_6], which builds decision trees.

The above methods all use supervised training methods, where the correct class label has to be given when updating the connections. There is another kind of training called Reinforcement Learning [2_3], in which only a global signal indicating if the answer was wrong or right is given. This is sometimes useful when e. g. learning to play a game, and it is only possible to know if a whole sequence of moves was good or bad (if it lead to a win or a loss), and not exactly what should have been done in each move.

Fuzzy Logic [2_37], which is a generalization of truth-values, and which can be used in logical inference methods to handle concepts with "fuzzy" boundaries has turned out to be a powerful tool in classification. Fuzzy techniques can also be incorporated into various neural network models. Each class description consists of a set of fuzzy expressions allowing the evaluation of specific features and their logical operation. A fuzzy rule can have one single condition, or can consist of a combination of several conditions that have to be fulfilled for an object to be assigned to a class.

Given the character of the overall optimization task, and having at our disposal gradient-based techniques and evolutionary mechanisms of optimization, our design decision is to complete the overall design process through genetic optimization. Even though the gradient-based techniques could be applicable at the level of parametric refinement of the network, from the design point of view, it would be advisable to apply genetic optimization. Genetic algorithms (GAs) [2_17] are classified as computational modeling approaches, based on evolution. Through the years, the possibility of practical application has been demonstrated in many areas in research and industry. GA starts with random models as candidates to the problem and searches effectively for the optimal solution by means of reproduction, crossover, and mutation. This feature may alleviate the possibility of falling into local minima and maxima.

Decision tree has long been applied in data modeling. ID3 [2_27] is a popular and efficient method in training decision trees. Much research has been carried out in improving the performance of decision trees and many researchers have tried to combine fuzzy logic and decision trees to build modeling tools that are known as fuzzy decision trees [2_18,2_20, 2_31].

Many real-world search and optimization problems are naturally posed as nonlinear programming problems having multiple objectives. Due to lack of suitable solution techniques, such problems are artificially converted into a single-objective problem and solved. The difficulty arises because such problems give rise to a set of Pareto-optimal solutions, instead of a single optimum solution. It then becomes important to find not just one Pareto-optimal solution, but as many of them as possible. Classical methods are not very efficient in solving these problems because they require repetitive applications to find multiple Pareto-optimal solutions and on some occasions repetitive applications do not guarantee that distinct Pareto optimal solutions will be found. The population approach of evolutionary algorithms allows an efficient way to find multiple Pareto-optimal solutions simultaneously in a single simulation run [2_7, 2_10]. These concepts and techniques are to be discussed in detail in Chapter 11.

Chapter 4 Fuzzy Logic

4.1 Introduction to Fuzzy Logic

Fuzzy logic could be considered as the generalized form of two-valued logic. Fuzzy logic stems directly from the theory of fuzzy sets introduced by L. A. Zadeh[2_37]. Instead of using 1 or 0 to indicate whether an object belongs to a set or not, fuzzy set uses a value in the unit interval *[0,1]* to indicate the relationship between an object and the set. This value is determined by a predefined function, called a membership function. Generally speaking, a fuzzy set ***F*** defined in the domain of ***X*** is characterized by a membership function $\mu_F(x)$ which gives out the values in the range *[0,1]*. There are many ways to define a membership function for the fuzzy set. In many practical instances, fuzzy sets can be represented by the following parameterized functions (shown in Figure 4.1). For more forms of the membership functions, see reference [2_23].

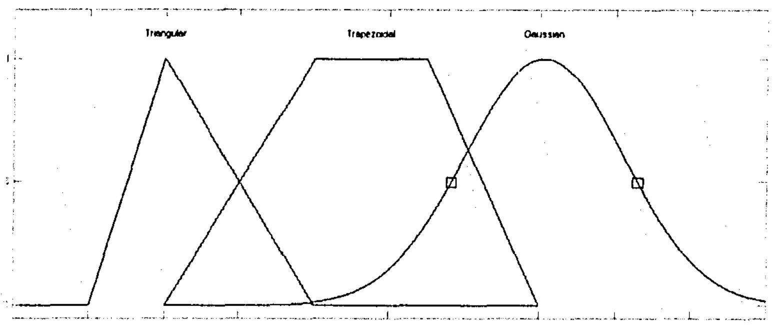


FIGURE 4.1 TRIANGULAR, TRAPEZOIDAL AND GAUSSIAN FUNCTIONS

In the same domain, we could define as many fuzzy sets as we need, but it would become more difficult to explain and understand when the number of fuzzy sets grows larger. It is preferable if the number of the fuzzy sets defined in a universe does not exceed 7. Usually we assign linguistic meanings to membership functions, such as low, medium and high for three membership functions of temperature. [2_23, 2_24] provide more information on fuzzy sets and membership functions.

With fuzzy logic, we treat original truth values in two-valued or many-valued logic as a linguistic characterization of numerical truth-values. "Thus, fuzzy logic concerns the principles of approximate reasoning [2_24, 2_22]." By applying the theory of fuzzy logic in solving modeling problems, we are able to generate fuzzy models to represent knowledge. This process is known as fuzzy modeling. According to [2_25], fuzzy modeling is primarily concerned with building models at a certain level of information granularity, which is conveniently quantified in terms of fuzzy sets. Rule-based models have assumed a dominant position in the plethora of fuzzy models. With the continuously growing diversity of modeling rules, one can project a significant development along these lines. Rule-based fuzzy models exploit the calculus of rule-based structures and, in general, can be structured as a series of "IF-THEN" conditional statements of the form:

IF Input_1 is A_{i1} and Input_2 is A_{i2} and ... and Input_n is A_{in} THEN output is Y

$i=1, 2, \ldots, c$, where $A_{i1}, A_{i2}, \ldots, A_{in}$, and Y are fuzzy sets (linguistic labels) of the corresponding system's variables being defined.

4.2 Data fuzzification for the datasets

All the data sets are fuzzified before being fed to the models. For each input attribute, it is considered to be continuous input if the containing values are floating point values or the integer values have more than 15 unique options. For continuous attributes, they will be fuzzified with a given number of fuzzy sets, using Gaussian membership function or Triangular membership function. For those attributes containing just integer values and having no more than 15 different options, they would be considered as discrete inputs and the fuzzification process applied on these attributes uses the 1-out-of-n approach. In this approach, there are n fuzzy sets if the number of discrete values is n. Each fuzzy set denotes 1 if the current value for this data point at this attribute is the discrete value represented by this fuzzy set. For each data point, just one fuzzy set in these n fuzzy sets is 1, while all others are 0. For example, if there are 3 discrete values, 1, 2, and 3, for the variable, we use 3 membership functions to represent fuzzy sets for this variable. We can code 1 as 1 0 0, 2 as 0 1 0 and 3 as 0 0 1. The first fuzzy set indicates if the variable is equal to 1 or not, the second set represents whether the variable is equal to 2, and the third indicates if the variable is equal to 3. In the experiments, all the data sets are randomly divided into 60-40 training-testing data sets.

Chapter 5 Evolutionary Computation

Evolutionary computation is based on Charles R. Darwin's theory of natural selection, and has been extensively used in optimization since the 1960s. In natural systems, evolution presents the changes (sometimes not optimal ones) in behaviors of organisms, over the biological hierarchy of cells, organs, individuals, and populations. Evolutionary computation focuses on an analog to natural systems. It adopts stochastic methods to generate new individuals and its searching strategy imitates the natural process of competing to survive, and successful inheritance. By using evolutionary computing methods, we are able to solve those problems that are difficult to find satisfactory solutions for using traditional approaches. Thomas Bäck et al. considered that "the most significant advantage of using evolutionary search lies in the gain of flexibility and adaptability to the task at hand, in combination with robust performance (although this depends on the problem class) and global search characteristics." [2_2] Evolutionary computation works on one or more sets of possible solutions to the problem. Each possible solution is referred to as an individual or a chromosome, and a set of chromosomes forms a population. During the process of solving the problem, the population is continuously changed by a cycle of selecting better chromosomes and generating new chromosomes by applying operators such as mutation and/or recombination. The selection of the chromosomes is based on the values given by an evaluation function applied to the chromosomes. The value that the evaluation function gives is called *fitness of the chromosome* and the evaluation function is named the *fitness function*. The selection process can be realized in many different ways, one of which is stochastic sampling with replacement. In this method, the entire population is mapped onto a roulette wheel where each genotype is represented by the area corresponding to its fitness value. Individuals forming an intermediate population are chosen by repetitive spinning of the roulette wheel.

In the thesis, we will be using two important, strongly related but independently developed, approaches in evolutionary computation: Evolutionary Strategy (ES) and Genetic Algorithm (GA)

5.1 Evolution strategies

Evolution strategies (ES) were developed by Rechenberg and Schwefel in the 1960s. They are algorithms that imitate the principles of natural evolution, and have been used as a method for solving parameter optimization problems [2_17] for a long time. In their early stage, evolution strategies were carried out with a population consisting of just one individual and one genetic operator (mutation) only. This method is called the two-member evaluation strategy. The most significant difference between simple genetic algorithms and evolution strategies lies in the representation of the variables. In ES, an individual was represented as a pair of floating point-valued vectors, i.e., $v = (x, \sigma)$. Here, the first vector x represents a point in the solution space; the second vector σ represents the corresponding standard deviation. The individual is altered by mutation by $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathrm{N}(0,\sigma)$, where $\mathrm{N}(0,\sigma)$ is a vector of independent identically normally distributed random numbers with a mean of zero and standard deviation σ. This is consistent with biological observation that smaller changes occur more often than larger ones [2_17]. The mutated individual replaces its parent if and only if the mutated individual has a better fitness and satisfies all existing constraints. Otherwise, the offspring is destroyed and the population remains unchanged.

Multi-member evolution strategies were also introduced [2_30]. Here a population consists of m individuals at generation g producing k offspring. The m best of $(m + k)$ individuals will survive as parents for the next generation. This model allows those individuals who have better fitness to exist for a long time, until better solutions appear, thereby causing premature convergence. In order to prevent premature convergence, a simple solution could be introduced to the multi-member evolution strategy. Here, a population consisting of m individuals at generation g produces k offspring, where $k > m$. During the selection stage, the m best of the k offspring are chosen to be parents of the following generation.

5.2 Genetic Algorithm

The Genetic Algorithm (GA) was introduced by Holland in 1962 in the U. S. A., at almost the same time as the evolutionary strategy emerged in Germany. The genetic optimization aimed at by the structural and parametric development of the network exploits Genetic Algorithms (GAs). In a nutshell, GA is a biologically inspired search method considering the principles of natural selection [2_9, 2_12]. In this setting, all possible solutions are encoded into a population of chromosome-like data structures named genotypes. The genotypes are "translated" into phenotypes that represent potential solutions to the problem. The phenotypes are evaluated based on their ability to solve the problem and quantified by some fitness function. Each time the phenotype is fed into this function, a fitness value is obtained that is assigned to the original genotype and represents its fitness. The results of this evaluation are used when forming a new collection of potential solutions. The choice of individuals to be reproduced is guided by some selection mechanism.

Genetic operators, i.e., crossover and mutation, are employed to form a new population. Crossover is applied to the randomly paired genotypes from the intermediate population with some probability p_c. In the case of mutation, all bits in all substrings of genotypes are altered with the probability p_m. A sequence of such actions is repeated until some final criterion has been satisfied. All steps described above are then repeated. The underlying GA pseudo-code is presented below; here $P_{GA}(t_{GA})$ denotes the population of solutions at some generation (denoted here by t_{GA}):

```
$t_{GA} := 0$
initialize $P_{GA}(t_{GA})$
evaluate $P_{GA}(t_{GA})$
**while not** terminate **do**
    $P'_{GA}(t_{GA}) :=$ select $(P_{GA}(t_{GA}))$;
    $P''_{GA}(t_{GA}) :=$ crossover $(P'_{GA}(t_{GA}))$;
    $P'''_{GA}(t_{GA}) :=$ mutate $(P''_{GA}(t_{GA}))$;
    $P_{GA}(t_{GA}+1) := P'''_{GA}(t_{GA})$;
    $t_{GA} := t_{GA} + 1$;
    evaluate $P_{GA}(t_{GA})$
**od**
```

Classified by the gene value, GAs are divided into two types: Binary Coded Genetic Algorithm (BCGA) and Real Coded Genetic Algorithm (RCGA) [2_14]. BCGAs use the binary or binary-like data, which are discrete values, as individual genes, while RCGA uses real values as genes. The use of real parameters makes it possible to utilize large domains for the connections. This is difficult to achieve in binary implementations where increasing the domain would mean sacrificing precision (assuming a fixed length for the chromosomes). Another advantage when using real parameters is their capacity to exploit the granularity of the functions with continuous variables, where the concept of granularity refers to the fact that slight changes in the variables correspond to slight changes in the function. Along these lines, a highlighted advantage of the RCGAs [2_14] is the capacity for local tuning of the solutions. A more detailed introduction to ES, GA, BCGA, RCGA and their applications could be found in [2_5, 2_11, 2_14, 2_30].

Chapter 6 Experimental data used in the research

In this research, we selected several well-known data sets from the UCI Machine Learning Data Repository [2_15], including Boston housing data, Abalone data, Auto-MPG data, Machine CPU data and IRIS data. We also adopted the software metric data. We will introduce these datasets below.

6.1 Boston housing dataset

The Boston Housing dataset is a small but widely used dataset derived from information collected by the U.S. Census Service concerning housing in the Boston area, which has been used extensively throughout the literature to benchmark algorithms. The goal is to predict the median value of a house, i.e., price. The dataset contains 506 records of the real estate price and related characteristics of the houses in Boston. The real estate price is the output of the record; 13 properties are used to describe the houses. The variables are described in Table 6.1, which contains the variable description, the abbreviations used in generated rules, and the variable type(discrete or continuous).

No.	Abbr.	Description	Type
1	CRIM	Per capita crime rate by town	Cont.
2	ZN	Proportion of residential land zoned for lots over25,000 sq.ft.	Cont.
3	INDUS	Proportion of non-retail business acres per town	Cont.
4	CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)	Disc.
5	NOX	Nitric oxides concentration (parts per 10 million)	Cont.
6	RM	Average number of rooms per dwelling	Cont.
7	AGE	Proportion of owner-occupied units built prior to 1940	Cont.
8	DIS	Weighted distances to five Boston employment centres	Cont.
9	RAD	Index of accessibility to radial highways (24,8,7,6,5,4,3,2,1)	Disc.
10	TAX	Full-value property-tax rate per $10,000	Cont.
11	PTRATIO	Pupil-teacher ratio by town	Cont.
12	B	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town	Cont.
13	LSTAT	% lower status of the population	Cont.
14	MEDV	Median value of owner-occupied homes in $1000's (Target)	Cont.

Table 6.1 Attributes occurring in the Boston Housing Data

(Cont. =continuous, Disc. =discrete)

6.2 Abalone data

This data set consists of eight inputs, namely the sex of the specimen (male, female, or infant), the length, the diameter, the height, the total weight, the shucked weight, the weight of the viscera, and finally the shell weight of the specimen, that were used to predict the age of the abalone. The original owner of the database is the Marine Research Laboratories in Tasmania, Australia. The age in years of an abalone can be obtained by adding 1.5 to the number of rings. The number of rings varies between 1 and 29 and we expect that two abalones with the same number of rings should also present similar values for the independent attributes sex, length, diameter, height, and so on. In other words, the degree of dissimilarity between crustaceans computed on the independent attributes should be proportional to the dissimilarity in the dependent attribute (i.e., the difference in the number of rings). The database contains 4,177 cases of marine crustaceans. 8 input variables and 1 output variable are stored in this dataset. Table 6.2 includes the abbreviation, description and type of the variables.

No.	Abbr.	Description	Type
1	Sex	Discrete value (Male, Female, and Infant)	Disc.
2	Length	Longest shell measurement	Cont.
3	Diameter	Perpendicular to length in millimeters	Cont.
4	Height	The thickness with meat in shell in millimeters	Cont.
5	Wholeweight	Total weight of abalone in grams	Cont.
6	Shuckedweight	Weight of meat in grams	Cont.
7	Visceraweight	Gut weight (after bleeding) in grams	Cont.
8	Shell weight	The weight of the shell after being dried in grams	Cont.
9	Rings	Gives the age in years when plus 1.5(output) 29 values	Cont.

Table 6.2 Attributes occurring in the Abalone Data

(Cont. =continuous, Disc. =discrete)

6.3 Auto-MPG data

This data set contained a set of records of different cars and their mileage (mpg) rates. The numeric attributes contained features such as displacement, horsepower, acceleration, weight, and model year of a car. The target to be predicted in this problem is the city-cycle fuel consumption of different car models in miles per gallon. There are 393 instances and 7 attributes – 6 numeric and 1 categorical. (See Table 6.3.)

No.	Abbr.	Description	Type
1	CYLIN	Number of cylinders (3, 4,5, 6,8)	Discrete
2	DISPL	Displacement	Continuous
3	HorseP	Horsepower	Continuous
4	Weight	Weight	Continuous
5	ACCE	Acceleration	Continuous
6	Model Y	Model year(82,81,80,79,78,77,76,75,74,73,72,71,70)	Discrete
7	Origin	Origin (1 = USA; 2 = Europe; 3 = Japan)	Discrete
8	Fuel Efficiency	fuel efficiency (miles per gallon) (output variable)	Continuous

Table 6.3 Attributes in Auto-mpg Data

6.4 Machine CPU data

This dataset is concerned with Relative CPU Performance. There are a total of 209 instances in the training data, and 7 attributes. It contains 6 continuous input variables which show the specific information about the CPU. The published relative performance works as the output variable in this dataset. Refer to Table 6.4 for details.

No.	Abbr.	Description	Type
1	MYCT	Machine cycle time in nanoseconds	Continuous
2	MMIN	Minimum main memory in kilobytes	Continuous
3	MMAX	Maximum main memory in kilobytes	Continuous
4	CACHE	Cache memory in kilobytes	Continuous
5	CHMIN	Minimum channels in units	Continuous
6	CHMAX	Maximum channels in units	Continuous
7	PRP	Published relative performance (output)	Continuous

Table 6.4 Attributes used in Relative CPU Performance Data

6.5 IRIS data

This data set concerns four attributes – sepal length (SL), sepal width (SW), petal length (PL), petal width (PW); and deals with three classes – Setosa, Versicolor and Virginica. All four input attributes are continuous data. The data set consists of 150 patterns, 50 in each class.

6.6 Software metric data

The software metrics dataset [2_32] contains 366 software objects in a Java-based biomedical image data analysis system. All 366 software objects were labeled by an experienced software architect in terms of usability. Three architects (experts A, D and V) were asked to rank each software object. That is, based on experience, they were asked to assign a value from 1 to 5 that ranks the overall usability of a particular software object. Classes with low ranking are determined to be difficult to use. A class ranked 1 should definitely be subject to a review while a class 5 is considered very easy to use. A set of 64 metrics was obtained from an evaluation version of Borland's Together Soft package and an in-house metrics parser. In this study, we combined classes 1 & 2 to be a new class 2 and classes 4 & 5 to be a new class 4, which changes the problem into a three-class problem.

No.	Abbr.	Description	Type
0	NAME	Object Name.	
1	TYPE	Type: GUI (=1), Data Model (=2), Algorithm (=3), Other (=4).	D(4)
2	METH	# methods.	Cont.
3	LOC	# lines of code.	Cont.
4	ALOC	Mean LOC per method.	Cont.
5	MLOC	Median LOC per method.	Cont.
6	RCC1	Ratio of comment lines of code to total lines of code including white space and comments.	Cont.
7	RCC2	True Comment Ratio	Cont.
8	TOK	# tokens.	Cont.
9	ATOK	Mean TOK per method.	Cont.
10	MTOK	Median TOK per method.	Cont.
11	DEC	# decisions: for, while, if, switch, etc.	Cont.
12	ADEC	Mean DEC per method.	D(13)
13	MDEC	Median DEC per method.	D(11)
14	WDC	Weighted # decisions based on nesting level i: Sum[$i*n_i$]	Cont.
15	AWDC	Mean WDC per method.	Cont.
16	MWDC	Median WDC per method.	D(12)
17	INCL	# inner classes.	D(12)
18	DINH	Depth of inheritance.	D(5)
19	CHLD	# children.	D(10)
20	SIBL	# siblings.	D(15)
21	FACE	# implemented interfaces.	D(6)
22	RCR	Code reuse: ratio of overloaded inherited methods to those that are not.	Cont.
23	CBO	Coupling between objects.	Cont.
24	LCOM	Lack of cohesion of methods.	Cont.
25	RFO	Response for an object. Response set contains the methods that can be executed in response to a message.	Cont.

26	RFC	Response for class.	Cont.
27	MNL1	Maximum method name length.	Cont.
28	MNL2	Minimum method name length.	Cont.
29	MNL3	Mean method name length.	Cont.
30	MNL4	Median method name length.	Cont.
31	ATCO	Attribute Complexity.	Cont.
32	CYCO	Cyclomatic Complexity.	Cont.
33	DAC	Data Abstraction Coupling.	Cont.
34	FNOT	Fan Out.	Cont.
35	HLDF	Halstead Difficulty.	Cont.
36	HLEF	Halstead Effort.	Cont.
37	HLPL	Halstead Program Length.	Cont.
38	HLVC	Halstead Program Vocabulary.	Cont.
39	HLVL	Halstead Program Volume.	Cont.
40	HLON	Halstead # operands.	Cont.
41	HLOR	Halstead # operators.	Cont.
42	HLUN	Halstead # unique operands.	Cont.
43	HLUR	Halstead # unique operators.	Cont.
44	MIC	Method Invocation Coupling.	D(8)
45	MAXL	Maximum # levels.	D(14)
46	MAXP	Maximum # parameters.	D(12)
47	MAXO	Maximum size operations.	Cont.
48	ATTR	# attributes.	Cont.
49	ADDM	# added methods.	Cont.
50	CLAS	# classes.	D(12)
51	CHCL	# child classes.	Cont.
52	CONS	# constructors.	D(9)
53	IMST	# import statements.	Cont.
54	MEMB	# Members.	Cont.
55	OPER	# operations.	Cont.
56	OVRM	# overridden methods.	D(15)
57	REMM	# remote methods.	Cont.
58	PKGM	% package members.	Cont.
59	PRVM	% private members.	Cont.
60	PROM	% protected members.	Cont.
61	PUBM	% public members.	Cont.
62	DEMV	Violations of Demeters Law.	Cont.
63	WMC1	Weighted Methods per class.	Cont.
64	WMC2	Weighted methods per class.	Cont.

Table 6.5 Attributes occurring in the Software metric Data

(Cont. =continuous, D(*n*) =discrete, has *n* options)

After the combination of the original classes, there are 3 levels of usability in the dataset. Figure 6.1 shows the class distribution of the three experts.

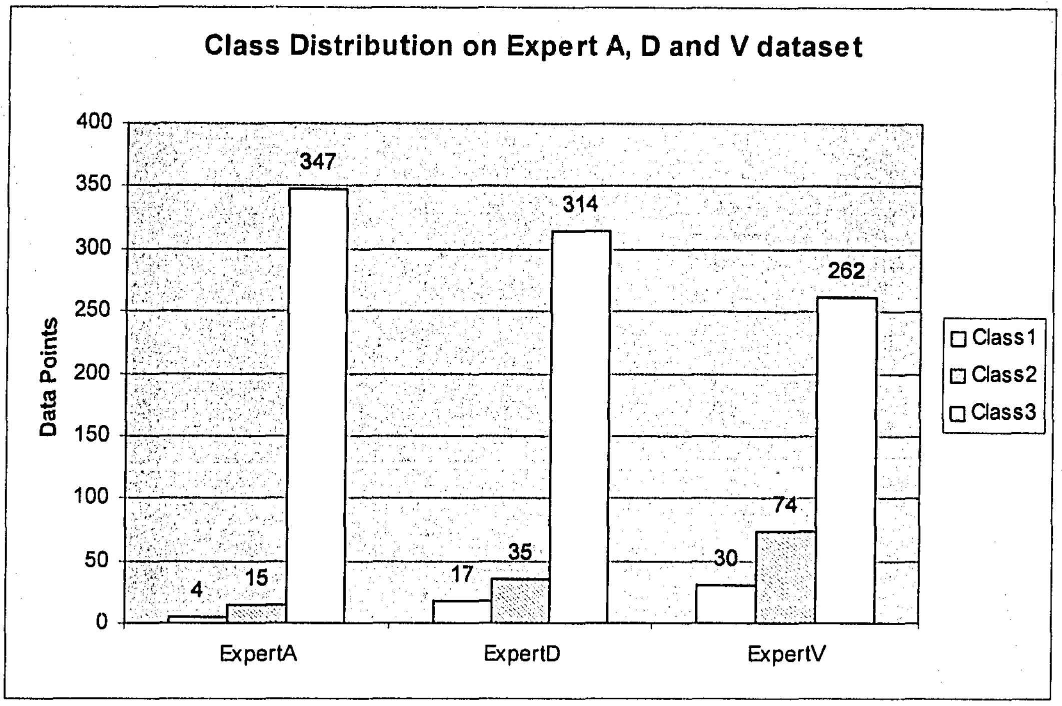


FIGURE 6.1 HISTORY GRAPHS OF THE USABILITY DATA FOR EXPERTS A, D AND V

In Figure 6.1, for Experts A, D and V, the left column is for class 1, the middle one is for class 2 and the right column is for class 3. From Figure 6.1, we can see that the usability data for expert A is the most skewed dataset, while expert D's dataset is next, and expert V's is the last.

Part III Data Modeling Using Reed-Muller Binary Decision Trees

In this part, we build the data modeling systems based on the Reed-Muller Binary Decision Tree (RMBDT) structure. The related concepts are introduced first,. then the modeling system is described and extensive experiments are performed. Finally, the analysis of the results is presented and discussed.

Chapter 7 Reed-Muller Models

The Binary Decision Diagram (BDD) method has been widely used for the synthesis, analysis and optimization of both combinational and sequential logic circuits. The Binary Decision Tree (BDT), as the specific form of BDD, has been applied in pattern recognition [3_12, 3_13] and classification [3_23].

Any Boolean function can be represented in modulo-2 algebra as the complement-free ring sum or Reed–Muller (RM) expansion [3_3, 3_4, 3_14]. For a given Boolean function, each Reed–Muller expansion is both unique and canonic. The use of Reed-Muller expansions to represent Boolean functions has gained increasing popularity [3_11, 3_17], and the use of the Reed-Muller Binary Decision Diagram (RMBDD) and the Reed-Muller Binary Decision Tree (RMBDT) for Reed-Muller expansion has become a research topic in its own right [3_15, 3_20, 3_22]. RMBDD and RMBDT have been applied in variable ordering [3_21] and construction of Reed-Muller universal logic module models [3_2].

Logic, multi-valued logic, and fuzzy logic lie at the heart of data understanding. It has become clear that fuzzy logic, strongly supporting the development of fuzzy models, is an essential conceptual component for the successful design of various user-centric models.

In the process of our research activity, we found that RMBDT, working with fuzzy logic, could also produce good generalization ability. The objectives of research on RMBDT are threefold: First, we revisit the concept of the Reed-Muller Binary Decision Tree, demonstrating how RMBDT is able to provide the function of generalization ability; second, we develop an overall architecture of the RMBDT; and third, a comprehensive genetic design procedure is established.

7.1 Reed-Muller expansion

In recent years, Reed-Muller expansion –alternative means of representing logic functions based on the operations of modulo-2 arithmetic – has presented an attractive and rewarding field of investigation. The Reed-Muller (RM) approach of AND/EX-OR logic realization has certain advantages over the Boolean AND/OR method. For instance, RM circuits have desirable features of ease of complementing and testing, and often require fewer product terms using PLA implementation. Moreover, logic functions, which do not minimize well in the Boolean domain, can be reduced substantially if implemented in RM logic [3_18]. An AND/OR Boolean function with n variables is represented by

$$f(x_n,\ldots,x_1)=\sum_{i=0}^{2^n-1} a_i m_i = a_0 m_0 + a_1 m_1 + \ldots a_r m_r \qquad (1)$$

where m_i are the minterms, a_i=0 or 1, and $r=2^n-1$. Since the minterms are mutually exclusive, $m_i \times m_j = 0\ \forall i, j\ (i \neq j)$, the OR (+) can be replaced by the EX-OR (⊕), or modular-2 sum of products, giving the Reed-Muller form as

$$f=b_0p_0 \oplus b_1p_1 \oplus \ldots \oplus b_rp_r \qquad (2)$$

where p_i are the product terms [3_1, 3_10] and b_i=0 or 1.

7.2 Reed-Muller binary decision tree

A Reed-Muller binary decision tree (RMBDT) is a graphical representation of a generalized Reed-Muller expansion. An n-variable RMBDT is a rooted, directed, acyclic graph with n levels of nodes and one level of leaves. It consists of nodes interconnected by branches. A terminal node, a leaf, assumes the binary value 0 or 1 and has a single output branch. A non-terminal node represents a variable, x_i (using direct form or complement form), and has one output branch and two input branches. The left input (0-input) branch indicates that the node variable is absent and the right input (1-input) branch indicates that the node variable is present.

Shannon's expansion theorem may be employed to decompose the Boolean function by expanding with respect to any variable x_i; hence for any n-variable function f:

$$f = \overline{x}_i f_i^0 + x_i f_i^1 \tag{3}$$

The residual functions f_i^0 and f_i^1 are cofactors of f, with x_i taking the values 0 and 1, respectively.

$$\begin{aligned} f_i^0 &= f(x_n \ldots x_{i+1}, 0, x_{i-1} \ldots x_1) \\ f_i^1 &= f(x_n \ldots x_{i+1}, 1, x_{i-1} \ldots x_1) \end{aligned} \tag{4}$$

Using this notation, it can be shown that

$$f = f_i^0 \oplus x_i (f_i^0 \oplus f_i^1) \tag{5}$$

The represented function by a non-terminal node is shown in Figure 7.1.

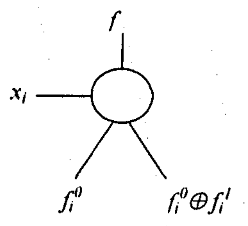


FIGURE 7.1 NON-TERMINAL NODE REPRESENTING FUNCTION VARIABLE X_i

The root of the tree is a single node and is defined as level 1. The output of the root is the output of the RMBDT. The 0-input branch of the root is connected to the output edge of one of the two nodes present at level 2. Similarly, the 1-input branch is connected to the output edge of the remaining level 2 node. In general, at level l, there are 2^{l-1} nodes. The output edge of each of these 2^{l-1} nodes is connected to an input branch of a node at level $(l-1)$, and the 2^{l} input branches of these level l nodes are connected to the output edges of the 2^{l-1} nodes at level $(l+1)$. At the final level, that is level $(n+1)$, there are 2^{n} leaves, and the output edge of each leaf is connected to one of the input branches of the nodes at level n. The depth of a RMBDT is defined as the depth of the tree containing just non-terminal nodes. For a RMBDT with the depth of n, it contains $2^{n}-1$ non-terminal nodes and 2^{n} terminal nodes. All the non-terminal nodes have the same function as the node in Figure 7.1. A sample RMBDT with depth of 2 is shown in Figure 7.2. By collecting the expressions from leaves with value 1, there will be 2 product terms in this RMBDT. Moving up the tree and following the rules of 1-input and 0-input branch, we find that the logical function presented by this RMBDT is

$$f=x_1 \oplus x_2 \tag{6}$$

We can also get Equation 6 by the following process (refer to Equations 4 & 5):

$f=f_2^0 \oplus x_2 f_2^1$

Where $f_2^0 = 0 \oplus x_1 (1)=x_1$ and $f_2^1 = 1 \oplus x_1 (0) = 1$

So $f= x_1 \oplus x_2 (1)= x_1 \oplus x_2$

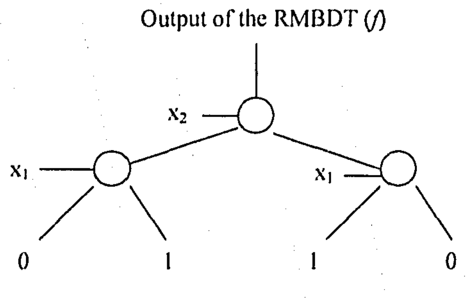


FIGURE 7.2 SAMPLE MULTI-LEVEL RMBDT

Chapter 8 Reed-Muller Binary Decision Trees in Data Modeling

8.1 Description of the Model

The development of the proposed RMBDT model involves parametric design decisions. The parameters of the model deal with the determination of ordered input variable indexes and optimal values of the terminal node binary values. We choose genetic-based optimization in training models. The decision to choose genetic algorithms is well supported by two arguments:

-Models we build usually accept fewer inputs, compared with the total input space. One could envision that the number of variables could be substantially reduced without any visible deterioration of the approximation abilities of the model, and the sequence of the selected variable set is sensitive to the performance of the tree. The depth of the tree is defined at the beginning of the design process (the value could be adjusted) and the optimization at this end is not of concern – a simple enumeration is far more productive than involving the depth as a part of the optimization process. The selection of a subset of the variables is a combinatorial problem and here the role of genetic optimization becomes essential. There are several design scenarios that in essence contribute to the increased dimensionality of the problem.

-Using genetic optimization in the optimization of the values of the *parameters* of the terminal nodes is also advantageous: the structure could exhibit significant depth that drastically hampers the efficiency of the exhaustive searching based method. The larger the depth of the structure, the more difficulties arise, and the lower the efficiency of the ensuing learning.

Having the genetic-based optimization task in mind, the chromosome consists of two parts (refer to Figure 8.1). The first part involves a binary representation of indexes of the input variables. The second part of the genotype is concerned with binary values of leaves. Both of these parts are coded as integers. This arrangement of the chromosome is very much in line with the phenotype-genotype transformations encountered in the literature, cf. [3_7- 3_9].

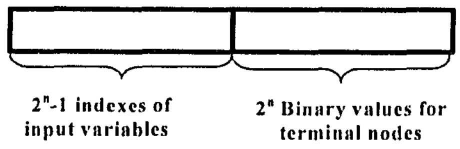

2ⁿ-1 indexes of input variables
2ⁿ Binary values for terminal nodes

FIGURE 8.1. A SCHEMATIC VIEW OF A CHROMOSOME DESCRIPTION OF THE MODEL

The model is evaluated by means of some performance index expressing a measure of fit of the resulting structure. Given the training dataset is given in the standard format of input-output pairs, that is, ($\mathbf{x}$(k), **target**(k)), k=1,2,.., N, the sum of squared errors

$$Q = \sum_{k=1}^{N} (\mathbf{x}(k) - \mathbf{target}(k))^T (\mathbf{x}(k) - \mathbf{target}(k))$$

is a commonly used measure expressing the fitness of the model. Given this, one could take the fitness function to assume the form $\frac{1}{1+Q}$. Here, the genetic optimization leads to the maximization of the fitness function. In the assessment of the quality of the model one could use any related index directly based upon Q, for example, a well-known root mean squared error (RMSE) performance measure.

Once the evaluation and selection have been completed, some of the chromosomes are subject to crossover and mutation. In the case of a basic crossover, called a one-point crossover, the genotypes of randomly selected pairs are split into two parts and then recombined into a new pair of genotypes.

8.2 Experimental setup

As usual, the data set is split into the 60-40 training and testing combination. The construction of the logic relationship between information granules of input and output attributes is considered as 3 fuzzy sets, quantified as *low*, *medium*, and *high* for continuous attributes. In the case of discrete variables assuming several discrete values, we consider a 1-out of – n decoding scheme. Because of this arrangement, we develop three separate models in which each deals with the corresponding linguistic granule of output result. In the case of the approximation nature of the problem, the development of the model could be conveniently monitored by reporting performance on the training and testing sets; this helps to achieve a tradeoff between sound approximation and generalization abilities.

In the experiments, the GA was run for 200 generations with the size of the population being equal to 500. Under these conditions, the results stabilized and larger values of these parameters did not contribute to any significant changes in the performance of the model.

In the experiments for construction of data models, the models with depth from 2 to 7 are trained. 10 experiments are carried out on each model and the performance indexes are recorded. The performance of the model could be immediately characterized by means of the mean RMSE and the standard deviation achieved on the training and testing set.

The experimental results are shown in tables and plots. Tables contain the mean value and the standard deviation over 10 experiments. The plots show the performance as it is varied via the changes in the depth of RMBDT. Lines are drawn between the mean values to show the trend of the performance. Enclosed vertical lines describe the standard deviation change, on which the mean values are marked as circles. The filled circles are the optimal mean values among the results

8.3 Experimental results

	Level 2		Level 3		Level 4		Level 5		Level 6		Level 7	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Low median price of real estate												
Mean	0.2276	0.2242	0.1772	0.1734	0.1681	0.1837	0.1662	0.1829	0.1696	0.1719	0.1800	0.1796
SD	0.0036	0.0092	0.0059	0.0097	0.0047	0.0143	0.0038	0.0084	0.0054	0.0074	0.0089	0.0075
Medium median price of real estate												
Mean	0.2482	0.2490	0.2114	0.2042	0.2039	0.2141	0.2029	0.2112	0.2016	0.2121	0.1985	0.2077
SD	0.0047	0.0100	0.0059	0.0084	0.0063	0.0070	0.0121	0.0188	0.0048	0.0120	0.0058	0.0071
High median price of real estate												
Mean	0.1389	0.1456	0.1327	0.1214	0.1093	0.1228	0.1315	0.1267	0.1398	0.1435	0.1531	0.1457
SD	0.0148	0.0211	0.0037	0.0143	0.0112	0.0277	0.0122	0.0164	0.0072	0.0138	0.0062	0.0262

SD: Standard Deviation

Table 8.1 Experimental results on Boston data

A comprehensive set of experiments is aimed at presenting the detailed design of the models as well as quantifying their performance. We use several representative examples from the Machine Learning Data Repository to demonstrate the generalization ability of RMBDT. The optimization is realized at the internal logic based level.

<u>Experiment 1</u> Boston Housing Data

In the analysis of Boston housing data, we are interested in the construction of the logic relationship between information granules of the price of real estate (that is, quantified as *low*, *medium*, and *high*). Therefore, we end up with logic models of real estate of *low*, *medium* and *high* price. (For the attributes and explanation see, Chapter 6.)

The experimental results are recorded in Table 8.1. The plots illustrating the performance of each model structure on training dataset and testing dataset are shown in Figure 8.2-8.3.

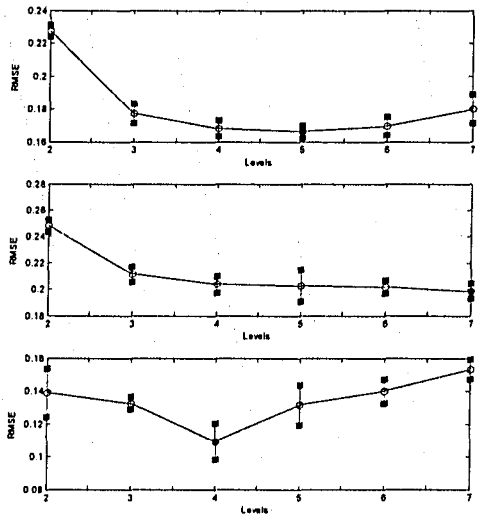


FIGURE 8.2 LOW (UPPER), MEDIUM (MIDDLE) AND HIGH (LOWER) MEDIAN PRICE OF REAL ESTATE ON TRAINING DATA

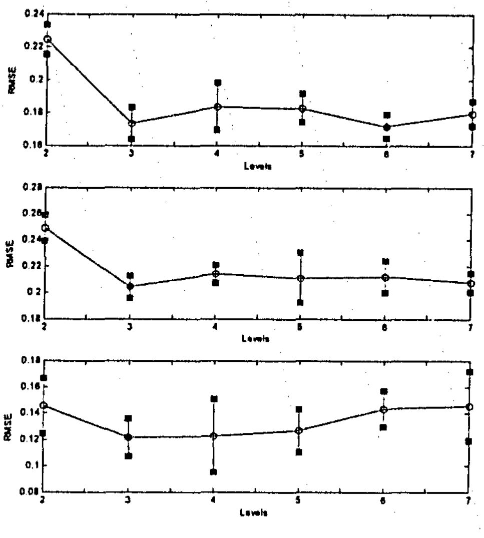


FIGURE 8.3 LOW (UPPER), MEDIUM (MIDDLE) AND HIGH (LOWER) MEDIAN PRICE OF REAL ESTATE ON TESTING DATA

The best models in the testing dataset for low, medium, and high price of real estate are found at the depths of 3, 5, and 4, respectively. The RMSE for low, medium and high price of real estate are 0.1622, 0.1861, and 0.0951. The sample plot, which is the RMBDT model for low price of real estate, is shown in Figure 8.4.

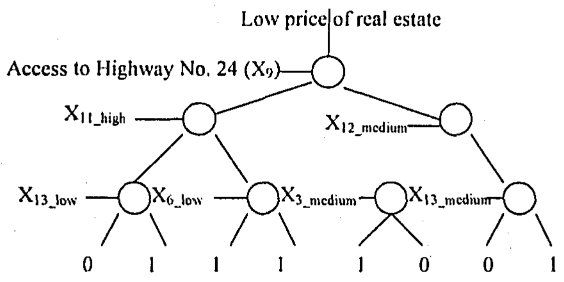


FIGURE 8.4 BEST RMBDT MODEL IN TESTING DATASET OF LOW PRICE OF REAL ESTATE

From Figure 8.4, we can derive the logical expression in the form of AND/EXOR arithmetic and transform it into the AND/OR forms. The process is shown below.

$$\begin{aligned}
&X_{13\_low} \oplus X_{6\_low} X_{11\_high} \oplus AHN_{24} \oplus X_{13\_medium} X_{12\_medium} AHN_{24} \\
&= (X_{13\_low} X_{6\_low} X_{11\_high} \overline{X_{13\_medium}} AHN_{24} + X_{13\_low} X_{6\_low} X_{11\_high} \overline{X_{12\_medium}} AHN_{24} + \\
&\overline{X_{6\_low}}\,\overline{X_{13\_low}}\,\overline{X_{13\_medium}} AHN_{24} + \overline{X_{6\_low}}\,\overline{X_{13\_low}}\,\overline{X_{12\_medium}} AHN_{24} + \\
&\overline{X_{11\_high}}\,\overline{X_{13\_low}}\,\overline{X_{13\_medium}} AHN_{24} + \overline{X_{11\_high}}\,\overline{X_{13\_low}}\,\overline{X_{12\_medium}} AHN_{24}) + \\
&\overline{X_{13\_low}} X_{6\_low} X_{11\_high} X_{13\_medium} X_{12\_medium} + \overline{X_{13\_low}} X_{6\_low} X_{11\_high} \overline{AHN_{24}} + \\
&X_{13\_low} \overline{X_{6\_low}} X_{13\_medium} X_{12\_medium} + X_{13\_low} \overline{X_{6\_low}}\,\overline{AHN_{24}} + \\
&X_{13\_low} \overline{X_{11\_high}} X_{13\_medium} X_{12\_medium} + X_{13\_low} \overline{X_{11\_high}}\,\overline{AHN_{24}}
\end{aligned}$$

Of course, we can continue to create rules from the former equation, but in the process of changing AND/EXOR logical expression into AND/OR logical expression, we have inevitably introduced the negative form to each variable. Sometimes, the negative form of the variable has no reasonable meaning in the rules. So, some side effects will show up in the final rule set. On the other hand, the transformation between two logical expressions is a tedium hard work. Errors are very easy to be made during the process. The former example comes from a RMBDT with depth of 3. If the depth comes to bigger than 3, it will become really difficult in this process of transformation. In practice, we usually do not treat RMBDT as the model which is able to generate the logical expression that is easy to interpret.

On observation of Figures 8.2-8.3, we see that the medium sized RMBDT model (depth from 3 to 6) usually gives better performance on both training and testing datasets.

Experiment 2 Other Machine Learning Datasets

We designed RMBDT models for several commonly used datasets, such as Auto-mpg, Machine CPU, and Abalone. The development process is the same as for the Boston Housing Dataset. In what follows, we briefly report the key results. The plots for the testing datasets are shown in Figures 8.5-8.7.

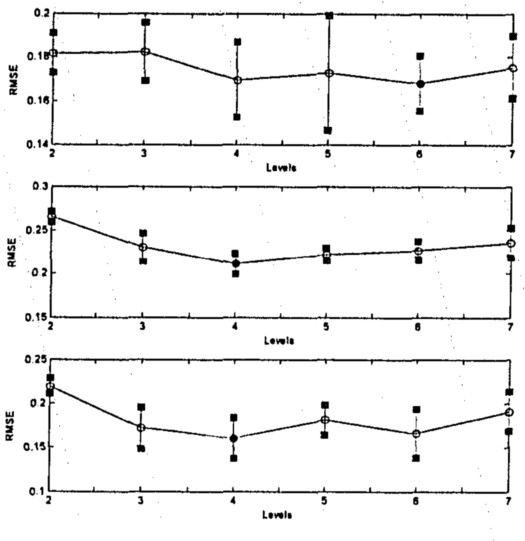


FIGURE 8.5 LOW (UPPER), MEDIUM (MIDDLE) AND HIGH (LOWER) MPG ON AUTO MPG TESTING DATA

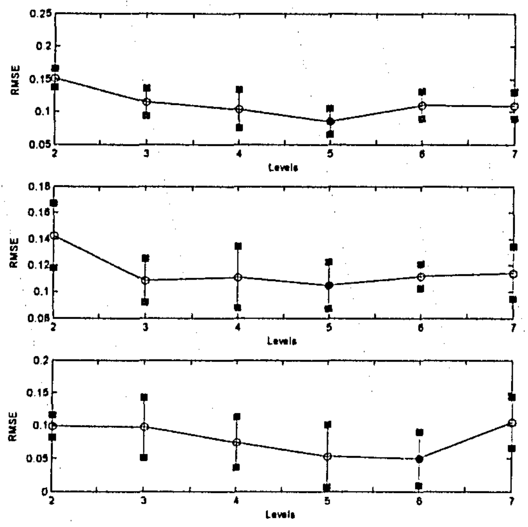


FIGURE 8.6 LOW (UPPER), MEDIUM (MIDDLE) AND HIGH (LOWER) PERFORMANCE OF CPU ON MACHINE CPU TESTING DATA

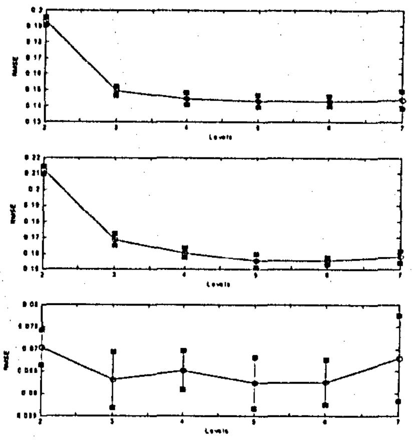


FIGURE 8.7 LOW (UPPER), MEDIUM (MIDDLE) AND HIGH (LOWER) OUTPUT ON ABALONE TESTING DATA

Figures 8.5 – 8.7 show the trend with this kind of model that models with medium size structure give better performance on predicting unseen data. When the model structure becomes more complex, the predicting ability goes down, because of over fitting of the models on the training data set.

8.4 Discussions

In this study, we have proposed a logic network composed of the Reed-Muller binary decision tree. We have proposed a genetic scheme of optimization of the network, with the intent of addressing the structural facet of learning. The critical issues in this learning deal with the depth of the tree: this design parameter implies the number of input variables used by the network, therefore one reduces the number of input variables existing in the problem.

In the design of the network we have demonstrated the role of the interface between the physical variables and those of logic character required by the model. The encoding scheme dwells on fuzzy sets treated as a collection of semantically sound information granules. From the experiments carried out on the data modeling approach, we can see that the RMBDT with depth from 3 to 6 usually gives better generalization ability than other trees with less or more complex structure.

Part IV Data Modeling Using Cascade OR/AND Neuron Based Fuzzy Neural Networks

We now introduce a novel fuzzy neural network based on OR/AND neurons. In this part, the OR/AND neurons are organized according to the Cascade OR/AND neuron-based Fuzzy Neural Networks (COAFNN), which have two structural parameters: inputs to each layer of OR/AND neurons, and the number of OR/AND neurons. COAFNNs are tested in predication and classification problems.

Chapter 9 Introduction to Fuzzy Neural Networks

Fuzzy neural networks are developed based on the principle of artificial neural networks. The description of artificial neural networks is presented first, then fuzzy neurons are introduced, and the OR/AND neuron is explained in detail.

9.1 Overview of Fuzzy Neural Networks

9.1.1 Artificial Neural Network

An Artificial Neural Network (ANN) is an information-processing paradigm. Artificial neural networks are simulations of biological nervous systems, such as animal brains. They are composed of a large number of highly interconnected processing elements (neurons) working together to solve specific problems. An ANN is defined by the neurons and the connection between these neurons. These connections are measured by having values assigned to them (primarily floating point values), which are called weights. ANNs are trainable, that is they can learn by example. Through a learning process, an ANN changes the weights between two neurons according to a set of training data. The neuron, as the basic element to an ANN, usually has a number of connections that send information in/out of the neuron. The neuron processes the input information and produces the output information for other neurons in the ANN, or the network output to the outer system. The typical structure of the neuron looks like that in Figure 9.1.

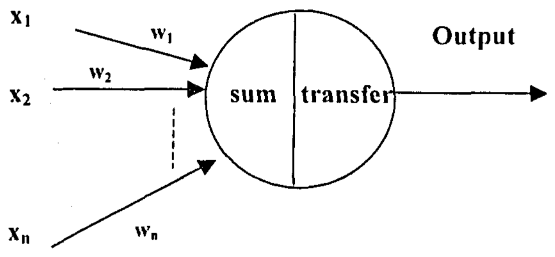


FIGURE 9.1 THE STRUCTURE OF THE NEURON

The neuron can be considered as having two parts. The first part has responsibility for the aggregation of the input information. In Figure 9.1, the sum is calculated by $sum = \sum_{i=1}^{n} x_i w_i$. If we denote $\mathbf{x}$ as the vector of the inputs and $\mathbf{w}$ as the vector of weights, that is $\mathbf{x} = [x_1, x_2, ..., x_n]$ and $\mathbf{w} = [w_1, w_2, ..., w_n]$, then the calculation of the sum can be expressed by $sum = \mathbf{x} \bullet \mathbf{w}'$. The second part of the neuron applies the transfer function (also called the activation function [4_8,4_15]) on the sum obtained in the first part, and generates the output of this neuron. There are many selections for the transfer function, such as the sigmoid function $f(x) = \frac{1}{1+e^{x}}$, the linear function $f(x) = x$, or the hard limit function $f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$. The topology of the neural network can be described by the number of layers and the number of neurons (or nodes) per layer. The types of layers include input, hidden, and output. The topology of a feed forward ANN, which has n inputs, 1 hidden layer containing m nodes, and just one node in the output layer, is shown in Figure 9.2.

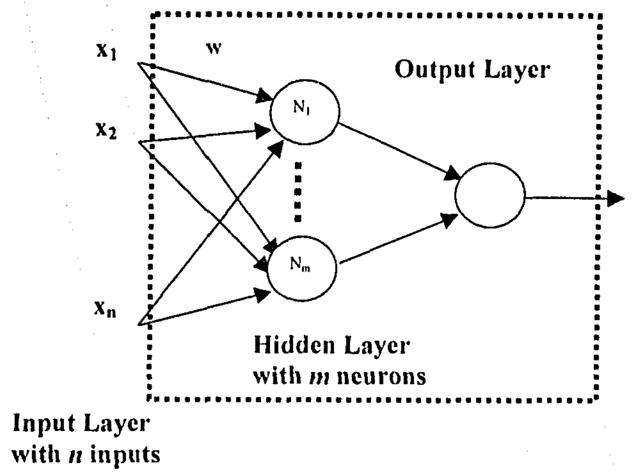


FIGURE 9.2 THE TOPOLOGY OF AN ANN

9.1.2 Fuzzy neurons

"One can think of neural networks as structure-free and fully distributed models. The distributivity contributes to profound learning capabilities, as the individual computing elements in the network are capable of adjusting their connections to carry out the best possible mapping. While this feature enhances learning, it makes it almost impossible to come up with a reasonable interpretation of the overall structure of the network worked out in terms of easily understood logical constructs (like "if-Then" statements, frames, etc.). [4_20] In order to make use of the profound learning capabilities of the ANN and gain the ability to interpret the knowledge in logical form, fuzzy neurons are introduced to the family of neural networks [4_17, 4_18, 4_19, 4_20]. There are two types of basic fuzzy neurons: AND and OR. Both types of neurons process information through the use of standard fuzzy set operations like AND, OR, and NOT. (See [4_22] for detailed information on these operators.) Let us briefly recall that by an OR neuron we mean a fuzzy neuron that achieves a logic mapping from $[0,1]^n$ into $[0,1]$ in the following format

$$y = \mathrm{OR}(\mathbf{x};\mathbf{w}) = \mathop{S}_{i=1}^{n}(x_i \; t \; w_i) \tag{1}$$

where x_i denotes the i-th input and w_i stands for the associated weight (connection), all of them assuming values in the unit interval. The aggregation operations are implemented using t- and s-norms (recall that t- and s-norms are models of logic operators of AND and OR, respectively). For logic values of 0 and 1 (Boolean logic), s- and t-norms result in standard AND and OR operators (logic intersection and union). Formally, by a t-norm we mean a two-argument operator mapping $[0,1]^2$ to $[0,1]$ such that it is (a) monotonic, (b) associative, and (c) commutative, and comes with boundary conditions as $0\ t\ a = 0$ and as $1\ t\ a = a$. The same properties hold for s-norms with the exception of the boundary conditions, which are spelled out as $0\ s\ a = a$ and as$1\ s\ a = 1$. Given the semantics of the logic operators, we can interpret (1) as a logic expression endowed with a collections of weights (connections) y = (x_1 and w_1) or (x_2 and w_2) or ... or (x_n and w_n). We use a convenient shorthand notation y = OR(x; w) by collecting all inputs and connections into two vectors, $x = [x_1\ x_2 \ldots x_n]^T$, $w = [w_1\ w_2 \ldots w_n]^T$. This representation helps emphasize the character of processing and underlines the available parametric flexibility of the module residing within its connections.

The AND neuron is governed by the expression

$$y = \mathrm{AND}(\mathbf{x};\mathbf{w}) = \mathop{T}_{i=1}^{n}(x_i\ s\ w_i) \qquad (2)$$

where in comparison with the previous construct, the order of aggregation operations has been reversed. Again, in terms of the abbreviated notation, we arrive at the expression, y = AND(x; w).

9.1.3 OR/AND neuron

The basic logic processing module to be used as a building module in the logic architectures of the network in this part of the research comes in the form of a so-called OR/AND neuron proposed by Hirota and Pedrycz [4_10][4_11] (refer also to [4_21] [4_22]). In essence, this processing unit can be regarded as a combination of several generic fuzzy logic neurons of AND and OR type [4_21] organized into a two-layer architecture. The OR/AND neuron arranges these two neurons in the form illustrated in Figure 9.3. In essence, the behaviors of the two neurons are aggregated (weighted) by using the OR neuron located in the output layer.

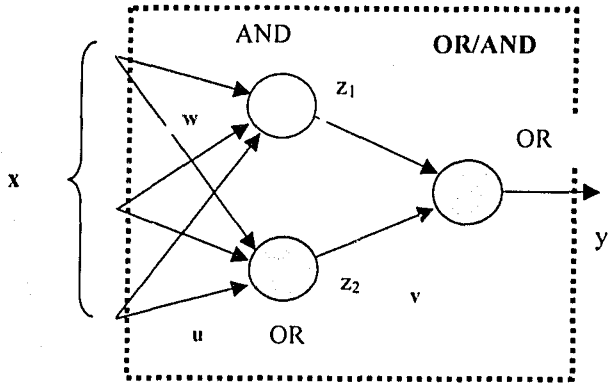


FIGURE 9.3 OR/AND NEURON: AN OVERALL ARCHITECTURE. ALSO SHOWN ARE THE CONNECTIONS OF THE CORRESPONDING NEURONS

Given this functional structure, the detailed formulas of the overall OR/AND neuron are as follows:

$$z_1 = \text{AND}(\mathbf{x}; \mathbf{w}); \quad z_2 = \text{OR}(\mathbf{x}; \mathbf{u}); \quad y = \text{OR}(\mathbf{z}; \mathbf{v}) \qquad (3)$$

where $v = [v_1\ v_2]^T$ and $z = [z_1\ z_2]^T$. The connections of the AND and OR neurons forming the first layer of this architecture are denoted by w and u, respectively. The overall aggregation of the logic behaviors is completed by the OR neuron located at the output layer of the OR/AND neuron. Here, the connections (v) play an important role; say, if v =[1 0], this leads to the "pure" type of the and-like aggregation. The two nonzero values of v produce a "mixed" type of logic aggregation, as both AND and OR neurons contribute to the output of the structure. The plots of the input-output relationships of the OR/AND neurons, regarded as a two-input single-output structure, are included in Figures 9.4-9.6. Here, we consider two collections of triangular norms and co-norms, and used several numeric combinations of the connections.

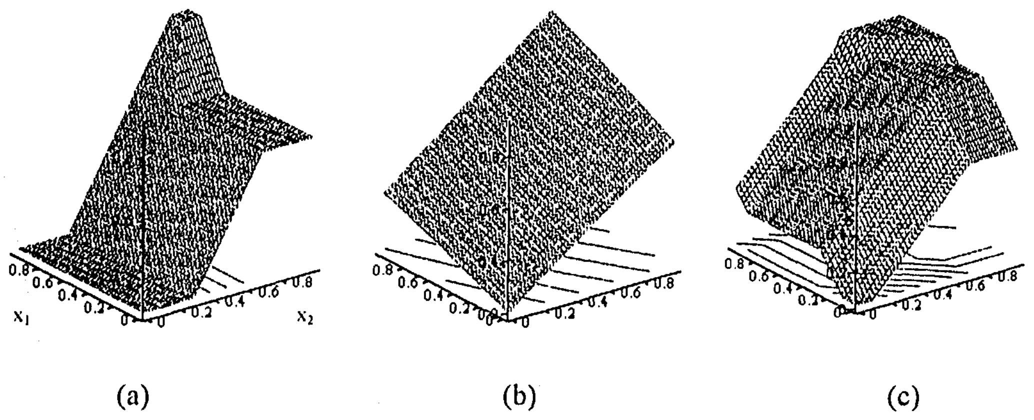


FIGURE 9.4. CHARACTERISTICS OF OR/AND NEURONS FOR SELECTED VALUES OF THE CONNECTIONS AND REALIZATIONS OF T- AND S-NORMS: (A) MIN AND MAX, (B) PRODUCT (ATB=AB) AND PROBABILISTIC SUM (ASB = A+B-AB), (C) LUKASIEWICZ CONNECTIVES (*AND* CONNECTIVE: ATB = MAX(0,A+B-1), *OR* CONNECTIVE: ASB = MIN(A+B,1)); **W** = [0.3 0.7], **U** = [0.4 0.9], **V** = [0.9 0.2]

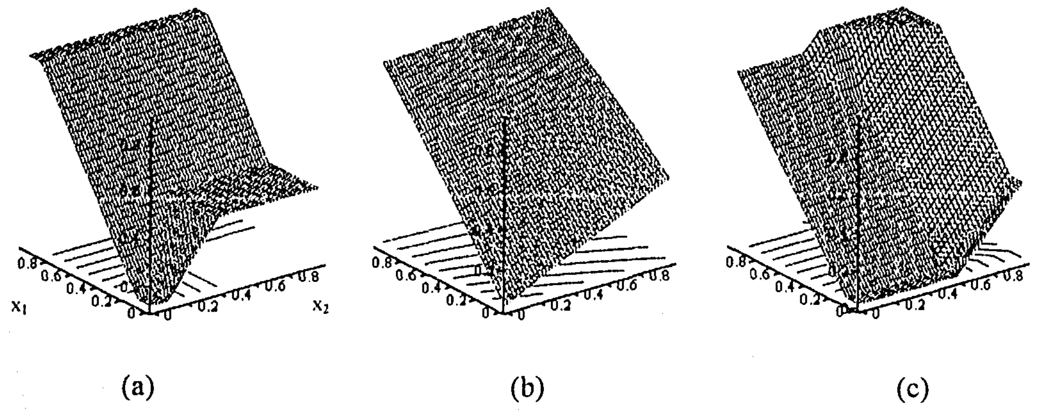


FIGURE 9.5. CHARACTERISTICS OF OR/AND NEURONS FOR SELECTED VALUES OF THE CONNECTIONS AND REALIZATIONS OF T- AND S-NORMS: (A) MIN AND MAX, (B) PRODUCT AND PROBABILISTIC SUM, (C) LUKASIEWICZ CONNECTIVES; **W** = [0.5 0.1], **U** = [0.4 0.9], **V** = [0.1 1.0]

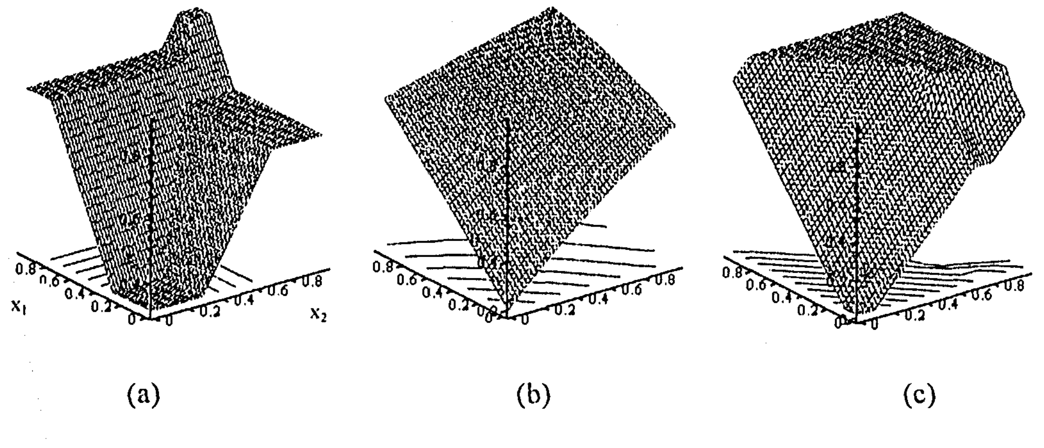


FIGURE 9.6. CHARACTERISTICS OF OR/AND NEURONS FOR SELECTED VALUES OF THE CONNECTIONS AND REALIZATIONS OF T- AND S-NORMS: (A) MIN AND MAX, (B) PRODUCT AND PROBABILISTIC SUM, (C) LUKASIEWICZ CONNECTIVES; **W** = [0.3 0.7], **U** = [0.4 0.9], **V** = [0.9 0.8]

Two observations become straightforward. First, the connections help us control the form of the input-output characteristics, which implies their important role in any learning activities:they deliver all necessary plasticity abilities of the network. Second, the use of various t-norms and s-norms is another mechanism one can use to adjust the network to conform to available experimental data. In particular, we note a difference between the use of the min - max tandem versus other combinations of t- and s-norms, whose usage usually gives rise to significantly smoother input-output dependencies. From the functional standpoint, we can accommodate complements of the inputs, that is, along with the (direct) inputs x, admit their complements $\overline{x}_i$, where $\overline{x}_i = 1 - x_i$. In this sense, given "n" original input variables, the total number of inputs becomes equal to "2n".

The evident advantage of the OR/AND neurons resides with their significant interpretability capabilities. Even though the neuron itself may appear somewhat complicated, its underlying logic expression is straightforward and readable. The two neurons positioned at the first layer of the OR/AND construct realize two generic logic operations of OR and AND. The results are combined OR-wise and this aggregation helps establish an interesting hybrid combination of the logic characteristics of the overall structure. Note that if $v_1 = 1$ and $v_2 = 0$ we end up with a "pure" and aggregation. The combination of $v_1 = 0$ and $v_2 = 1$ leads to the "pure" or-wise aggregation of the inputs. A whole spectrum of situations in-between the pure and and or aggregations is captured by the intermediate values of the connections of the OR neuron in the output layer. Interestingly, the hybrid style of logic connectives used in fuzzy sets has been around for a long time; one may refer to early experimental studies conducted by Zimmermann and Zysno [4_25], who introduced a broad family of compensative logic operators that share some properties of both and and or connectives. The same concept is realized here; the significant difference lies in the algorithmic realization of the operator where now we are provided with interesting learning abilities of the structure, essential to the adjustment of the properties of the neuron. Returning to the logic neurons in the first layer, we note that they have a straightforward interpretation. The OR neuron aggregates inputs or-wise with the weights (connections) that are essential to modeling the contribution of individual inputs to the output. The higher the value of the connection, w_i, the more significant the corresponding input is. In this sense, different inputs affect the output to a different degree, and this feature is of profound relevance for learning purposes. The situation is the opposite in the case of the AND neuron: here, higher values of the connections "mask" the inputs and therefore the highest impact (relevance) of the input happens when the input comes with the connection equal to zero. The impact of a certain input on the output could be quantified by computing the partial derivative of the output with regard to this specific input, say $\frac{\partial y}{\partial x_i}$. The detailed expressions depend upon the specific form of the t- and s-norms. For instance, in the case of the product and probabilistic sum we arrive at concise and interesting expressions of the form,

OR neuron

$$\frac{\partial y}{\partial x_i} = w_i[1 - A_i] \text{ where } A_i = \mathop{S}_{\substack{j=1 \\ j \neq i}}^{n} (x_j \text{ t } w_j) \tag{4}$$

AND neuron

$$\frac{\partial y}{\partial x_i} = B_i(1 - w_i) \text{ where } B_i = \mathop{T}_{\substack{j=1 \\ j \neq i}}^{n} (x_j \text{ s } w_j) \tag{5}$$

that are in essence linear relationships of w_i. The derivations of the above expressions dwell upon the specific type of the t- and s-norm that reads as a t b = ab and a s b = a+b-ab, where the arguments are in the unit interval. As an example, let us derive (3). Given the product realization of the t-norm, we have

$$y = \mathop{S}_{j=1}^{n} (x_j \text{ t } w_j) = [\mathop{S}_{\substack{j=1 \\ j \neq i}}^{n} (x_j \text{ t } w_j)]s(x_i t w_i) = [\mathop{S}_{\substack{j=1 \\ j \neq i}}^{n} (x_j w_j)]s(x_i w_i) \tag{6}$$

We now introduce the notation A_i to denote all but the ith component of the aggregation completed by the s-norm. Then, $y = A_i \, s(x_i w_i) = A_i + x_i w_i - A_i x_i w_I$, whose derivative taken with respect to x_i becomes equal to $w_i(1-A_i)$.

When dealing with "classic" artificial neurons, we are concerned with excitatory and inhibitory dependencies. In logic computing (as we do not allow for any negative connections), the aspect of inhibition is captured by admitting complements of the original inputs. In essence, this extension serves exactly the same purpose: when x_i decreases then $1-x_i$ increases and the inhibition effect takes place.

9.3 Input – output interfaces of fuzzy logic networks and optimization modes

So far the fuzzy network has been discussed in the context of logic mapping between the unit hypercubes, that is, $[0,1]^m$ to $[0,1]^n$. While these two somewhat abstract spaces could be directly encountered in some problems, in general we encounter physical variables that need to be interfaced with the network. A problem of this nature is commonly encountered in a variety of constructs involving fuzzy sets. The most straightforward solution is to assume a collection of reference fuzzy sets in each input and output variable, and treat them as conceptual entities that allow us to describe individual variables. Those could be standard semantic entities, such as small, medium, large, etc. From the computational standpoint we use them as nonlinear transformations (normalizations) of numeric inputs. Given a numeric input $z \in R$ and a collection of fuzzy sets defined in this space, say $\{A_1, A_2, \ldots, A_c\}$, the result of the transformation is a vector of membership grades $x = [A_1(z) \ldots A_c(z)]$, that is an element in the c-dimensional unit hypercube, $x \in [0,1]^c$. In this sense, each input and output datum comes with its manifestation in the space of reference fuzzy sets. When dealing with a data set transformed in this manner, we are concerned with the mapping between fuzzy sets rather than numeric values; the mapping is then realized in the form of the logic network. The optimization of the network is completed at the level of the vectors in the unit hypercubes rather than at the level of the original numeric data. The overall schematic flow of processing involving the interfacing of the network is displayed in Figure 9.7.

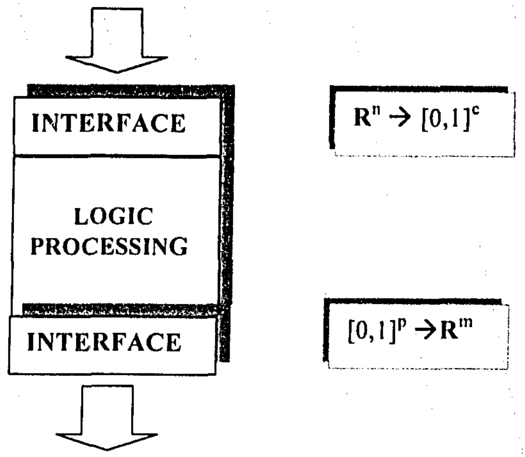


FIGURE 9.7. GENERAL SCHEMATICS OF INTERFACES OF FUZZY LOGIC NETWORKS: A FLOW OF COMPUTING AT THE PHYSICAL AND LOGICAL LEVEL

Obviously we could envision optimization at the level of the original data, that may include the interface itself, yet this may not be essential if we assume a certain type of membership functions for the fuzzy sets defined in the output space.

Chapter 10 Fuzzy Neural Network in Data Modeling

The fuzzy neural networks based on OR/AND neurons are created by placing the OR/AND neurons in the cascade manner. In this chapter, the structure is explained and a large number of experiments are carried out. In addition, the performance of cascade OR/AND neuron based fuzzy neural networks is analyzed.

10.1 Description of the Model

10.1.1 Architecture of the Model

The topology of the network depends/is based on the usage of OR/AND neurons as its basic computing nodes. The nodes are arranged into a form of the cascade (modular) structure as shown in Figure 10.1.

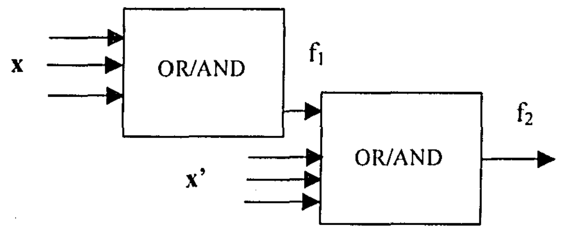


FIGURE 10.1. CASCADE TYPE OF NETWORK CONSTRUCTED WITH THE AID OF OR/AND NEURONS

Using this approach, the logic expression between input variables and the output is constructed stepwise. More specifically, denote the output of the first OR/AND neuron as f_1 = OR/AND (x; W_1) with W_1 combining all connections of the neurons in this structure. The output of the second OR/AND neuron is computed by considering other inputs x' and f_1 giving rise to f_2 = OR/AND ($[x' \ f_1]^T$, W_2) with $[x' \ f_1]^T$ forming the vector of inputs of this neuron. In the successive units of the network, the processing is carried out in the same manner. There are two important parameters we use to design the network: the number of layers (viz., the depth of the network), and the width of the individual module

(OR/AND neuron). Deeper structure results in fewer inputs at each layer, and conversely, more inputs to a single OR/AND node produce a network of less depth. The structure of the network controlled by these two parameters presents some obvious computational implications, and there are interesting insights into the logic description offered by the network. We can treat each OR/AND neuron as an implementation of a certain concept; the number of inputs to the neuron predetermines the complexity of the concept. The connections set up a certain balance between the or and and-like type of processing. They directly imply the nature of processing occurring at the subsequent phases of the network. In essence, at consecutive layers of the network, this concept is subject to further refinement. In some particular cases (see Figure 10.2), the concept is generalized by the or-wise inclusion of some additional variables or becomes specialized with the and-wise combination of other variables.

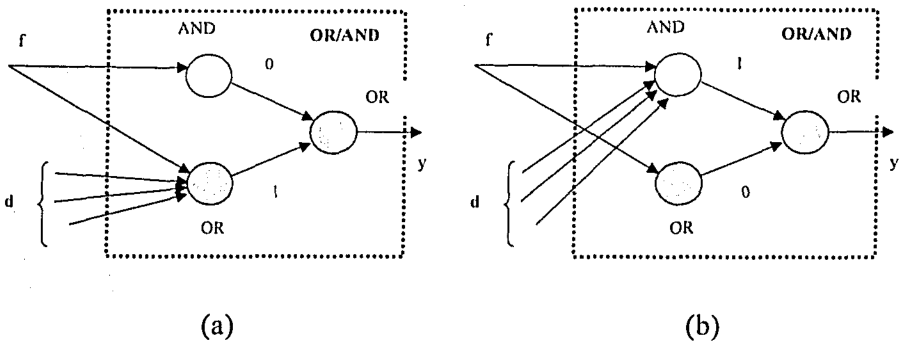


FIGURE 10.2. EXAMPLES OF REFINEMENT OF THE CONCEPT (F) DEVELOPED AT SOME LAYER OF THE NETWORK: (A) GENERALIZATION WITH THE AID OF ADDITIONAL VARIABLES (**D**), AND (B) SPECIALIZATION. SHOWN ARE TWO SUCCESSIVE MODULES (OR/AND NEURONS) OF THE NETWORK.

Interestingly the overall network achieves a decomposition of a fuzzy function, so f_1, f_2, and all successive outputs of the cascaded neurons build partial realizations of such a function. The decomposition of Boolean functions has been at the center of theoretical investigations since the fundamental work of Shannon and others, with the classic paper by Ashenhurst [4_1] (Refer also to [4_3]). The investigations in this study are in line with the generalizations of the decomposition and optimization results occurring in the realm of fuzzy functions [4_2][4_12][4_13][4_14][4_24].

To develop some preliminary idea of what the network can produce, let us consider the network shown in Figure 10.3. The interpretation of the network is straightforward. The only working knowledge we need at this point comes in the form of the boundary conditions of t- and s-norms. In particular, we know that the zero connections of the OR neuron eliminate the corresponding inputs. The connections of the AND neurons equal to 1 have the same elimination effect: all inputs associated with the connections set up to 1 do not exhibit any impact on the output of this AND neuron.

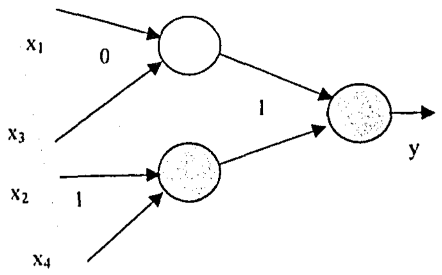


FIGURE 10.3. EXAMPLE OF THE FUZZY LOGIC NETWORK

Following these two simple interpretation rules, we arrive at the logic expression that could be read out directly from the network, y = OR (AND(x_1, x_3), OR(x_2, x_4)) = (x_1 and x_3) or (x_2 or x_3). This underlines the high level of transparency of the construct and its immediate interpretation.

10.1.2 Design aspects of the networks

The development of the proposed network involves two main categories of design decisions – (a) structural and (b) parametric. The diversity of the network may be very significant. Structural optimization is concerned with the characteristics of the network that result from selecting among its various structural options. The parameters of the network deal with the determination of optimal values of the connections of the individual OR/AND neurons. While this task is usually carried out through some gradient-based optimization techniques, the first group falls under the realm of genetic-based optimization, [4_4][4_16][4_23][4_27]. The choice of the optimization environment is influenced by a number of requirements. The decision of choosing genetic algorithms is well supported by two compelling arguments:

- We are interested in the structural development of the network. Being motivated by a quest for an interpretable description of data, one could envision that the number of variables could be substantially reduced without any visible deterioration of the approximation abilities of the network. The width and depth of the network are defined at the beginning of the design process (their values could be adjusted) and optimization at this end is not of concern – a simple enumeration is far more productive than involving them as a part of the optimization process. The selection of a subset of the variables is a combinatorial problem and here the role played by genetic optimization becomes essential. There are several design scenarios that, in essence, contribute to the increased dimensionality of the problem. In light of the inhibitory effect of some variables (obviously we do not know which of them could exhibit this phenomenon), we have to admit complements of all the variables. There are also two possible design alternatives in which we allow some variables to repeat (that is they could show up in the network more than once

-The optimization of the values of the parameters of the connections of the neurons could be a domain of gradient-based learning. The use of genetic optimization is still advantageous: the structure could exhibit significant depth that drastically hampers the efficiency of the method where a backpropagation of error has to be realized throughout the entire structure of the network. The greater the depth of the structure, the more difficulties arise, and the lower the efficiency of the ensuing learning.

10.1.3 GA applied in training

The genetic optimization aimed at the structural and parametric development of the network exploits Genetic Algorithms (GAs). In a nutshell, GA is a biologically inspired search method considering the principles of natural selection [4_4][4_9].

Having the genetic-based optimization task in mind, a chromosome consists of two parts (refer to Figure 10.4). The first part involves a binary representation of indexes of the input variables. The second part of the genotype is concerned with the connections of the neurons; it is coded as real numbers (floating point representation). This arrangement is very much in line with the phenotype-genotype transformations encountered in the literature, cf. [4_5] [4_6][4_7][4_9].

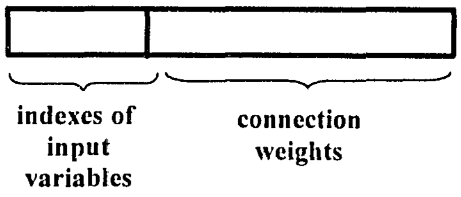


FIGURE 10.4. A SCHEMATIC VIEW OF A CHROMOSOME CONSISTING STRUCTURAL (INDEXES ARE CODED AS INTEGERS) AND PARAMETRIC (WEIGHTS ARE CODED AS REAL NUMBERS) DESCRIPTION OF THE NETWORK

The network is evaluated by means of some performance index expressing a measure of fit of the resulting structure. Given the training dataset stated in the standard format of input-output pairs, that is (x(k), target(k)), k=1,2,.., N, the sum of squared errors,

$$Q = \sum_{k=1}^{N} (\mathbf{x}(k) - \mathbf{target}(k))^T (\mathbf{x}(k) - \mathbf{target}(k))$$

is a commonly used measure expressing the fitness of the network. Given this, one could take the fitness function to assume the form $\frac{1}{1+Q}$. Here, genetic optimization leads to the maximization of the fitness function. In the assessment of the quality of the network, one could use any related index directly based upon Q, such as, for example, a known RMSE performance measure.

Once the evaluation and selection have been completed, some of the chromosomes are subject to crossover and mutation. In the case of a basic crossover, called a one-point crossover, the genotypes of randomly selected pairs are split into two parts and then recombined into a new pair of genotypes.

10.1.4 Interpretation aspects of the network

As already underlined, in the heart of fuzzy logic networks lies their interpretability. In essence, this means that we can take the optimized network and effortlessly produce its logic expression. As the neurons come with well-defined semantics, the transparency of the network is evident. There is however, a possibility of producing an even more condensed logic description by confining ourselves to the most "essential" part of the network. This is accomplished by reducing the weakest and least meaningful connections. This concept of pruning is guided by the properties of AND and OR neurons, and the underlying t- and s- norms, in particular.

10.1.5 A (λ,μ) pruning procedure and its realization

The pruning procedure introduced in this section is inherently associated with the (λ, μ) notation, hence the name of the pruning procedure. The underlying concept is straightforward and relates to the nature of the AND and OR neurons and their monotonicity with regard to the values of the corresponding connections. Owing to the expression for the OR neuron, we note that the higher the value of the connection, the more essential the input. Lower values of the connections could then be dropped (pruned). To formalize this effect, let us introduce a certain threshold λ with the values confined to the [0,1]:

$$w_\lambda = \begin{cases} w & \text{if } w \geq \lambda \\ 0 & \text{otherwise} \end{cases}$$

which returns an original value of the connection (w) if it exceeds or is equal to λ. Otherwise, if it is weak, we prune the connection and return a zero value. The higher the threshold, the more connections are eliminated from the original network. The operation reflects our previous observation that weaker connections are related to the variables that could be eliminated.

The threshold operation for the AND neurons is defined as follows:

$$w_\mu = \begin{cases} w & \text{if } w \leq \mu \\ 1 & \text{otherwise} \end{cases}$$

where we use a certain different threshold, say μ. Now, the connections whose values are above the threshold μ are pruned. By setting their values to 1 (corresponding to a complete masking effect), we eliminate the corresponding input. The lower the value of the threshold, the more radical the resulting pruning of the AND neurons. The combination of these two thresholds (λ, μ) serves as a general pruning mechanism. The choice of their values arises as a compromise between the reduced accuracy of the network and its increasing interpretability. More radical pruning (with the values of λ close to one and values of μ close to zero) leads to higher approximation error but at the same time results in more compact and therefore easily interpretable structure (its underlying logic expression). These two characteristics are in conflict. Possible optimization is very much user-oriented: we may wish move with further reductions of the network given its interpretation benefits from the reduced form of the model and accept somewhat higher values of the approximation error.

10.2 Experiment results

A comprehensive set of experiments is aimed at presenting the detailed design of the networks as well as quantifying their performance. Three categories of problems are considered. In the first one we use some Boolean data to visualize how Boolean functions can be effectively learned and interpreted. Next we move on to continuous data governed by expressions of multivalued logic. Finally we consider several representative examples coming from the Machine Learning Data Repository.

10.2.1 Synthetic binary data

Given a collection of input-output binary pairs {(x(k), y(k))}, k=1,2,...,N, $x(k)\in\{0,1\}^n$, $y(k)\in\{0,1\}^m$ we construct a logic network and interpret it so that a final result comes in the form of the logic expression. Having this in mind, we use all data for training purposes. From the design standpoint we allow for repeated variables and admit their complements as well. In the case of many outputs we design a family of networks with each of them having a single output.

Experiment 1 Here we consider Boolean data with n = 4 inputs and a single output, Table 10.1 (each column represent a single input-output pair)

x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
y	1	1	0	0	1	1	0	0	0	0	1	1	0	0	1	0

Table 10.1. Boolean data used in the experiment

The genetic optimization is run with the crossover rate of 0.7 and mutation rate of 0.05. The inputs are taken as original variables as well as their complements. Several combinations of the structural parameters of the network have been experimented with,. The scenario with 3 layers and 3 input OR/AND neurons is illustrated in Figure 10.5.

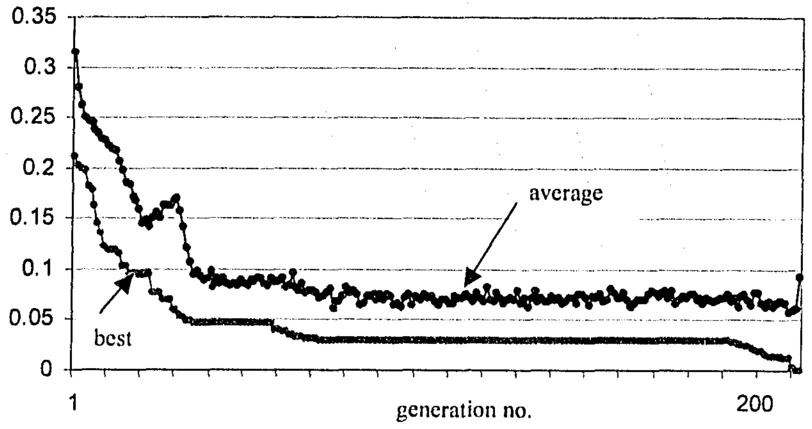


FIGURE 10.5. PERFORMANCE INDEX IN SUCCESSIVE GENERATIONS – BEST AND AVERAGE IN THE POPULATION (3-LAYER 3-INPUT STRUCTURE)

With this particular configuration the network produced a zero error. A smaller network (having 3 layers and 2 inputs) produced a nonzero value of error. Larger network gave rise to slower learning as visualized by the less substantial drop in the values of the performance index.

Let us discuss the 3-layer 3-input architecture of the network whose connections are equal to (we also include the indexes of the input variables)

layer-1			layer-2			layer-3		
	AND	OR		AND	OR		AND	OR
x_3	0.00	1.00	z_1	1.00	1.00	z_2	1.00	1.00
x_1	0.00	0.07	x_1	0.00	0.00	not(x_1)	0.00	0.00
not(x_2)	0.00	0.54	x_3	0.00	0.00	not(x_3)	0.00	0.00
			not(x_4)	0.00	0.00	x_4	1.00	0.00
OR	[1.00 0.00]		OR	[1.00 1.00]		OR	[1.00 1.00]	

Table 10.2 Network connections for the model

Further pruning of the connections (with the values of the threshold levels set up as λ = 0.9 and μ = 0.1, respectively) leads us to the reduced network as shown in Figure 10.6.

This reduction has not introduced any additional representation error (that is the error is still equal to zero).

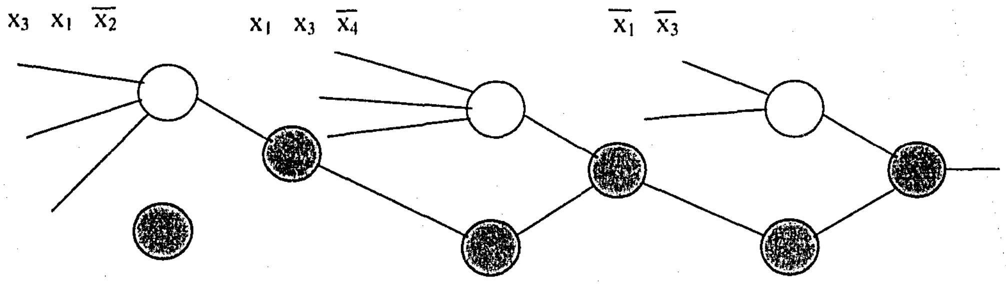

$x_3\ x_1\ \overline{x_2}$
$x_1\ x_3\ \overline{x_4}$
$\overline{x_1}\ \overline{x_3}$

FIGURE 10.6. THE TOPOLOGY OF THE NETWORK AFTER PRUNING (THE REMAINING MEANINGFUL CONNECTIONS ARE EQUAL TO 1 IN THE CASE OF OR NEURONS AND 0 FOR AND NEURONS).

A direct inspection of Figure 10.7 reveals that the network was able to simplify the Boolean data (the process which could have been easily verified by simplifying the data using a standard technique of Karnaugh maps, K-maps) to the following logic expression $x_3x_1\overline{x}_2 + x_1x_3\overline{x}_4 + \overline{x}_1\overline{x}_3$. Here for simplicity of notation we confined ourselves to the logic operators of and and or denoted here by * and + (which is a typical notation being used in digital systems). With regard to the simplified expression refer also to the corresponding K-map in which we have carried out the standard simplification by grouping the adjacent entries of the table.

	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	0	0	0	1
10	0	0	1	1

FIGURE 10.7. K-MAP AND THE RESULTING SIMPLIFICATION OF THE BOOLEAN DATA; HIGHLIGHTED ARE THE GROUPS OF THE LOGIC VARIABLES LEADING TO THE REDUCED EXPRESSION

There are some other interesting developments when pursuing different learning scenarios. When increasing the size of the population to 700 individuals we reached an ideal solution in less than 30 generations as illustrated in Figure 10.8.

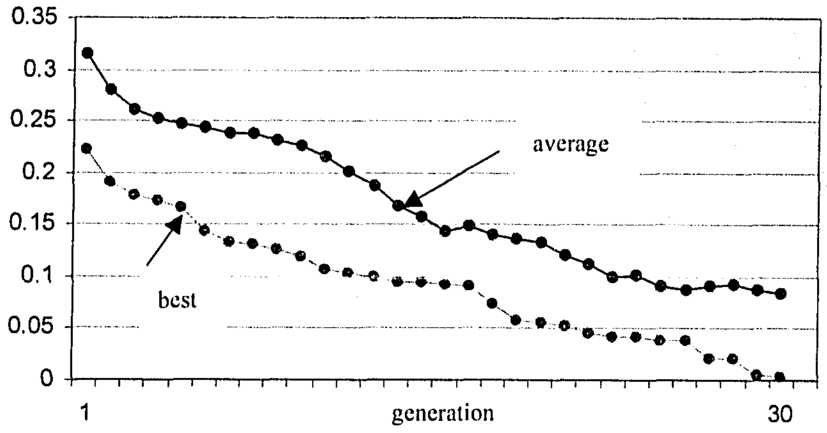
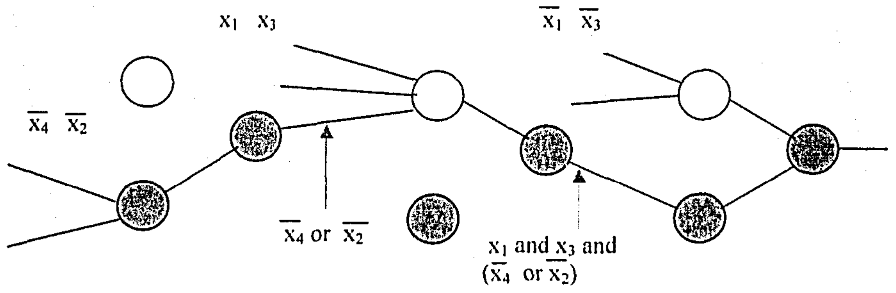


FIGURE 10.8. GENETIC OPTIMIZATION: THE BEST AVERAGE OF THE POPULATION IN SUCCESSIVE GENERATIONS

The topology of the network is now quite different from the architecture we obtained before; the first layer produces the or combination and this becomes aggregated and-wise at the next layer with some other variables, Figure 10.9.



x_1 x_3
$\overline{x}_1$ $\overline{x}_3$
$\overline{x}_4$ $\overline{x}_2$
$\overline{x}_4$ or $\overline{x}_2$
x_1 and x_3 and ($\overline{x}_4$ or $\overline{x}_2$)

FIGURE 10.9. THE PRUNED FUZZY LOGIC NETWORK; INCLUDED ARE ALSO PARTIAL RESULTS WHEN MOVING DOWN THE NETWORK

The logic expression comes now in the following form $x_1x_3(\overline{x}_2 + \overline{x}_4) + \overline{x}_1\overline{x}_3$, which in essence is equivalent to the one we produced when optimizing a different topology of the network. Opening the brackets we obtain $x_1x_3\overline{x}_2 + x_1x_3\overline{x}_4 + \overline{x}_1\overline{x}_3$ which is equivalent to the one derived earlier.

Larger networks lead to the zero value of the performance however its development takes more time. The 3-layer 4-input network required 50 generations and the use of the larger population of 1,500 individuals, see Figure 10.10. Interestingly enough, the network after pruning with λ =0.9 and μ =0.1 has resulted in the same topology as already portrayed in Figure 10.9.

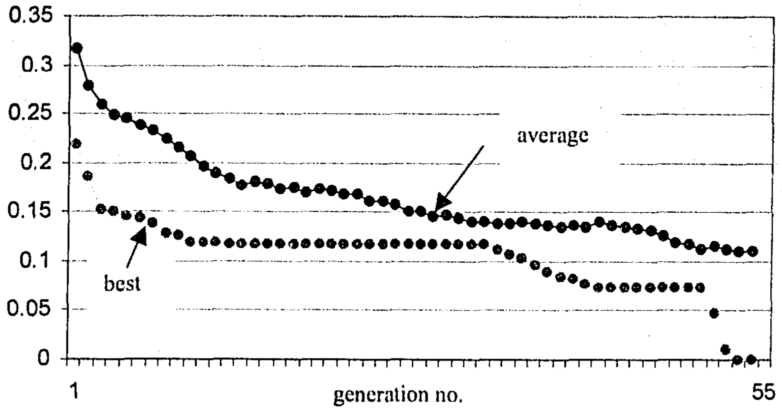


FIGURE 10.10. PERFORMANCE INDEX IN SUCCESSIVE GENERATIONS (BEST AND AVERAGE INDIVIDUAL IN THE POPULATION) OF NETWORK COMPOSED OF 3 LAYERS AND 4 INPUTS PER NEURON

Experiment 2. In this study, we are concerned with the continuous mapping from $[0,1]^2$ to [0,1] governed by the following logic expression

$$f(x_1, x_2, w_1, w_2, w_3) = [(w_1 t x_1) \, s \, (w_2 \, t(1-x_2))] \, t \, [(x_1 t x_2) \, s \, w_3]$$

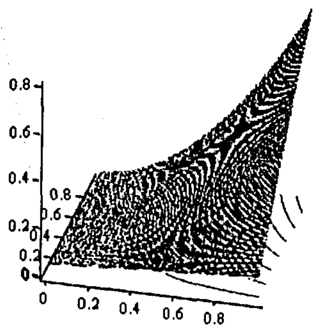


FIGURE 10.11. INPUT – OUTPUT CHARACTERISTICS OF THE LOGIC EXPRESSION

where the connections w_1, w_2, and w_3 are equal to 0.8, 0.4, and 0.1, respectively. The t- and s-norms are implemented as the product and probabilistic sum while the plot of the input – output characteristics for this specific configuration of the parameters is shown in Figure 10.11.

We have randomly generated 500 data points (here the inputs are drawn from a uniform distribution over the unit interval while the outputs are computed with the use of the logic expression given above). The experiment is completed with 60% of data used for training and the remaining 40% used for testing purposes. Several combinations of t- and s-norms have been experimented with to observe their impact on the performance of the network. The values of the parameters of the GA environment are typical and similar to those encountered in the literature: the crossover rate is 0.7 while the probability of mutation is 0.05. The size of the population is 100 and the optimization has been run for 100 generations (in most cases the convergence was noted earlier nevertheless we allow the method to run longer). Several topologies have been used along with the reported performance.

(a) Single layer with two inputs. The t- and s-norm is the product and probabilistic sum (so the logic operators are in agreement with those used in the generation of synthetic data). The resulting network comes with the RMSE of 0.001627 on the training set and 0.001505 on the testing set. As shown in Figure 10.12 the optimization requires very few generations, which is again not surprising considering the dimensionality of the problem and relatively large population of the individuals. In the sequel, the plot of the data versus the output of the network is given in Figure 10.13. The logic expression of the data reads as 0.63 t [(0.211 s x_1) t x_2] and involves a single AND neuron in the input layer of the OR/AND structure, Figure 10.14. Both inputs heavily contribute to the output (the connections of the AND neuron are equal to 0.211 and zero.

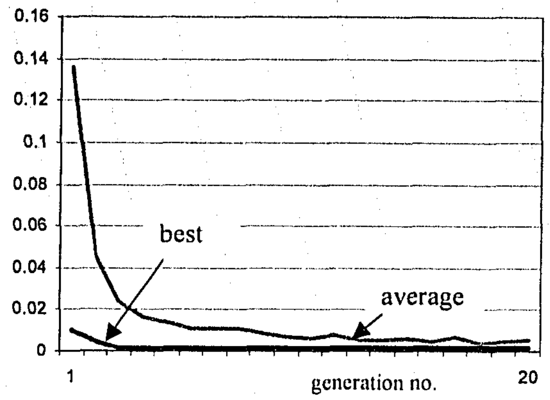


FIGURE 10.12. GENETIC OPTIMIZATION OF THE NETWORK: PERFORMANCE INDEX IN SUCCESSIVE GENERATIONS (BEST AND AVERAGE IN THE POPULATION)

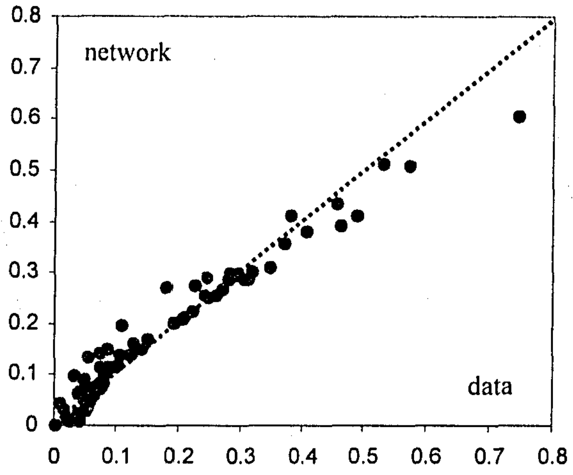


FIGURE 10.13. EXPERIMENTAL DATA VERSUS THE NETWORK (TRAINING SET)

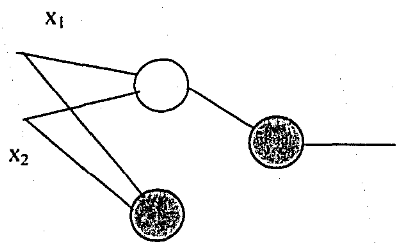
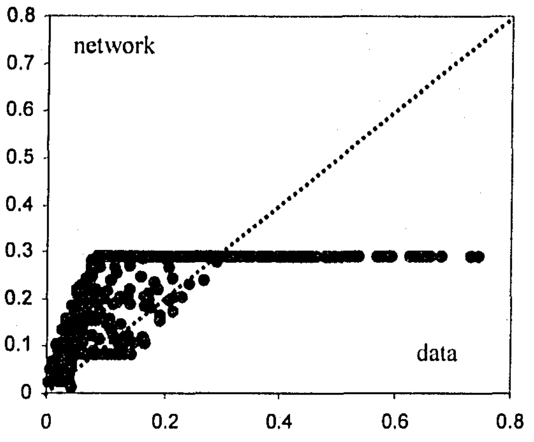


FIGURE 10.14. THE TOPOLOGY OF THE NETWORK; NOTE THAT THE OR NEURON COMES WITH A SINGLE CONNECTION (THE OTHER ONE IS EQUAL TO ZERO)

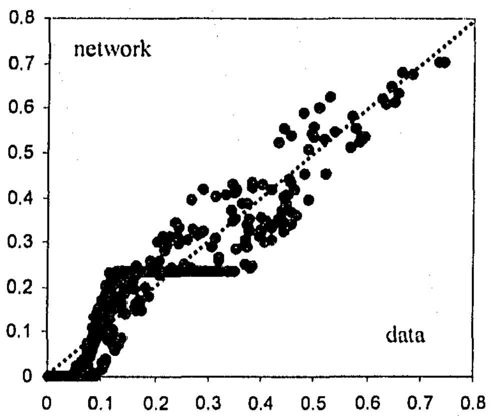
The use of other realizations of the logic connectives gives rise to higher values of the performance index as shown in Table 10.3; the min and max operators (that are appealing from the implementation perspective) produce a significant clipping effect, Figure 10.15.

t- and s-norm	RMSE (training)	RMSE (testing)
Min and Max	0.014527	0.015953
Lukasiewicz *and* and *or*	0.002602	0.003191

Table 10.3. Performance index for selected realizations of t- and s-norms



(a)



(b)

FIGURE 10.15. EXPERIMENTAL DATA VERSUS THE NETWORK (TRAINING SET): (A) MIN AND MAX OPERATIONS, (B) LUKASIEWICZ CONNECTIVES (ATB = MAX(0, A+B-1); ASB =MIN(1,A+B))

We carried out the design for the Lukasiewicz logic connectives; they lead to worse results (as illustrated in Figure 10.15(b), there is some effect of clipping) than the product – probabilistic pair but are far better than the results obtained for the network with the min and max operators. The poor performance of the min and max connectives could be attributed to their noninteractive character of processing the contributing arguments.

(b) three-input two-layer structure. In this structure we allowed both the complements of the input variables along with their duplicates in each layer (so this builds in an extra flexibility). The optimization resulted with the RMSE equal to $6*10^{-6}$ for both training and testing set (to get there it took about 20 generations). Practically, the results fully match the data as this becomes visible in this figure 10.16.

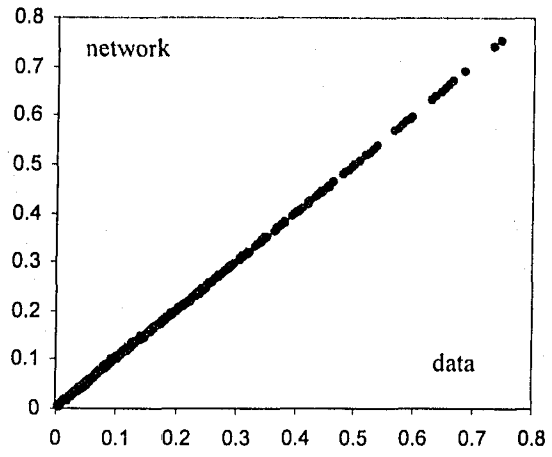


FIGURE 10.16. DATA VERSUS THE OUTPUT OF THE NETWORK

By inspecting the values of the connections while monitoring the values of the approximation error associated with the reduced network, it is obvious that the network is very much excessive and could be easily pruned. Experimenting with various values of the threshold levels (λ and μ), Table 10.4, one can use the values of 0.4 (λ) and 0.6 (μ) as a reasonable compromise: the RMSE went up but also a significant portion of connections has been eliminated.

OR neurons (λ)	0.4	0.5	0.6
AND neurons (μ)			
0.7	$1.85*10^{-4}$, 10	$1.85*10^{-4}$, 10	$1.85*10^{-4}$, 10
0.6	$8.69*10^{-4}$, 11	$8.69*10^{-4}$, 11	$8.69*10^{-4}$, 11
0.5	$8.69*10^{-4}$, 11	$8.69*10^{-4}$, 11	$8.69*10^{-4}$, 11
0.4	$8.69*10^{-4}$, 11	$8.69*10^{-4}$, 11	$8.69*10^{-4}$, 11

Table 10.4. RMSE and number of eliminated connections for selected values of thresholds; note that the original number of the connections is equal to 18.

Noticeably, the changes of the threshold levels affect the approximation abilities (reflected in the values of the RMSE), number of pruned connections and (what is not directly quantified in the table above), the readability of the resulting logic expression. Two combinations of the thresholds are: at the combination of (0.4, 0.7) we have eliminated 10 connections and ended up with the RMSE that is over 4 times lower when in case we modify the thresholds and eliminate 11 connections. The interpretability of the structure could be another essential criterion: a careful inspection of the pruned network reveals that at the second layer of the structure the first variable repeats itself and enters the network with different connections. The second combination of the connections eliminates this effect.

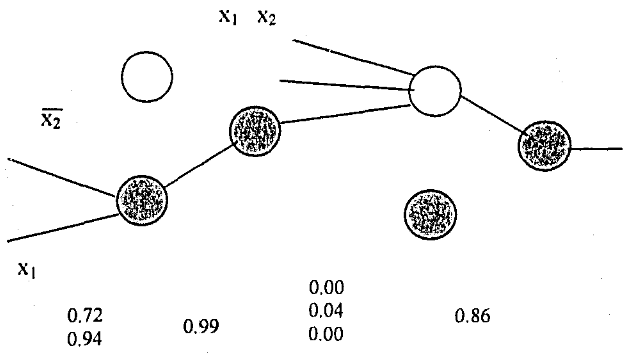


FIGURE 10.17. NETWORK AFTER PRUNING (λ =0.4; μ =0.6), THE VALUES OF THE CONNECTIONS ARE SHOWN BELOW

10.2.2 Machine Learning datasets

A collection of the ensuing experiments concerns Machine Learning data. As we are dealing with continuous data, the network interfaces with the external world through some interface as discussed in Section 6. The optimization is realized at the internal logic based level. We are concerned with continuous as well as discrete data; with the latter being typical for classification problems.

<u>**Experiment 3**</u> In the analysis of Boston housing data we are interested in the construction of the logic relationship between information granules of price of real estate (that is quantified as low, medium, and high) and the corresponding fuzzy sets defined in the input space. The number of fuzzy sets for the inputs is equal to three as well. In the

case of discrete variables assuming several discrete values, we consider a 1-out of – n decoding scheme. Because of this arrangement, we develop three separate networks where each of them deals with the corresponding linguistic granule of the price of real estate. This means that we end up with a logic description of real estate of low, medium and high price. As usual, the data set is split into the 60-40 training and testing combination. The selection of the structure of the network is not that obvious as in the previous case where we are after the zero value of the performance index (in essence, in the previous experiment we were concerned with the representation problem). In the case of the approximation nature of the problem, the development of the network could be conveniently monitored by reporting performance on the training and testing sets so this helps us achieve a tradeoff between sound approximation and generalization abilities. In the experiments, the GA was run for 60-200 generations with a size of the population being equal to 500. Under these conditions the results stabilized and larger values of these parameters have not contributed to any significant changes in the performance of the network. The results of learning are reported in a series of tables, Table 10.5(a)-(c). There are two important design parameters of the network such as its depth (the number of layers) and width (the number of inputs entering each processing module). We anticipate that by choosing their values, we could achieve an optimal performance of the overall network. Likewise we may envision some tradeoffs between its depth and width. The performance of the network could be immediately characterized by means of the RMSE achieved on the training set. While compelling in general, one could be faced with the ultimate danger of memorization and reduced generalization abilities. To monitor the impact of the network's parameters on the performance of the network we could record the values of the ratio (κ) of the values of the RMSE on the training and testing (or validation) set, $\kappa = \frac{Q_{test}}{Q_{train}}$. Typically, this ratio would take values greater than 1, which is obvious given somewhat weaker performance over the testing set. In essence, high values of the ratio become indicative of the poor generalization abilities of the network. Obviously, one needs to look at both measures otherwise we would be favoring models that are highly consistent across the data but equally poor on the testing and training data. The results are summarized in Table 10.5. Those concern the realization of t- and s-norms as the product and probabilistic sum.

Layers / Inputs	1		2		3		4		5		6	
	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)
2	0.105	2.2	0.095	2.38	0.089	2.46	0.096	2.65	0.088	2.5	0.097	2.33
3	0.099	*2.19*	0.089	2.49	0.094	2.78	0.088	3.14	0.093	2.88	0.089	2.65
4	0.088	2.67	0.089	2.84	0.103	2.29	**0.090**	**2.29**	0.084	2.92	0.094	2.67
5	0.094	2.38	0.090	2.99	0.086	2.91	0.088	2.91	0.088	2.68	0.086	2.81
6	0.090	2.48	0.089	2.67	0.089	2.66	0.084	2.97	0.095	2.37	0.087	2.64
7	*0.080*	3.08	0.087	2.53	0.096	2.8	0.089	2.65	0.099	2.42	0.093	2.64

(a) *low* median price of real estate

Layers / Inputs	1		2		3		4		5		6	
	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test trai n (κ)	Train (Q)	test train (κ)
2	0.156	1.85	0.146	1.81	0.147	1.81	0.145	1.81	0.153	1.85	0.155	1.85
3	0.153	1.82	0.145	1.9	0.148	1.87	**0.143**	*1.64*	0.147	1.81	0.143	1.82
4	0.152	1.81	0.149	1.88	0.153	1.87	0.140	1.94	0.147	1.75	0.148	1.71
5	0.153	1.85	0.150	1.89	*0.139*	1.89	0.154	1.85	0.152	1.88	0.152	1.93
6	0.153	1.88	0.152	1.84	0.149	2.02	0.152	1.86	0.152	1.91	0.153	1.9
7	0.154	1.79	0.149	1.95	0.154	1.87	0.148	1.93	0.157	1.83	0.157	1.81

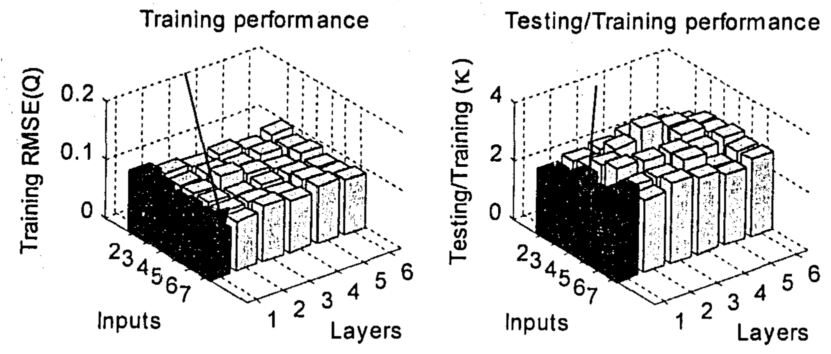
(b) *medium* median price of real estate

Layers / Inputs	1		2		3		4		5		6	
	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)	Train (Q)	test train (κ)
2	0.102	1.59	0.098	1.66	0.100	1.66	0.099	1.63	0.100	1.62	0.111	1.59
3	0.106	1.62	**0.098**	**1.62**	0.106	1.62	0.102	1.7	0.112	1.62	0.099	1.74
4	0.102	*1.58*	*0.094*	5.87	0.103	1.68	0.106	1.66	0.106	1.63	0.111	1.63
5	0.102	1.58	0.100	1.6	0.105	1.67	0.101	1.72	0.102	1.58	0.106	1.67
6	0.106	1.68	0.102	1.58	0.101	1.64	0.099	1.6	0.102	1.58	0.099	1.65
7	0.106	1.61	0.106	6.75	0.107	1.65	0.105	1.71	0.101	1.6	0.108	1.69

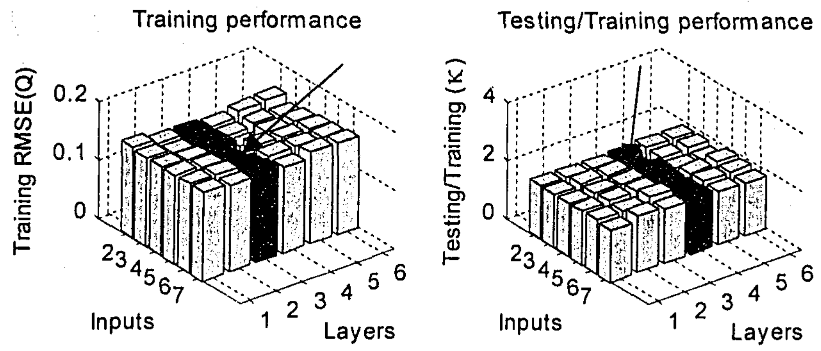
(c) *high* median price of real estate

Table 10.5. Quantification of the network for the training set and the values of ratio κ for different combinations of the number of layers and width of the node. Note that there are three networks each for the specific linguistic term defined in the output (*low*, *medium*, and *high*) , say (a)-(c), respectively. The boldface entries represent the optimal configurations of the networks *which are chosen based on the RMSE values of both training and testing data*

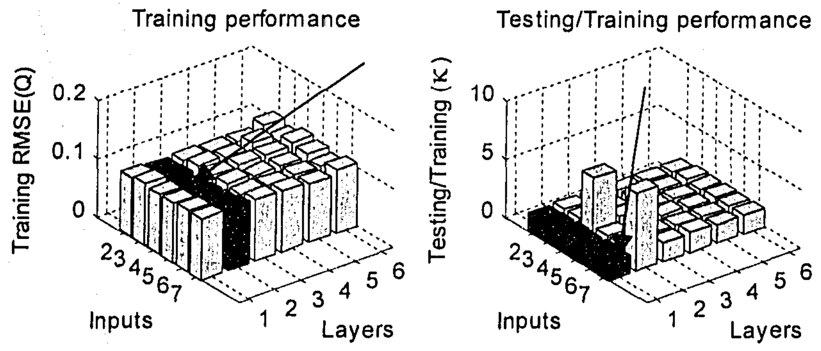
The above results are also shown in a form of bar graphs, Figure 10.18 that help illustrate some general tendencies. The lowest values for Q and are indicated by the arrows in Figure 10.18 and the italic numbers in Table 10.5.



(a) *low* median price of real estate



(b) *medium* median price of real estate



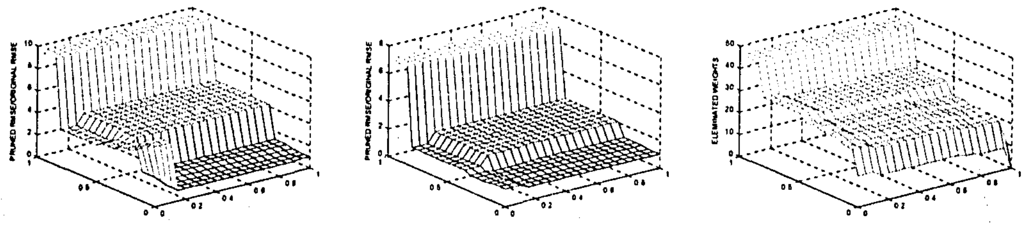
(c) *high* median price of real estate

FIGURE 10.18. BAR GRAPH VISUALIZING THE PERFORMANCE (FOR TRAINING SET) AND RATIO κ FOR THREE DIFFERENT NETWORKS DESCRIBING *LOW*, *MEDIUM*, AND *HIGH*

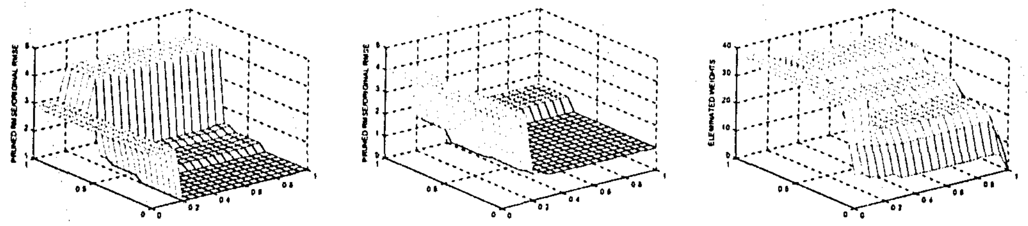
MEDIAN PRICE OF REAL ESTATE (A) – (C), RESPECTIVELY (I IS SHORT FOR INPUT L IS SHORT FOR LAYER)

Interestingly enough, we note a certain general tendency. For the first network the best results are produced when the depth of the network is medium and the width of each OR/AND is medium. For the network governing the medium values of output the best results are obtained for medium depth of the network that comes with small width of the units. Finally, for the third network we end up with somewhat low values of width and depth.

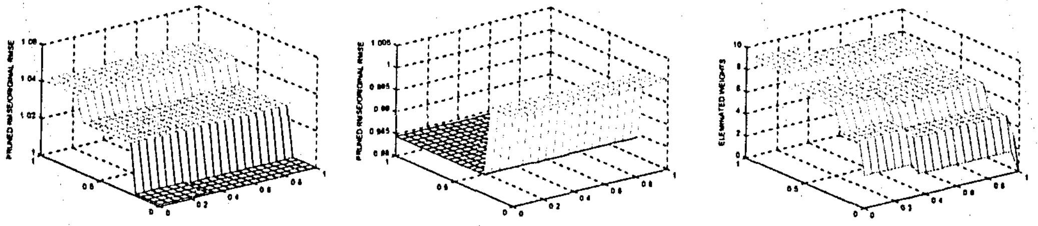
We selected optimal configurations of the networks (layer 4, input 4 for low MEDV, layer 4, input 3 for medium MEDV and layer 2, input 3 for high MEDV) and we carried out their pruning ending up with results shown in Figure 10.19. These graphs show the impact on the threshold levels of the neurons (μ and λ) on the relative performance of the reduced network. Thus we use a ratio of the performance index (Q) of the pruned network versus the original one; the values higher than 1 point at the deterioration in the mapping properties because of the increased interpretability. Here, an interesting observation concerns a tradeoff between the accuracy and the reduction in the size of the network (as the weakest connections become eliminated its interpretability increases). There are quite large regions of the (λ,μ) plane where the changes of the threshold do not impact the values of the performance index. The impact coming from the threshold applied to the AND neurons is not the same as the one OR neurons are subjected to.



(a) *low* median price of real estate



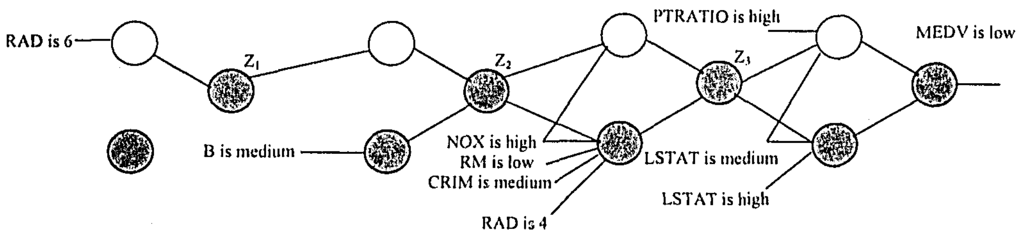
(b) *medium* median price of real estate



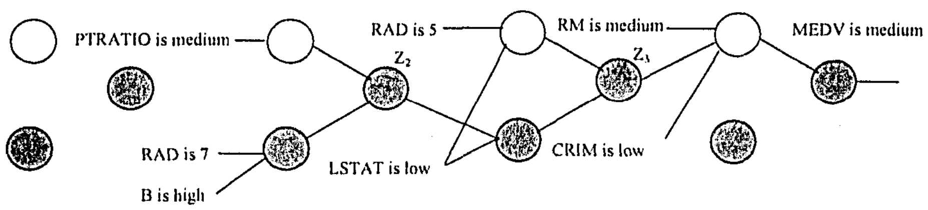
(c) *high* median price of real estate

FIGURE 10.19. THE EFFECT OF PRUNING PROCESS OF THE NETWORKS SHOWING IMPACT OF THE THRESHOLD VALUES ON THE PERFORMANCE OF THE NETWORK *FOR TRAINING DATA (LEFT COLUMN), FOR TESTING DATA (MIDDLE COLUMN) AND ON THE STRUCTURE OF THE NETWORKS (RIGHT COLUMN)*

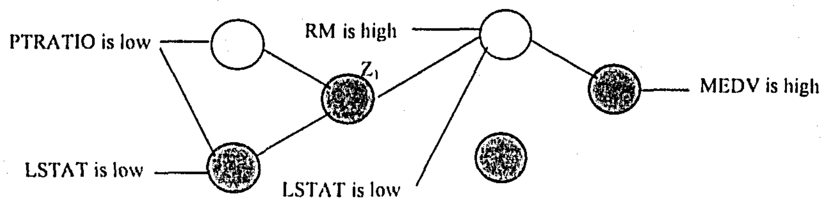
Considering the threshold values of λ and μ are equal to (0.1, 0.3, 1) and (0.2, 0.3, 0) (these two combinations led to a sound accuracy-transparency compromise), we end up with a linguistic description (protocol) of the data. This description is produced directly from the reduced networks shown in Figure 10.20. All pertinent abbreviations of the variables standing in the rules see part 0.



a) *low* median price of real estate (λ=0.1 and μ=0.2, Q=0.089767, κ=2.30)



b) *medium* median price of real estate (λ=0.3 and μ=0.3, Q=0.144053, κ=1.63)



c) *high* median price of real estate (λ=1 and μ=0, Q=0.099679, κ=1.58)

FIGURE 10.20. NETWORKS RESULTING AFTER THE PRUNING PROCESS

Median price	Rules

low	IF $\{\{Z3\}_{0.00}$ AND $\{$LSTAT is *medium*$\}_{0.11}$ AND $\{$PTRATIO is *high*$\}_{0.13}\}_{0.81}$ OR $\{\{Z3\}_{0.14}$ OR $\{$LSTAT is *medium*$\}_{0.32}$ OR $\{$LSTAT is *high*$\}_{1.00}\}_{0.81}$ THEN MEDV is *low*
	Z3=$\{\{Z2\}_{0.00}$ AND $\{$NOX is *high*$\}_{0.00}\}_{1.00}$ OR $\{\{Z2\}_{1.00}$ OR $\{$RM is *low*$\}_{1.00}$ OR $\{$NOX is *high*$\}_{1.00}$ OR $\{$CRIM is *medium*$\}_{1.00}$ OR $\{$RAD is 4$\}_{0.18}\}_{1.00}$
	Z2=$\{\{Z1\}_{0.00}\}_{1.00}$ OR $\{\{$B is *medium*$\}_{1.00}\}_{1.00}$
	Z1=$\{\{$RAD is 6$\}_{0.00}\}_{0.46}$

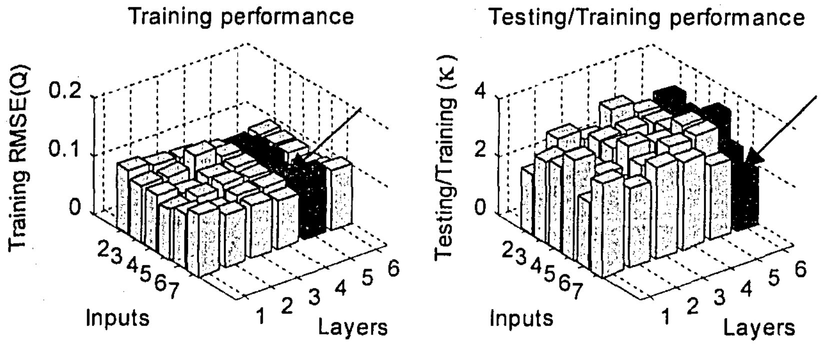
medium	IF $\{\{Z3\}_{0.15}$ AND $\{$RM is *medium*$\}_{0.00}$ AND $\{$CRIM is *low*$\}_{0.29}\}_{1.00}$ THEN MEDV is *medium*
	Z3=$\{\{$LSTAT is *low*$\}_{0.00}$ AND $\{$RAD is 5$\}_{0.00}\}_{0.61}$ OR $\{\{Z2\}_{1.00}$ OR $\{$LSTAT is *low*$\}_{1.00}\}_{1.00}$
	Z2=$\{\{$PTRATIO is *medium*$\}_{0.09}\}_{0.71}$ OR $\{\{$RAD is 7$\}_{1.00}$ OR $\{$B is *high*$\}_{0.64}\}_{0.38}$

high	IF $\{\{Z1\}_{0.00}$ AND $\{$LSTAT is *low*$\}_{0.00}$ AND $\{$RM is *high*$\}_{0.00}\}_{1.00}$ THEN MEDV is *high*
	Z1=$\{\{$PTRATIO is *low*$\}_{0.00}\}_{1.00}$ OR $\{\{$LSTAT is *low*$\}_{1.00}$ OR $\{$PTRATIO is *low*$\}_{1.00}\}_{1.00}$

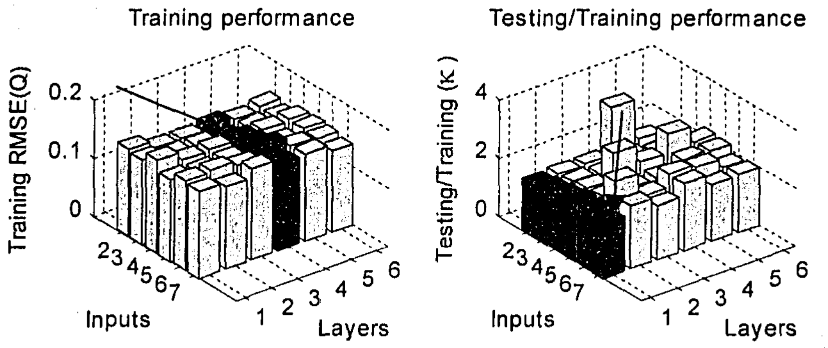
Table 10.6. Collection of rules describing Boston housing data

We can now fully appreciate the expressive power of the network; the relationships are quite revealing. In many cases they reflect our intuitive feel as to the nature of the relationships between the price of real estate and the essential factors that impact it. More importantly, though, the model quantifies these essential relationships and tells which variables exhibit essential impact on the median price of real estate. We also see how intermediate variables are constructed and what they entail.

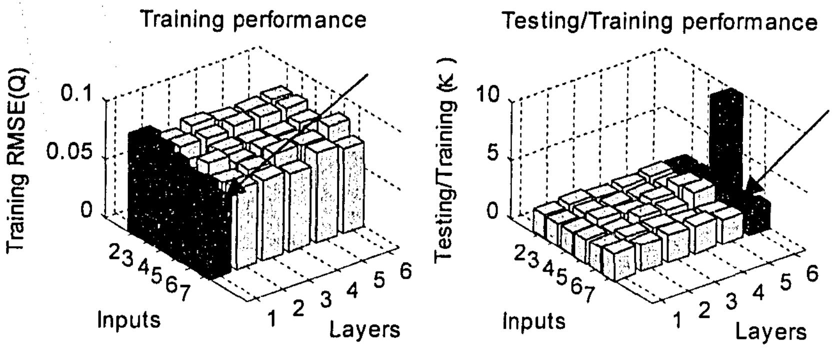
The experiment was repeated for several other combinations of t- and s-norms; the remaining values of the parameters of the network and the overall setup of the optimization environment were the same as before. The findings are quite similar. While the numeric performance of the network varies from case to case, the changes of the performance index with respect to the key parameters of the network remain the same, see Figures 10.21-10.22 for more details. Essentially, we note that there always exists a certain tradeoff between he depth of the network and the width of the individual nodes. For the best option, we get the following values (the results are reported for three networks in a combined manner): Q = (0.082420,0.135074,0.072684) κ =(2.45,1.83,2.07) (Lukasiewicz connectives) and Q = (0.116752, 0.153483, 0.112303) κ =(2.24,1.72,1.55) (min and max operators).



a) Median price of real estate is *low*

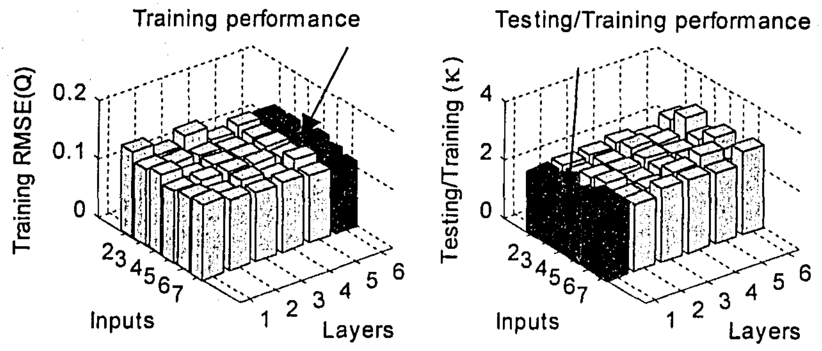


b) Median price of real estate is *medium*

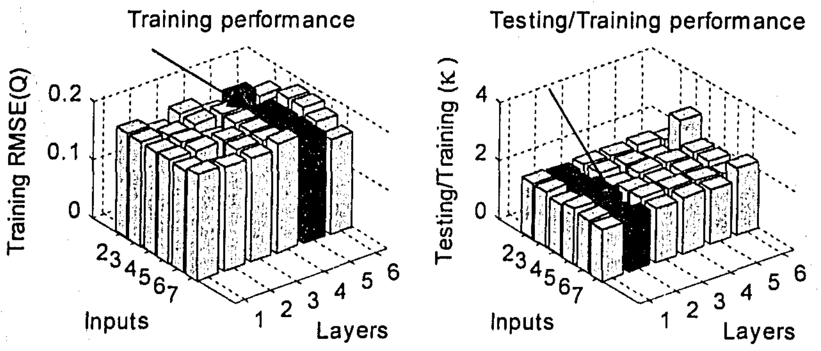


c) Median price of real estate is *high*

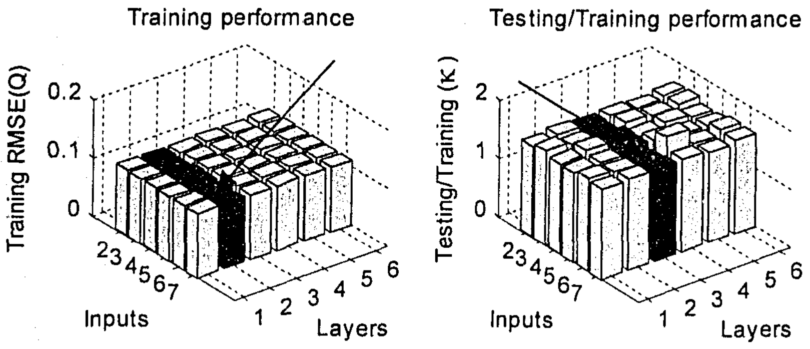
FIGURE 10.21. PERFORMANCE INDEX REGARDED AS A FUNCTION OF DEPTH AND WIDTH OF THE NETWORK FOR THE LUKASIEWICZ LOGIC CONNECTIVES (I IS SHORT FOR INPUT L IS SHORT FOR LAYER)



a) Median price of real estate is *low*



b) Median price of real estate is *medium*



c) Median price of real estate is *high*

FIGURE 10.22. PERFORMANCE INDEX AS A FUNCTION OF DEPTH AND WIDTH OF THE NETWORK FOR MIN AND MAX (I IS SHORT FOR INPUT L IS SHORT FOR LAYER)

Through the series of detailed experiments, we found that the optimal configuration of the networks based on Lukasiewicz logic connectives turn out to offer the best performance, which is followed by the networks exploiting product and probabilistic sum. The min and max operations produced the least favorable results. The intuitive feel as to the contribution of the variables is fully reflective in the topology of the network. When talking about real estate of low price, the network points out at the three highly indicative attributes of ISTAT, PTRATIO and NOX; all of them assume high or medium values. All three of them are reflective of the standard of schools (crowded classes), environmental conditions (nitric oxides concentration) and high percentage of lower status of the population. When we move to the medium price range, some of the previous attributes are in place with the number of rooms coming into the expression. For the high median price of real estate, the high number of rooms is the common attribute identified by all networks. The rules derived with the connectives of product and probabilistic sum put stress on the combination of low LSTAT and low PTRATIO. In the case of Lukasiewicz connectives, "AGE," "TAX," and "DIS" are taken into consideration.

Considering the best architecture of the network and completing its pruning, we arrive at the linguistic description of the data as shown in Table 10.7 (the use of the Lukasiewicz connectives) and Table 10.8 (here we considered the min and max operations).

Median price	Rules
low (width: 6 , depth: 2, λ=0.4 and μ=0.7)	IF {{LSTAT is *high*}$_{0.68}$ AND {PTRATIO is *high*}$_{0.58}$ AND {LSTAT is *medium*}$_{0.70}$ AND {NOX is *high*}$_{0.63}$}$_{0.98}$ OR {{RM is *medium*}$_{0.85}$ OR {RM is *low*}$_{0.95}$ OR {LSTAT is *medium*}$_{0.41}$}$_{0.31}$ THEN MEDV is *low*

medium
(width: 2, depth: 3, λ=0.2 and μ=0.7)
	IF {{Z2}$_{0.21}$ AND {RM is *medium*}$_{0.20}$ AND {LSTAT is *low*}$_{0.63}$}$_{0.60}$ OR {{Z2}$_{1.00}$ OR {RM is *medium*}$_{0.69}$}$_{0.28}$ THEN MEDV is *medium*
	Z2={{Z1}$_{0.69}$}$_{1.00}$ OR {{Z1}$_{0.26}$ OR {PTRATIO is *medium*}$_{0.96}$}$_{0.60}$
	Z1={{TAX is *high*}$_{0.00}$ AND {RAD is *2*}$_{0.65}$}$_{0.32}$

high
(width: 4, depth: 3, λ=0.45 and μ=0.7)
	IF {{Z2}$_{0.57}$ AND {RM is *high*}$_{0.00}$}$_{0.94}$ OR {{LSTAT is *low*}$_{0.58}$ OR {TAX is *high*}$_{0.46}$}$_{0.56}$ THEN MEDV is high
	Z2={{Z1}$_{0.82}$ OR {AGE is *low*}$_{1.00}$ OR {RAD is *5*}$_{1.00}$}$_{1.00}$
	Z1={{AGE is *medium*}$_{0.34}$ AND {DIS is *low*}$_{0.40}$ AND {INDUS is *high*}$_{0.31}$}$_{0.97}$ OR {{AGE is *medium*}$_{0.64}$ OR {RAD is *3*}$_{0.84}$}$_{1.00}$

Table 10.7. Rules describing Boston housing data (the use of Lukasiewicz connectives)

Median price	Rules

low
(width: 6 , depth: 2, λ=0.3 and μ=0.2)
	IF {{Z1}$_{0.10}$ AND {DIS is *low*}$_{0.17}$ AND {LSTAT is *medium*}$_{0.03}$}$_{0.97}$ THEN MEDV is *low*
	Z1={{B is *low*}$_{0.79}$ OR {LSTAT is *high*}$_{0.86}$ OR {RAD is *24*}$_{0.34}$ OR {NOX is *high*}$_{0.48}$ OR {PTRATIO is *high*}$_{0.48}$}$_{0.62}$

medium
(width: 3, depth: 5, λ=0.4 and μ=0.6)
	IF {{Z4}$_{0.52}$ AND {RM is *medium*}$_{0.01}$}$_{0.90}$ THEN MEDV is *medium*
	Z4={{Z3}$_{1.00}$ OR {LSTAT is *low*}$_{1.00}$ OR {RAD is *5*}$_{0.68}$}$_{0.97}$
	Z3={{Z2}$_{0.85}$ OR {PTRATIO is *medium*}$_{0.92}$}$_{0.83}$
	Z2={{AGE is *medium*}$_{0.85}$ OR {RM is *low*}$_{0.87}$}$_{0.92}$

high (width: 6, depth: 3, λ=1 and μ=0.9)	IF {{RAD is *5*}$_{0.85}$ AND {B is *high*}$_{0.00}$ AND {RM is *high*}$_{0.00}$}$_{1.00}$ THEN MEDV is *high*

Table 10.8. Rules describing Boston housing data (the use of min and max operators)

Experiment 4. We design fuzzy logic networks for several commonly used datasets such as Auto-mpg, Machine CPU and Abalone. The development process is the same as before. In what follows we briefly report the key results, which lend themselves to the linguistic characterization of the data. Again, for each case the best architecture was searched for. Afterwards we pruned it by choosing sound values of the threshold levels (again the issue of reaching a legitimate compromise was discussed earlier and we followed the same development path). The quantification of the performance of the corresponding networks is summarized in Table 10.9-10.10.

Data set	Q (training set)	κ	Threshold layers and number of connections pruned	Comments
Auto-mpg	*low*: 0.097652 *Med*: 0.110055 *high*: 0.027749	1.62 2.54 8.66	Layer:5 Input:2 Layer:4 Input:4 Layer:3 Input:2	The depth and size of the network depends upon the output of the network; the one corresponding to *low* output requires the largest depth

	low: 0.124896
	Med: 0.188752
	high: 0.061673
	After pruning
		1.03
		1.69
		3.29
			λ=0.4, μ=0.8
			17 eliminated
			λ=0.6, μ=0
			42 eliminated
			λ=0.3, μ=0
			13 eliminated	

Machine CPU	*low*:0.057367 *Med*: 0.063419 *high*:0.002449	1.55 1.68 20.24	Layer:3 Input:3 Layer:2 Input:6 Layer:4 Input:6	Here the largest network is required for the network describing *high* performance of the CPU

	low: 0.057367
	Med: 0.063390
	high: 0.002411
	After pruning
		1.55
		1.68
		20.55
			λ=0.6, μ=0.95
			12 eliminated
			λ=0.2, μ=0
			13 eliminated
			λ=0.1, μ=0
			36 eliminated	

Abalone	*low*: 0.137430 *Med*: 0.152729 *high*: 0.069635	0.93 0.90 0.56	Layer:5 Input:3 Layer:6 Input:5 Layer:5 Input:2	It was observed that the network describing *medium* output required a large network. The two others concerning *low* and *high* output led to networks of large depth and small width.

	low: 0.137430
	Med: 0.152729
	high: 0.069635
	After pruning
		0.93
		0.90
		0.56
			λ=0.05, μ=0
			35 eliminated
			λ=0.2, μ=0.3
			68 eliminated
			λ=0.2, μ=0.9
			18 eliminated	

Table 10.9. Quantification of the optimized networks along with the values of their main parameters. The number of linguistic terms in all networks was set up to 3

Data set	Q (training set)	κ	Threshold layers and number of connections pruned	Comments
Auto-mpg	*low*: 0.097954 *Med*: 0.107995 *high*: 0.024739	1.75 2.43 10.47	Layer:5 Input:4 Layer:5 Input:4 Layer:5 Input:3	It is worth noting that the performance of the network depends quite substantially on its depth and to a lesser extent on the width of the nodes (number of its inputs)

	low: 0.102036
	Med: 0.118444
	high: 0.061573
	After pruning
		1.56
		2.13
		3.72
			λ=0.35, μ=0.4
			41 eliminated
			λ=0.4, μ=0
			36 eliminated
			λ=0.2, μ=0
			45 eliminated	

Machine CPU	*low*: 0.059506 *Med*: 0.050577 *high*: 0.001732	1.38 1.64 6.63	Layer:5 Input:2 Layer:4 Input:3 Layer:5 Input:5	The network characterizing *high* performance of the CPU was the most demanding and has resulted in the large number of layers and number of inputs. In two other networks we require lower number of inputs
	low: 0.059506 *Med*: 0.050904 *high*: 0.003412 After pruning	1.38 1.61 3.73	$\lambda=0.2$, $\mu=0.1$ 22 eliminated $\lambda=0.2$, $\mu=0.1$ 28 eliminated $\lambda=1$, $\mu=0.1$ 65 eliminated	
Abalone	*low*: 0.137931 *Med*: 0.154198 *high*: 0.069685	0.94 0.92 0.59	Layer:3 Input:6 Layer:5 Input:5 Layer:4 Input:4	The network used to model *medium* output is the largest. The one pertaining to the *low* output is quite shallow but comes with a significant depth (the number of inputs to each node)
	low: 0.146374 *Med*: 0.153408 *high*: 0.074152 After pruning	0.91 0.91 0.53	$\lambda=0.2$, $\mu=0$ 26 eliminated $\lambda=0.3$, $\mu=0.1$ 34 eliminated $\lambda=0.5$, $\mu=0.05$ 24 eliminated	

Table 10.10. Quantification of the optimized networks along with the values of their main parameters. The number of linguistic terms describing variables was set to 5

The rules involving 3 and 5 linguistic terms are shown in Table 10.11-10.11.

Fuel Efficiency	Rules

low
($\lambda=0.4$, $\mu=0.8$)
	IF {{Z4}$_{0.00}$ AND {DISPL is *medium*}$_{0.37}$}$_{0.84}$ OR {{Z4}$_{1.00}$ OR {DISPL is *medium*}$_{0.47}$ OR {CYLIN is *8*}$_{0.47}$}$_{0.79}$ THEN Fuel Efficiency is *low*
	Z4={{Z3}$_{0.00}$ AND {Weight is *high*}$_{0.73}$}$_{1.00}$ OR {{Weight is *high*}$_{1.00}$}$_{1.00}$
	Z3={{HorsP is *high*}$_{0.00}$}$_{1.00}$ OR {{Z2}$_{1.00}$ OR {HorsP is *high*}$_{0.73}$}$_{0.80}$
	Z2={{CYLIN is *3*}$_{0.00}$}$_{1.00}$ OR {{CYLIN is *3*}$_{1.00}$}$_{1.00}$

medium ($\lambda=0.6$, $\mu=0$)	IF {{Weight is *low*}$_{0.95}$ OR {DISPL is *low*}$_{0.79}$ OR {HorsP is *low*}$_{0.68}$}$_{0.87}$ THEN Fuel Efficiency is *medium*

high
($\lambda=0.3$, $\mu=0$)
	IF {{Z2}$_{0.00}$ AND {HorsP is *low*}$_{0.00}$ AND {Weight is *low*}$_{0.00}$}$_{0.39}$ THEN Fuel Efficiency is *high*
	Z2={{ModelY is *82*}$_{0.00}$}$_{0.54}$ OR {{ModelY is *82*}$_{0.91}$ OR {Origin is *3*}$_{0.36}$}$_{1.00}$

Table 10.11. Linguistic description of data for Auto-mpg; the number of linguistic terms is equal to 3

Fuel Efficiency	Rules

low
($\lambda=0.35$, $\mu=0.4$)
	IF {{Weight is *high*}$_{0.25}$ AND {Weight is *very_high*}$_{0.37}$}$_{0.52}$ OR {{Z4}$_{0.57}$ OR {Weight is *medium*}$_{0.35}$ OR {Weight is *high*}$_{0.62}$ OR {Weight is *very_high*}$_{1.00}$ OR {CYLIN is *8*}$_{0.65}$}$_{0.88}$ THEN Fuel Efficiency is low
	Z4={{CYLIN is *3*}$_{0.00}$ AND {ModelY is *73*}$_{0.00}$}$_{0.54}$ OR {{CYLIN is *3*}$_{1.00}$ OR {ModelY is *72*}$_{0.35}$ OR {CYLIN is *6*}$_{0.75}$ OR {ModelY is *73*}$_{0.51}$}$_{0.77}$

medium (λ=0.4, μ=0)	IF {{Z4}$_{0.91}$ OR {HorsP is *low*}$_{0.50}$ OR {CYLIN is *4*}$_{1.00}$}$_{0.88}$ THEN Fuel Efficiency is *medium* Z4={{DISPL is *low*}$_{0.00}$}$_{0.76}$ OR {{Z3}$_{0.79}$ OR {DISPL is *low*}$_{0.58}$ OR {Weight is *low*}$_{0.64}$ OR {ModelY is *81*}$_{0.79}$}$_{0.70}$ Z3={{Weight is *medium*}$_{0.00}$}$_{0.50}$ OR {{Z2}$_{1.00}$ OR {ModelY is *76*}$_{1.00}$ OR {CYLIN is *8*}$_{0.84}$}$_{0.50}$ Z2={{Z1}$_{1.00}$ OR {ModelY is *77*}$_{1.00}$}$_{1.00}$ Z1={{ACCE is *very_low*}$_{0.00}$}$_{1.00}$
high (λ=0.2, μ=0)	IF {{DISPL is *very_low*}$_{0.00}$ AND {Weight is *very_low*}$_{0.00}$}$_{0.44}$ THEN Fuel Efficiency is *high*

Table 10.12. Linguistic description of the Auto-mpg data; the number of linguistic terms (that are quantified as *very_low*, *low*, *medium*, *high* and *very_high*) is equal to 5

The results for the CPU Performance Data are reported in the same manner, see Table 10.13-10.14

Published relative performance	Rules
low (λ=0.6, μ=0.95)	IF {{Z2}$_{0.91}$ AND {CACHE is *low*}$_{0.00}$ AND {MMIN is *low*}$_{0.02}$}$_{0.98}$ OR {{Z2}$_{1.00}$ OR {MMAX is *low*}$_{1.00}$ OR {MMIN is *low*}$_{0.69}$}$_{0.68}$ THEN PRP is low Z2={{Z1}$_{0.00}$ AND {MMAX is *medium*}$_{0.00}$}$_{1.00}$ OR {{Z1}$_{0.65}$ OR {MMAX is *low*}$_{1.00}$}$_{1.00}$ Z1={{MMAX is *low*}$_{0.00}$}$_{1.00}$
medium (λ=0.2, μ=0)	IF {{Z1}$_{0.00}$ AND {MMIN is *medium*}$_{0.00}$}$_{0.77}$ OR {{CACHE is *medium*}$_{1.00}$ OR {MMAX is *high*}$_{1.00}$ OR {CACH is *high*}$_{1.00}$ OR {MMIN is *high*}$_{1.00}$}$_{0.28}$ THEN PRP is *medium* Z1={{MYCT is *low*}$_{0.00}$ AND {MMIN is *medium*}$_{0.00}$ AND {MMAX is *medium*}$_{0.00}$}$_{0.71}$ OR {{CHMAX is *high*}$_{1.00}$ OR {MMAX is *high*}$_{1.00}$ OR {MMIN is *low*}$_{0.88}$ OR {MMAX is *medium*}$_{0.92}$}$_{1.00}$
high (λ=0.1, μ=0)	IF {{Z3}$_{0.00}$ AND {MMAX is *high*}$_{0.00}$}$_{1.00}$ THEN PRP is *high* Z3={{Z2}$_{0.13}$ OR {CACH is *high*}$_{1.00}$ OR {MMIN is *high*}$_{1.00}$ OR {MYCT is *high*}$_{0.50}$ OR {CHMAX is *high*}$_{0.50}$ OR {MMAX is *low*}$_{0.50}$}$_{1.00}$ Z2={{Z1}$_{0.00}$ AND {MMAX is *medium*}$_{0.00}$}$_{0.16}$ OR {{Z1}$_{0.34}$ OR {CHMIN is *low*}$_{0.50}$ OR {CACH is *high*}$_{0.90}$ OR {MYCT is *low*}$_{1.00}$ OR {MMAX is *medium*}$_{0.50}$ OR {CACH is *medium*}$_{0.63}$ OR {MMIN is *low*}$_{0.21}$}$_{0.42}$ Z1={{CACH is *medium*}$_{0.39}$ OR {MMAX is *low*}$_{0.19}$ OR {CACH is *high*}$_{1.00}$ OR {CHMAX is *medium*}$_{0.18}$}$_{1.00}$

Table 10.13. Linguistic description of Machine CPU data; the number of linguistic terms is equal to 3

Published relative performance	Rules

low
(λ=0.2 and
μ=0.1)
	IF {{CACH is *very_low*}$_{0.05}$ AND {MMIN is *very_low*}$_{0.00}$}$_{0.99}$ OR {{Z4}$_{0.99}$ OR {CACH is *very_low*}$_{0.63}$ OR {MMIN is *very_low*}$_{0.22}$}$_{0.93}$ THEN PRP is *low*
	
	Z4={{Z3}$_{1.00}$ OR {MMIN is *low*}$_{0.34}$ OR {MMAX is *low*}$_{1.00}$}$_{0.86}$
	
	Z3={{MMAX is *low*}0.00}$_{1.00}$ OR {{MMAX is *low*}$_{1.00}$ OR {CHMIN is *very_low*}$_{0.56}$}$_{1.00}$

medium
(λ=0.2 and
μ=0.1)
	IF {{MMIN is *medium*}$_{0.09}$}$_{0.77}$ OR {{Z3}$_{1.00}$ OR {CHMIN is *low*}$_{0.22}$ OR {MMIN is *medium*}$_{1.00}$ OR {MMAX is *medium*}$_{0.53}$}$_{0.51}$ THEN PRP is *medium*
	
	Z3={{MMIN is *very_low*}$_{0.00}$ AND {CACH is *medium*}$_{0.00}$}$_{1.00}$

high (λ=1, μ=0.1)	IF {{MMAX is *very_high*}$_{0.00}$ AND {CACH is *medium*}$_{0.04}$}$_{1.00}$ THEN PRP is *high*

Table 10.14. Linguistic description of Machine CPU data; the number of linguistic terms (that are quantified as *very_low*, *low*, *medium*, *high* and *very_high*) was set up to 5

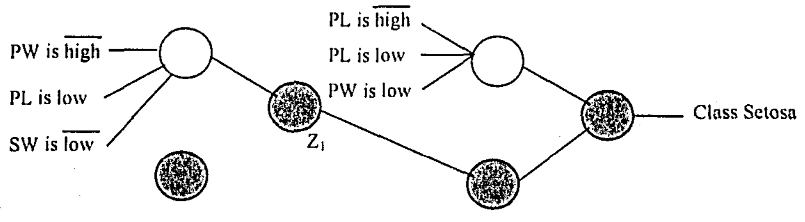
There is an interesting effect of more detailed granulation of variables (the increased number of fuzzy sets defined therein) on the performance of the model. In general, the accuracy of the network increases (an obvious tendency one could have easily expected), refer to Table 10.15.

Data set	3	4	5	6	7
Auto-mpg	Q =(0.097652 0.110055 0.027749) κ = (1.62, 2.54, 8.66) Optimal parameters: (Layer:5Inp:2 Layer: 4Inp: 4 Layer:3Inp:2)	Q=(0.089163 0.110095 0.026796) κ = (1.74, 2.50, 9.50) Optimal parameters: (Layer:4Inp:3 Layer:2np:2 Layer:2np:5	Q=(0.097954 0.107995 0.024739) κ = (1.75, 2.43, 10.47) Optimal parameters: (Layer:5Inp:4 Layer:5Inp:4 Layer:5Inp:3)	Q=(0.103933 0.113212 0.021932) κ = (1.73, 2.37, 12.11) Optimal parameters: (Layer:5Inp:3 Layer:4Inp:2 Layer:4Inp:2)	Q=(0.109918 0.104494 0.024900) κ = (1.51, 2.62, 10.69) Optimal parameters: (Layer:3Inp:3 Layer:3Inp:4 Layer:3Inp:3)
Machine CPU	Q=(0.057367, 0.063419, 0.002449) κ=(1.55, 1.68, 20.24) Optimal parameters: (Layer:3Inp:3 Layer:2Inp:6 Layer:4Inp:6)	Q=(0.064684 0.053470 0.001732) κ = (1.37, 1.60, 64.34) Optimal parameters: (Layer:5Inp:5 Layer:2Inp: 6 Layer:1Inp:5)	Q=(0.059506 0.050577 0.001732) κ = (1.38, 1.64, 6.63) Optimal parameters: (Layer:5Inp:2 Layer:4Inp:3 Layer:5Inp:5)	Q=(0.062562 0.052820 0.002000) κ = (1.52, 1.74, 55.88) Optimal parameters: (Layer:5Inp:5 Layer:3Inp:4 Layer:1Inp:5)	Q=(0.051478 0.055263 0.001414) κ = (1.89, 1.59, 79.49) Optimal parameters: (Layer:2Inp:4 Layer:2Inp:4 Layer:2Inp:4)

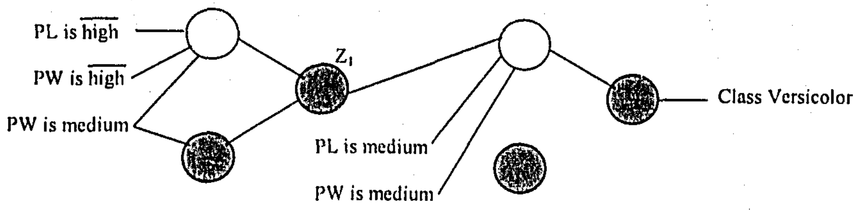
Abalone	Q=(0.137430 0.152729 0.069635) κ=(0.93, 0.90, 0.56) Optimal parameters: (Layer:5Inp3: Layer:6Inp:5 Layer:5Inp:2)	Q=(0.137543 0.149760 0.069642) κ = (0.93, 0.92, 0.59) Optimal parameters: (Layer:4Inp:6 Layer:6Inp: 7 Layer:4Inp:3)	Q=(0.137931 0.154198 0.069685) κ = (0.94, 0.92, 0.59) Optimal parameters: (Layer:3Inp:6 Layer:5Inp: 5 Layer:4Inp:4)	Q=(0.139492 0.154803 0.069484) κ = (0.93, 0.91, 0.60) Optimal parameters: (Layer:6Inp:4 Layer:4Inp:5 Layer:5Inp:2)	Q=(0.139989 0.156831 0.070385) κ = (0.91, 0.91, 0.62) Optimal parameters: (Layer:5Inp:3 Layer:4Inp:4 Layer:5Inp:3)

Table 10.15. Performance of the network for the optimal configuration of the structural parameters (depth and width). No pruning has been completed (Inp is short for Input)

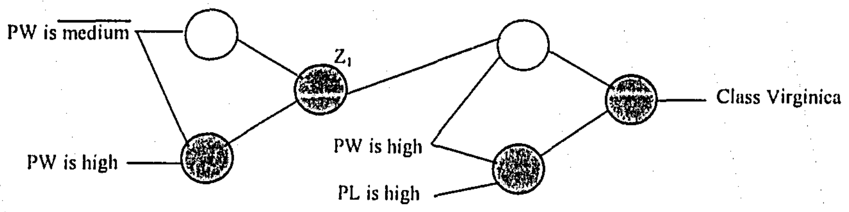
Experiment 5. Here we elaborate on the use of the logic network to a classification problem such as the well-known IRIS data. The data set consists of 150 patterns, 50 in each class. The data set has been divided into the training and testing set with the standard split of 60%-40%. The classification variable is coded following the 1-out-of-n scheme. All input attributes are coded with the use of three Gaussian fuzzy sets (whose semantics pertains to linguistic terms such as low, medium, and high). Through a series of experiments we determined that the optimal network comes with a width of 3 where the depth is equal to 2. For this configuration, the performance is quantified as Q = (0.025866,0.1511328,0.163378) and κ=(1.10,1.37,1.37), respectively. The threshold mechanism leads to the structure displayed in Figure 10.23. It is noticeable that the reduction of the network has not produced any increase of the classification error. In this case the classification rate on the training data set achieved 98.9% (that is 89 out of 90 correctly classified patterns) while on the testing data set the result is 93.3% (56 out of 60 correctly classified patterns).


PW is $\overline{\text{high}}$
PL is low
SW is $\overline{\text{low}}$
PL is $\overline{\text{high}}$
PL is low
PW is low
Z_1
Class Setosa

(a)


PL is $\overline{\text{high}}$
PW is $\overline{\text{high}}$
PW is medium
Z_1
PL is medium
PW is medium
Class Versicolor

(b)



(c)

FIGURE 10.23. PRUNED LOGIC NETWORK FOR IRIS DATA: (A) SETOSA (λ=1 AND μ=0); (B) VERSICOLOR (λ=0.9 AND μ=0); (C) VIRGINICA (λ=0.6 AND μ=0)

Equivalently, the classification rules derived from the pruned networks are included in Table 10.16.

Plant	Rules

Setosa	IF {{PL is *low*}$_{0.00}$ AND {NOT PL is *high*}$_{0.00}$ AND {PW is *low*}$_{0.00}$}$_{1.00}$ OR {{Z1}$_{1.00}$}$_{1.00}$ THEN Output is *setosa*
	Z1={{NOT PW is *high*}$_{0.00}$ AND {PL is *low*}$_{0.00}$ AND {NOT SW is *low*}$_{0.00}$}$_{1.00}$

Versicolor	IF {{Z1}$_{0.00}$ AND {PL is *medium*}$_{0.00}$ AND {PW is *medium*}$_{0.00}$}$_{1.00}$ THEN Output is *versicolor*
	Z1={{NOT PL is *high*}$_{0.00}$ AND {NOT PW is *high*}$_{0.00}$ AND {PW is *medium*}$_{0.00}$}$_{1.00}$ OR {{PW is *medium*}$_{0.99}$}$_{1.00}$

Virginica	IF {{Z1}$_{0.00}$ AND {PW is *high*}$_{0.00}$}$_{1.00}$ OR {{PL is *high*}$_{0.62}$ OR {PW is *high*} 1.00} 1.00 THEN Output is *virginica*
	Z1={{NOT PW is *medium*}$_{0.00}$}$_{1.00}$ OR {{PW is *high*}$_{1.00}$ OR {NOT PW is *medium*}$_{0.99}$}$_{1.00}$

Table 10.16. Logic description of IRIS data

7.4 Discussions

In this study, we have proposed a logic network composed of functional modules of OR/AND fuzzy neurons. The neurons of this type come with a high level of functional flexibility and encompass "pure" AND and OR fuzzy neurons. By adjusting the connections of the neurons one could easily model intermediate logic characteristics of the logic mapping. We have proposed a genetic scheme of optimization of the network with intent of addressing the structural facet of learning. The critical issues in this learning deal with the depth of the network (the number of the neurons organized in series) and the width of the network (that is the number of inputs to each neuron); apparently these two design parameters imply the number of the input variable used by the network so in this manner one reduces the number of all input variables existing in the problem. Architecturally, the network forms a cascade type of realization of the multivariable logic mapping so in essence we can come up with a stepwise logic description of experimental data.

In the design of the network we have demonstrated the role of the interface between the physical variables and those of logic character required by the model. The encoding scheme dwells on fuzzy sets treated as a collection of semantically sound information granules. Interestingly by choosing their number one could easily control the level of specificity of the network as well as contribute to its interpretability. The development process of the fuzzy networks is highly interactive and designer-oriented: the design relies on the iterative process that allows us to form a plausible tradeoff between accuracy of the network (which usually results in large structures) and interpretability of the logic description whose compactness is a genuine asset. This process relies heavily on the interaction with the designer who has to be provided with various options. The ability to select specific values of the threshold levels is one of the essential design mechanisms that help achieve a required tradeoff between accuracy and interpretability.

Part V Data Modeling Using Parallel OR/AND Neuron Based Fuzzy Neural Networks

In this part of the thesis, we place OR/AND neurons in a parallel manner, to form a new modeling structure. This parallel OR/AND neuron based fuzzy neural network (POAFNN) structure is designed for classification problems. The main goal of this research is using the Pareto-based multi-objective optimization approach to gain better performance on skewed dataset.

Chapter 11 Pareto approach

In this chapter, the multi-objective optimization technique is introduced. The Pareto approach is defined, and other methods used to solve multi-objective optimization problems are also presented.

11.1 Multi-objective optimization

Optimization can be defined as the search for the best possible solution(s) to a given problem. Real-world problems often entail the optimization of multiple objectives. If these objectives are conflicting, then no best solution exists, but a set of good compromise solutions may be found[5_1].

A multi-objective optimization problem often involves formulating a design in which there are several criteria or design objectives. Almost every real-world problem involves simultaneous optimization of several incommensurable and often competing objectives.

While in single-objective optimization the optimal solution is usually clearly defined, this does not hold for multi-objective optimization problems. If the objectives are opposing, then the problem becomes one of finding the best possible design which still satisfies the opposing objectives. An optimum design problem must then be solved with multiple objectives and constraints taken into consideration. Instead of a single optimum, there is rather a set of alternative trade-offs. This type of problem is known as a multi-objective, multi-criteria, or vector optimization problem.

Definitions of a multi-objective problem:

A general multi-objective problem (MOP) includes a set of n parameters (decision variables) $\mathbf{x} = \{x_1, x_2, ..., x_n\} \in \mathbf{X}$, where $\mathbf{X} \subseteq \mathbf{R}^n$ is the n-dimensional decision space or solution space, a set of k objective functions $f_1(\mathbf{x}), f_2(\mathbf{x})..., f_k(\mathbf{x})$. Objective functions are functions of the decision variables. The optimization goal is to $\min\limits_{x \in X}\{f_1(\mathbf{x}), f_2(\mathbf{x})..., f_k(\mathbf{x})\} where \mathbf{X} \subseteq \mathbf{R}^n$ or $\max\limits_{x \in X}\{f_1(\mathbf{x}), f_2(\mathbf{x})..., f_k(\mathbf{x})\} where \mathbf{X} \subseteq \mathbf{R}^n$ according to different problems. In some cases, these objective functions have a set of m constraints functions, which define the boundary of the problem. In the following discussion, we consider the problem of maximizing optimization. Each feasible set of decision parameters is considered a solution.

11.2 Multi-objective optimization methods

In this section, we provide a short survey of several multi-objective optimization methods. Information on other methods can be found in [5_4, 5_5, 5_7, 5_11, 5_16].

11.2.1 Pareto approach

Given a set of solutions to the problem, a partial ordering can be found by the principle of dominance: A solution is clearly better than (dominating) another solution, if it is better or equal in all objectives, but at least better in one objective. Using this principle, the set of best compromise solutions results by removing all solutions that are dominated by at least one other solution. The remaining solutions are all of equal quality (indifferent). A mutual comparison of two solutions shows that each one is always better and worse in at least one objective. This set of indifferent solutions is referred to as the Pareto set, named after the work of the engineer and economist Vilfredo Pareto. Starting from a Pareto solution, one objective can be improved only at the expense of at least one other objective. All the definitions described here are in accordance with [5_1, 5_6, 5_9, 5_12, 5_14, 5_16].

Definition 1 A solution $\mathbf{a} \in \mathbf{X}$ is dominating solution $\mathbf{b} \in \mathbf{X} (\mathbf{a} \succ \mathbf{b})$ if and only if it is superior or equal in all objectives, and at least superior in one objective. This can be expressed as $\mathbf{a} \succeq \mathbf{b}$, if $\forall i \in \{1,2,\dots,m\} : f_i(\mathbf{a}) \geq f_i(\mathbf{b})$ and $\exists j \in \{1,2,\dots,m\} : f_j(\mathbf{a}) > f_j(\mathbf{b})$.

When we define a weaker condition for definition 1, we get the definition of a weak dominating relationship between two solutions.

Definition 2 A solution $\mathbf{a} \in \mathbf{X}$ is weakly dominating a solution $\mathbf{b} \in \mathbf{X}$ $(\mathbf{a} \succeq \mathbf{b})$ if and only if it is superior or equal in all objectives. This can be expressed as $\mathbf{a} \succeq \mathbf{b}, if \forall i \in \{1,2,\dots,m\} : f_i(\mathbf{a}) \geq f_i(\mathbf{b})$.

Now we are able to define the indifferent relationship between two solutions.

Definition 3 The solution $\mathbf{a} \in \mathbf{X}$ is indifferent to a solution $\mathbf{b} \in \mathbf{X} (\mathbf{a} \sim \mathbf{b})$, if and only if neither solution is dominating the other ($\neg (\mathbf{a} \succeq b) \wedge \neg (b \succeq a)$).

Definition 4 A solution $\mathbf{a} \in \mathbf{X}$ is said to be nondominated with regard to a set $\mathbf{A} \subseteq \mathbf{X}$, if and only if $\neg \exists \mathbf{b} \in \mathbf{X} : \mathbf{b} \succ \mathbf{a}$. Here, $\mathbf{X}_f \subseteq \mathbf{X}$ represents the set containing all the feasible solutions we currently have. If and only if the solution $\mathbf{a} \in \mathbf{X}$ is nondominated regarding the set $\mathbf{X}_f$, $\mathbf{x}$ is said to be Pareto optimal.

When no *a priori* preference is defined among the objectives, dominance is the only way to determine if one solution performs better than the other. Furthermore, the best solutions to a multi-objective problem are the nondominated subset among all feasible solutions. These are denoted as the Pareto optimal set; the corresponding objective vectors form the Pareto-optimal front or surface.

Definition 5 Let $\mathbf{A} \subseteq \mathbf{X}_f$. The function *p(A)* gives the set of nondominated decision vectors in A: p(A)={ $\mathbf{b} \in \mathbf{A}$ | b is nondominated with regard to A}.

The set *p(A)* is the nondominated set with regard to A and the resulting set of objective vectors *f(p(A))* is the nondominated front (surface) with respect to A. In the sequel, if $A=X_f$, the $p(X_f)$ is called the Pareto-optimal set and the set $f(p(X_f))$ is denoted as the Pareto optimal front.

11.2.2 Weighted sum method

Weighted sum is a method of secularization of vector functions.

Definition 6 For a function $F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x})..., f_k(\mathbf{x}))$, which is a vector of target objects and a vector $\mathbf{w} = (w_1, w_2, ..., w_k)$, so that $\sum_{i=1}^{k} w_i = 1$, define

$$Fw(\mathbf{x}) = \sum_{i=1}^{k} w_i \bullet f_i(\mathbf{x})$$

We have transferred the original optimization problem into the problem of finding the optimal result for function $Fw(\mathbf{x})$ with a suitable vector w. In this way, we are able to deal with the original problem using single-objective optimization methods.

A normal way to assign the weight vector is to consider the importance of the objectives. The more important objective will get a higher weight, while the less important objective will get a lower weight. However, since not all the objectives necessarily have the same value range, these objectives should be normalized before being weighted.

The advantages of using weighted sum are:

a) multi-objective function is reduced to a single-objective function;

b) traditional optimization methods can now be applied.

The disadvantages are also very obvious. Since the results of solving an optimization problem using the weighted sum method can vary significantly as the weighting coefficients change, and since very little is generally known about how to choose these coefficients, a necessary approach is to solve the same problem for many different values of *w*. In this case, however, we are still confronted with the decision of having to choose the most appropriate solution based on our intuition.

11.2.3 The ε-constraint Method

This method is based on minimization of one (the most preferred or primary) objective function, and consideration of the other objectives as constraints bound by some allowable levels ε_i. Hence, a single objective minimization is carried out for the most relevant objective function, say f_r, subject to additional constraints on the other objective functions. The levels ε_i are then altered to generate the entire Pareto optimal set. The method could be formulated as follows:

$f_r(\mathbf{x}^*) = \min f_r(\mathbf{x})$ subject to additional constraints of the form $f_i(\mathbf{x}) \le \varepsilon_i$ for i=1,2,...,k and i≠r, where ε_i are assumed values of the objective functions which we wish not exceed.

This method is also known as a trade-off method, because of its main concept of trading the value of one objective function for the value of another function. A difficulty with this technique is that the new feasible solution set which is obtained, might be empty. This will occur if the lower bounds are not chosen appropriately. In order to avoid this situation, a suitable range of values for the ε_i must be known beforehand. In other words, knowledge of the problem may be required for this approach, but may not always be available.

Chapter 12 Pareto in Data Modeling for Skewed Datasets

The parallel OR/AND neuron based fuzzy neural network is designed and tested mainly for performance on skewed datasets. Single-objective and multi-objective optimization approaches are used to construct the POAFNN models. The comparison for the two methods are presented in this chapter.

12.1 Description of Model

12.1.1 Model Structure

We use the parallel fuzzy OR/AND neuron as the fundamental part of the model, but modified the connections to OR neuron at the second layer in the OR/AND neuron and we combined several OR/AND neurons together to form a united model. The number of inputs to each OR/AND neuron is identical. Each neuron at the first layer in the OR/AND neuron has a connection to the second layer OR neurons. Each OR/AND neuron represents one output for the whole system, and the final output of the system is the class number derived by comparing the outputs of the OR/AND neurons. The system structure dealing with 3 classes is shown in Figure 12.1.

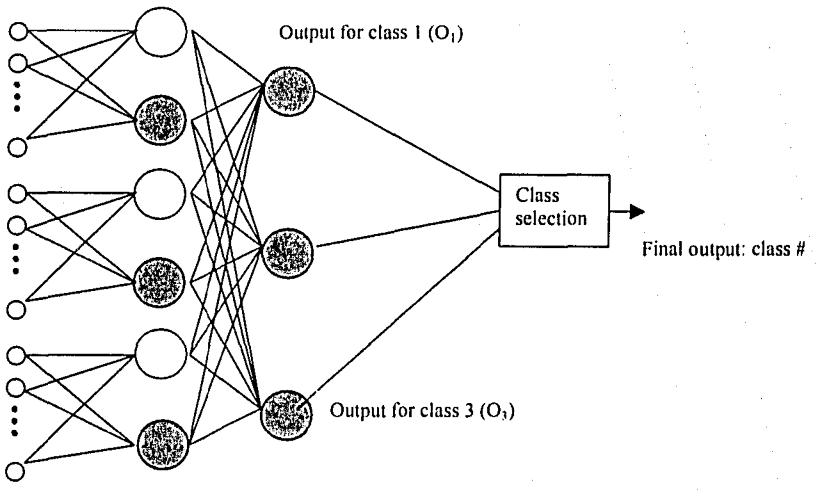


FIGURE 12.1 THE STRUCTURE FOR THE MODEL WITH 3 CLASSES

There are two ways to implement the "class selection" function. One is using maximum output and another is using threshold value.

- Maximum output

In this approach, we compare the output value of all the outputs for these classes in the whole structure. The one giving the max value will be selected and the output of the model will be generated according to the index of this output.

- Threshold value

This technique is implemented by defining a threshold value t, with which the output of each class will be compared. If there is one and only one output, O_i is greater than or equal to t, and the final output is the class number represented by O_i. Otherwise, the output of the model is *no class* (value is –1).

We have performed a large number of experiments using both maximum and threshold techniques, and found that, by using the threshold value, we could extract rules from the raw data with better combination performance. In other words, the rules obtained with the threshold approach have a higher confidence level. The confidence level can be calculated by the Laplace error estimate [5_2].

12.1.2 Model Representation

We use genetic algorithm to construct models when using the single objective optimization approach. For the Pareto-based multi-objective optimization approach, we trained the models using the evolutionary strategy. The model structure is represented as a chromosome in the process of both the genetic algorithm and the evolutionary strategy. Each chromosomeconsists of two parts – variable indexes and connection weights. In the variable indexes part, the index values are presented by integers ranging from 0 to $N-1$ (supposing there are N variables). The length of this part is determined by the number of classes (C) and input variable number (I) to separating OR/AND neurons, which is $C*I$. The connection weights part contains float values ranging from 0 to 1. The number of weights for a chromosome is $(2*I+2*C)*C$.

The model needs to be post-processed after the training of the model is completed in order to gain better interpretability without shrinking the system performance too much. Model post-processing is performed using the technique of model pruning. Model pruning is carried out by changing the thresholds for OR and AND neurons in a similar manner to what was done in Part 4. For the AND neuron, the input variable has significant influence on the final output when the weight for this variable to the neuron is close to 0. For the OR neuron, the input has a stronger effect on the output if the weight is closer to 1. So, for the AND neuron, the inputs with weight near *1* will not contribute a great deal to the output. The same is true with the OR neuron, if the weight is close to *0*. If we change the threshold values *1* for the AND neuron and *0* for the OR neuron, we may find that more inputs are not very useful. By changing threshold values, we are able to eliminate some non-significant inputs to obtain a simpler model, perhaps with little change to the accuracy of the model. In our research, we change the threshold for the AND neuron and the OR neuron separately, while ensuring the output accuracy will not drop by more than *5%*.

12.1.4 Rule Recombination

After we achieve optimal models (or rule sets) for each class, we can combine these rule sets to obtain the final classification results. In this approach, we compare the output of each rule set; the rule set which gives the maximum value will be considered to be the final classification result.

12.2 Training Criteria Introduction

We tried out several modeling criteria in training this type of structure. All the criteria are related to the classification confusion matrix. In this section, we first introduce the classification confusion matrix, followed by several related concepts .

12.2.1 Confusion Matrix

The confusion matrix [5_10] is a common tool for assessing the quality of classification results. It is simply a square matrix that shows the various classifications and misclassifications of the model in a compact area. The columns of the matrix correspond to the number of instances classified as a particular value, and the rows correspond to the number of instances with that actual classification. For binary classification problems, the confusion matrix contains the following possibilities: the number of True Negative (TN), False Positive (FP), False Negative (FN) and True Positive (TP) classifications (Figure 12.1).

	Actual Positive	Actual Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

FIGURE 12.1 THE CONFUSION MATRIX FOR 2-CLASS CLASSIFICATION

12.2.2 Confusion Matrix Derived Measures

Based on the confusion matrix, a number of measures have been defined to describe classification performance. Some of them are introduced as follows. (To simplify, we define $N = TP + TN + FP + FN$.)

- Classification Rate (Accuracy)

The goal of classification problems is to maximize TP and TN. The classification rate is the sum of TP and TN over the whole dataset: $\frac{TP + TN}{N}$. This measure provides us with the accuracy of the classifier on the dataset.

- Prevalence

This measure gives the proportion of the actual positive samples in the whole dataset. It is defined as $\frac{TP + FN}{N}$.

- Recall or Sensitivity

This measure is the proportion of positive cases that were correctly identified, as calculated using the equation $\frac{TP}{TP+FN}$. This measure is useful when capturing present points is most important (i.e., a model can capture all positives, but there may be many false positives).

- Specificity

The specificity measure is the proportion of negative cases that were incorrectly classified as positive, as calculated using the equation $\frac{TN}{TN+FP}$. This measure is most useful when eliminating absent points is most important; for example, a model has few false positives, but some actual present points may be eliminated.

- Positive Predictive Power

This measure is the proportion of the predicted positive cases that were correct, as calculated using the equation $\frac{TP}{TP+FP}$.

When we apply the confusion matrix to problems that have more than *2* classes, a simple method is to convert this kind of confusion matrix into a two-class confusion matrix by focusing on just one class at one time. In this way, the selected class classifies positive samples, and the other classes contribute to the negative samples. For each class, we can calculate the relevant measures. For example, we have the confusion matrix with *n* classes as $\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$. For class *i* (*i=1..n*), $TP = a_{ii}$, $FP = \sum_{j=1}^{n} a_{ij} - a_{ii}$, $FN = \sum_{j=1}^{n} a_{ji} - a_{ii}$ and $TN = \sum_{j=1}^{n}\sum_{k=1}^{n} a_{ij} - TP - FP - a_{ii}$. After we obtain the values for *TP*, *FP*, *FN* and *TN*, we continue to calculate the measures mentioned above.

12.2.3 Measures Applied

We tried several modeling criteria in training this type of structure and have selected 3 of them for presentation of their results:

- The classification Overall Accuracy (OA) on the whole dataset

With the confusion matrix $\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$, the OA is calculated as

$OA = \dfrac{\sum_{i=1}^{n} a_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}}$. This measure is used in single optimization.

- The Pareto approach using the combination of the classifications, Specificity, Sensitivity, and Accuracy (abbreviated as SSA)

SSA is calculated using the product of the Specificity, Sensitivity, and Accuracy measures when we transfer the n-class confusion matrix to a *2*-class confusion matrix.

- The Pareto approach using the combination of the classifications of Accuracy, Positive predictive power, and Recall (abbreviated as APR)

APR is calculated using the product of the Accuracy, Positive predictive power, and Recall measures when we transfer the *n*-class confusion matrix to a *2*-class confusion matrix.

12.2.4 Training Method

Generally speaking, solving multi-objective problems is very difficult. In an attempt to stochastically solve problems of this generic class in an acceptable timeframe, specific multi-objective evolutionary algorithms (MOEAs) [5_3, 5_8, 5_13, 5_15] were initially developed in the mid-1980s. In this section, we continue to use the Evolutionary Computation approach to train the models. We use GA to train the models using the OA approach and apply ES when we attempt to train the models using the multi-objective optimization approach, Pareto-based, SSA, and APR methods. GA and ES share the same chromosome structure. The MOEA algorithms applied here are in part taken from [5_18, 5_16, 5_17].The datasets are divided into training and testing data 10 times. For each split of the training and testing data, 5 experiments are carried out. The following results are based on these 50-experiment sets

12.3 Experimental results

12.3.1 Experiments on Synthetic Data

Here we use a synthetic dataset which contains 12 input attributes. Within these 12 input attributes, there are 9 continuous attributes and 3 discrete attributes (consisting of 8, 5, and 6 unique values, respectively). The data have 2 classes and the proportion of these two classes is 1:4. This synthetic data set is constructed based on three rules (**X_n** represents the nth input attribute):

1) IF X_9 is Medium THEN class 1

2) IF X_{10} is High THEN class 2

3) IF X_5 is 5 THEN class 2

Experiments were carried out on this synthetic data set with the three training criteria discussed above. For each training criteria, we randomly split the original data 10 times set into training-testing data. For each data split, we trained the model – which has 5 inputs for every output – with different randomizing seed 5 times. The original data set was transformed into a fuzzified data set by applying 3 fuzzy sets (Low-Medium-High) on continuous attributes, and the 1-out-of-n technique on discrete attributes.

12.3.1.1 Experimental Results on OA

First, we examine the training results on the statistical analysis of the confusion matrixes. Table 12.1 shows the mean value and the standard deviation on the confusion matrixes.

For training data	
0.000± 0.000	0.000± 0.000
112.360± 17.071	470.040± 15.336
For test data	
0.000± 0.000	0.000± 0.000
73.100± 11.709	313.340± 13.106

Table 12.1 Statistical analyses on confusion matrixes

From Table 12.1, we see that the OA approach focuses only on the class which is the majority in the data set; therefore, the models simply classified all the data to the majority class. For OA, we obtain the rules from the model, as in Table 12.2.

IF $\{\{$X10 is Low$\}_{0.732346}$ OR $\{$X10 is High$\}_{0.624400}$ $\}_{0.927565}$ THEN class 2
IF $\{\{$X10 is Medium$\}_{0.958442}$ $\}_{0.914048}$ THEN class 2

Table 12.2 Sample rules obtained in OA approach

By observing Tables 12.1 and 12.2, we can easily conclude that the approach using OA is not suitable for extracting knowledge from this skewed synthetic dataset.

12.3.1.2 Experimental results on the APR approach

First, we examine the training results on the statistical analysis of the confusion matrixes. Table 12.3 shows the mean value and the standard deviation on the confusion matrixes.

Pareto focuses on class 1		Pareto focuses on class 2	
For training data		For training data	
116.040± 15.995	0.580± 1.071	54.460± 33.036	0.000± 0.000
0.040± 0.283	367.380± 100.042	1.240± 2.308	464.400± 60.951
For test data		For test data	
71.780± 20.966	0.980± 1.684	34.400± 22.144	0.000± 0.000
0.040± 0.198	235.280± 80.261	1.500± 3.099	292.360± 81.417

Table 12.3 Statistical analyses on confusion matrixes

From Table 12.3, we see that the APR approach is able to get good results on both the majority class and the minority class. For APR, we obtain the rules from the model, as in Table 12.4.

IF $\{\{X9 \text{ is Medium}\}_{0.703631}$ OR $\{X9 \text{ is High}\}_{0.301510}$ OR $\{X10 \text{ is Low}\}_{0.093988}$ OR $\{X10 \text{ is Medium}\}_{0.096336}\}_{0.978602}$ THEN class 1
IF $\{\{X9 \text{ is Medium}\}_{0.202465}$ AND $\{X10 \text{ is Low}\}_{0.254935}$ AND $\{X10 \text{ is Medium}\}_{0.624705}\}_{0.308416}$ THEN class 1
IF $\{\{X5 \text{ is } 5\}_{0.209147}$ OR $\{X9 \text{ is Low}\}_{0.983534}$ OR $\{X9 \text{ is High}\}_{0.698462}$ OR $\{X10 \text{ is Medium}\}_{0.191100}$ OR $\{X10 \text{ is High}\}_{0.234073}\}_{0.935656}$ THEN class 2

Table 12.4 Sample rules obtained using the APR approach

The Pareto surface for APR is shown in Figure 12.2

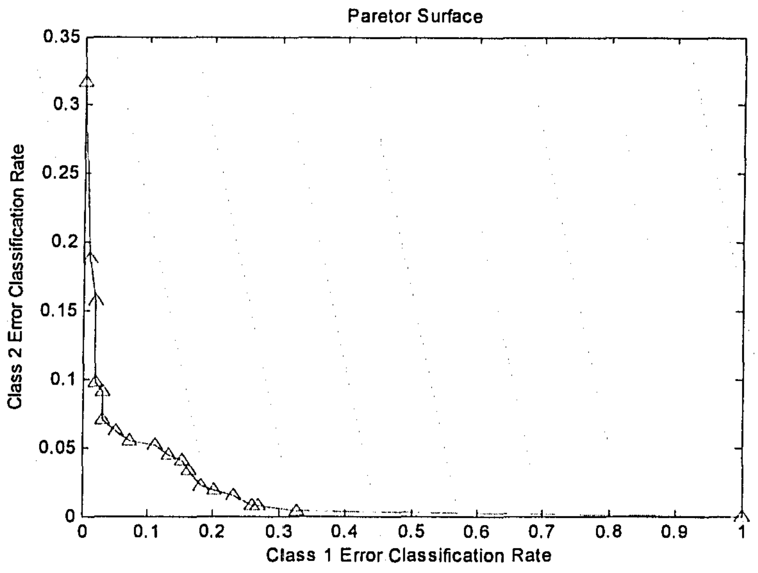


FIGURE 12.2 PARETO SURFACE OBTAINED WITH THE APR APPROACH

12.3.1.3 Experimental results using the SSA approach

First, we examine the training results on the statistical analysis of the confusion matrixes. Table 12.5 shows the mean value and the standard deviation on the confusion matrixes.

Pareto focuses on class 1		Pareto focuses on class 2	
For training data		For training data	
119.280± 5.489	2.980± 3.809	63.620± 34.427	0.020± 0.141
0.000± 0.000	382.240± 82.081	0.360± 0.898	473.960± 8.046
For test data		For test data	
77.320± 5.073	2.580± 2.935	40.580± 21.415	0.060± 0.314
0.000± 0.000	256.200± 54.108	0.400± 1.245	316.080± 6.446

Table 12.5 Statistical analyses on confusion matrixes

From Table 12.5, we see that the APR approach is able to get good results on both the majority class and the minority class. For APR, we obtain the rules from the model, as in Table 12.6.

IF $\{\{X9 \text{ is Medium}\}_{0.945245}$ OR $\{X9 \text{ is High}\}_{0.369780}\}_{0.899098}$ THEN class 1
IF $\{\{X5 \text{ is } 5\}_{0.436560}$ OR $\{X9 \text{ is Low}\}_{0.970481}$ OR $\{X9 \text{ is High}\}_{0.777895}$ OR $\{X10 \text{ is High}\}_{0.678811}\}_{0.711764}$ THEN class 2

Table 12.6 Sample rules obtained using the SSA approach

The Pareto surface for SSA is shown in Figure 12.3.

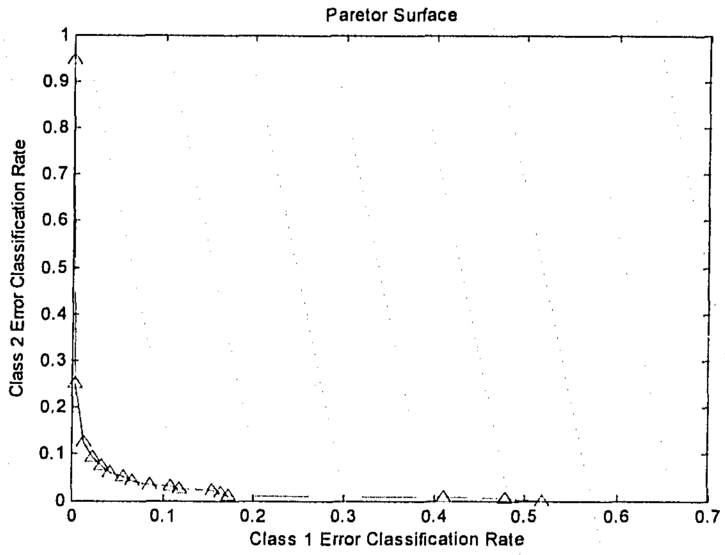


FIGURE 12.3 PARETO SURFACE OBTAINED WITH THE APR APPROACH

12.3.1.4 Comparing the rules

From Tables 12.3 and 12.5, we see that the rules derived from the APR and SSA approaches are similar. They all have a close relationship to the rules used to generate the synthetic dataset, but there are still some differences. For class 1, we use the rule "*IF X9 is Medium THEN class 1*", but in Tables 12.3 and 12.5, we see that there is a rule, "*IF X9 is Medium OR X9 is High THEN class 1.*" The main reason for this difference is the different membership functions used in generating dataset and training models. Figure 12.4 describes the difference between these membership functions.

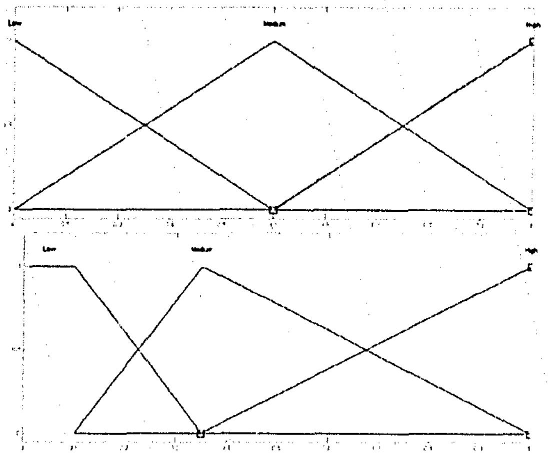


FIGURE 12.4 MEMBERSHIP FUNCTION IN GENERATING DATASET (LEFT) AND MODEL TRAINING (RIGHT)

12.3.2 Experimental results for the Software Metric Usability Dataset

From the experiments in 12.3.1, we see that the modeling approach using Pareto APR or SSA is able to extract knowledge from the raw dataset, although there would be some difference due to the membership function type, or format applied on the raw dataset when we fuzzify the data. In this section, we apply the modeling approach on the Software Metric dataset for software usability.

From Chapter 6, we know that the usability data for expert A is the most skewed dataset, expert D's dataset is next ,and expert V's is the last. We choose the most skewed dataset, which is the usability data for expert A, to demonstrate the performance of the modeling approach. For experts D's and V's usability data, we also present the experimental results. With the synthetic data, we have 10 splits on the datasets, and 5 experiments on each split.

12.3.2.1 Experimental results for the OA approach

From 12.2.1.1, we find that the OA approach is not a good solution for the proposed model in the synthetic data. Here we use the statistical data to show the feasibility of the OA approach in a real dataset. The statistical data are shown in Table 12.7.

For training data		
0.000± 0.000	0.000± 0.000	0.040± 0.198
0.040± 0.198	0.000± 0.000	0.140± 0.756
2.020± 1.040	8.140± 2.030	187.940± 22.377
For test data		
0.000± 0.000	0.000± 0.000	0.040± 0.283
0.000± 0.000	0.000± 0.000	0.060± 0.314
1.300± 0.974	5.000± 1.591	124.080± 15.183

Table 12.7 Statistical analysis on OA for usability data expert A

From Table 12.7, it is clear that the OA approach focuses on the majority class and ignores the two minority classes.

12.3.2.2 Experimental results for the APR approach

First, we show the statistical analysis on the APR approach in Table 12.8.

Pareto focuses on class 1		
For training data		
2.260± 1.006	0.000± 0.000	0.000± 0.000
0.000± 0.000	1.640± 2.229	0.700± 1.474
0.240± 0.431	3.920± 2.656	164.000± 41.304
For test data		
0.080± 0.274	0.020± 0.141	0.320± 0.683
0.120± 0.385	0.360± 0.776	1.380± 2.828
0.620± 0.923	3.120± 2.047	06.120± 27.412
Pareto focuses on class 2		
For training data		
0.440± 0.675	0.000± 0.000	0.380± 1.176
0.060± 0.240	6.580± 1.295	0.440± 0.675
0.360± 0.525	1.000± 0.926	114.200± 62.083
For test data		
0.020± 0.141	0.040± 0.198	0.340± 0.872
0.340± 0.593	0.860± 1.125	3.460± 2.121
0.200± 0.495	1.040± 1.228	74.240± 42.200
Pareto focuses on class 3		
For training data		
0.320± 0.587	0.000± 0.000	0.000± 0.000
0.020± 0.141	0.060± 0.240	0.000± 0.000
0.940± 0.620	5.020± 1.801	207.980± 1.995
For test data		
0.000± 0.000	0.000± 0.000	0.000± 0.000
0.020± 0.141	0.000± 0.000	0.000± 0.000
1.020± 0.958	5.160± 1.530	134.900± 2.809

Table 12.8 Statistical analyses on confusion matrixes (expert A)

Table 12.8 shows that the classification on the specific class improved with the changing focus in the target class; it provides a much better performance for the minority classes than does the OA approach.

Next, we select data split #6 to analyze the rules we obtained and the performance of the combination of the rules. For data split #6, the historical graph of the class distribution on training and testing is shown in Figure 12.5.

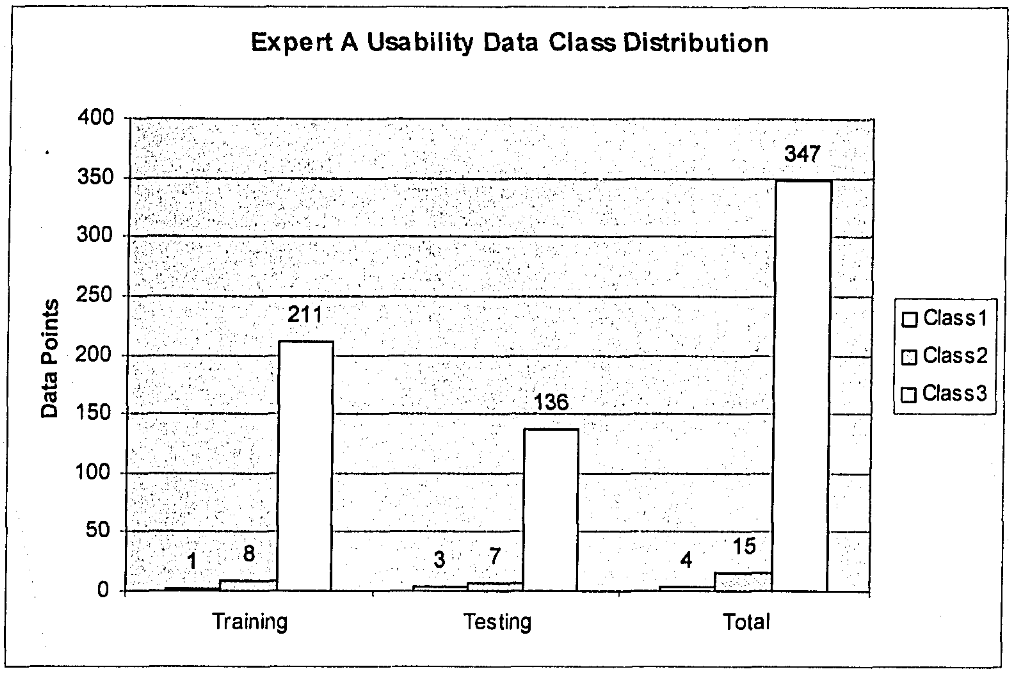


FIGURE 12.5 HISTORICAL GRAPHS OF THE USABILITY DATA FOR EXPERT A SPLIT #6

In Figure 12.5, for training, testing and total datasets, the left column represents class 1, the middle column represents class 2 and the right column represents class 3. One of the Pareto 3D scatter graphs is shown in Figure 12.6.

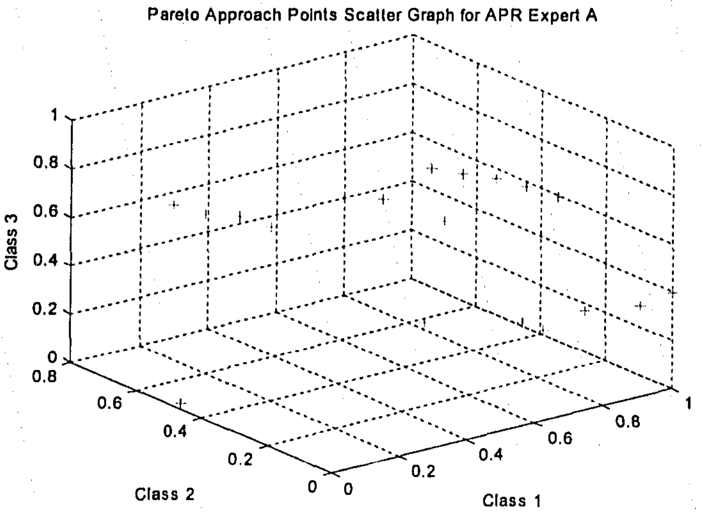


FIGURE 12.6 THE 3D SCATTER PLOT FOR THE APR APPROACH

The models obtained with the APR Pareto approach are pruned as indicated in 12.1.3, in order to achieve more interpretable rule sets. Table 12.9 shows the rules after pruning for data split #6.

For class 1

And rule 1
{CHLD is 8}$_{0.425557}$ AND {HLUN is Low}$_{0.073286}$
Or rule 1
{ATCO is High}$_{0.581149}$
And rule 3
{ADEC is 7}$_{0.393473}$ AND {FACE is 3}$_{0.428018}$ AND {LCOM is High}$_{0.070016}$ AND {OVRM is 5}$_{0.580455}$

IF {And rule 1}$_{0.406305}$ OR {And rule 3}$_{0.918951}$ OR {Or rule 1}$_{0.780950}$ THEN class 1

For class 2

And rule 1
{FACE is 4}$_{0.154157}$ AND {FNOT is Low}$_{0.811515}$ AND {CLAS is 5}$_{0.419970}$ AND {CONS is 8}$_{0.013363}$
Or rule 1
{FACE is 4}$_{0.370459}$ OR {FNOT is Low}$_{0.030629}$ OR {CLAS is 5}$_{0.302440}$ OR {CONS is 8}$_{0.624515}$ OR {WMC2 is High}$_{0.514258}$
And rule 2
{RFO is High}$_{0.047521}$ AND {MIC is 12}$_{0.689692}$ AND {ATTR is Medium}$_{0.445078}$ AND {OVRM is 10}$_{0.156858}$ AND {PROM is Low}$_{0.735041}$
Or rule 2
{RFO is High}$_{0.364495}$ OR {MIC is 12}$_{0.838123}$ OR {ATTR is Medium}$_{0.951888}$ OR {OVRM is 10}$_{0.835529}$ OR {PROM is Low}$_{0.962305}$
And rule 3

{TOK is Low}$_{0.309708}$ AND {ADEC is 9}$_{0.480861}$ AND {WDC is Medium}$_{0.393158}$ AND {MNL1 is Medium}$_{0.827871}$ AND {MAXP is 6}$_{0.819408}$ Or rule 3 {TOK is Low}$_{0.057898}$ OR {ADEC is 9}$_{0.017304}$ OR {WDC is Medium}$_{0.082244}$ OR {MNL1 is Medium}$_{0.902176}$ OR {MAXP is 6}$_{0.182413}$ IF {And rule 1}$_{0.550176}$ OR {And rule 2}$_{0.012364}$ OR {And rule 3}$_{0.209279}$ OR {Or rule 1}$_{0.936670}$ OR {Or rule 2}$_{0.176856}$ OR {Or rule 3}$_{0.159848}$ THEN class 2
For class 3
Or rule 1 {MDEC is 1}$_{0.309051}$ OR {CHLD is 0}$_{0.687076}$ OR {RFO is Medium}$_{0.605976}$ OR {HLUN is Low}$_{0.484937}$ Or rule 2 {TYPE is 2}$_{0.881146}$ OR {LOC is Low}$_{0.632196}$ OR {TOK is Medium}$_{0.553126}$ OR {CONS is 7}$_{0.721461}$ OR {PROM is High}$_{0.494498}$ Or rule 3 {CLAS is 5}$_{0.858582}$ IF {Or rule 1}$_{0.654649}$ OR {Or rule 2}$_{0.724382}$ OR {Or rule 3}$_{0.348169}$ THEN class 3

Table 12.9 Rules obtained in APR Pareto approach for data split #6

After we obtain the pruned models, we combine these models to classify the dataset. In the combination, we use the maximum of the models' output as the final class, and we obtain the confusion matrix following the combination of pruned models. (Table 12.10).

```
Training data confusion matrix
  0  0  1
  0  3  1
  1  5  209
Classification accuracy: 0.963636
Testing data confusion matrix
  0  1  0
  1  1  4
  2  5  132
Classification accuracy: 0.910959
```

Table 12.10 Confusion matrix after combining models

12.3.2.3 Experimental results on the SSA approach

First, we show the statistical analysis on the APR approach, in Table 12.11.

Pareto focuses on class 1		
For training data		
2.500± 1.035	0.020± 0.141	0.020± 0.141
0.000± 0.000	3.560± 3.131	6.780± 8.379
0.000± 0.000	0.560± 0.760	133.000± 58.893
For test data		
0.080± 0.274	0.260± 0.723	0.760± 1.271
0.240± 0.517	0.780± 1.130	5.680± 7.221
0.340± 0.557	1.720± 1.727	82.560± 38.146
Pareto focuses on class 2		
For training data		
0.640± 0.942	0.020± 0.141	3.260± 5.876
0.280± 0.536	9.000± 1.278	6.180± 3.863
0.160± 0.370	0.060± 0.240	94.720± 57.912
For test data		
0.020± 0.141	0.200± 0.404	2.180± 3.998
0.520± 0.677	1.440± 1.033	7.540± 2.823
0.120± 0.328	0.620± 0.697	61.500± 38.767
Pareto focuses on class 3		
For training data		
0.240± 0.716	0.040± 0.283	0.120± 0.521
0.060± 0.240	0.760± 1.572	0.940± 1.953
0.020± 0.141	0.220± 0.465	192.260± 6.922
For test data		
0.080± 0.274	0.040± 0.198	0.040± 0.198
0.080± 0.340	0.100± 0.303	0.740± 1.549
0.660± 0.823	3.060± 1.778	123.900± 3.813

Table 12.11 Statistical analyses on confusion matrixes

Table 12.11 shows that classification on the specific class improved with the changing focus in the target class. It provides much better performance for the minority classes than the OA approach does. One of the Pareto 3D scatter graphs is shown in Figure 12.7.

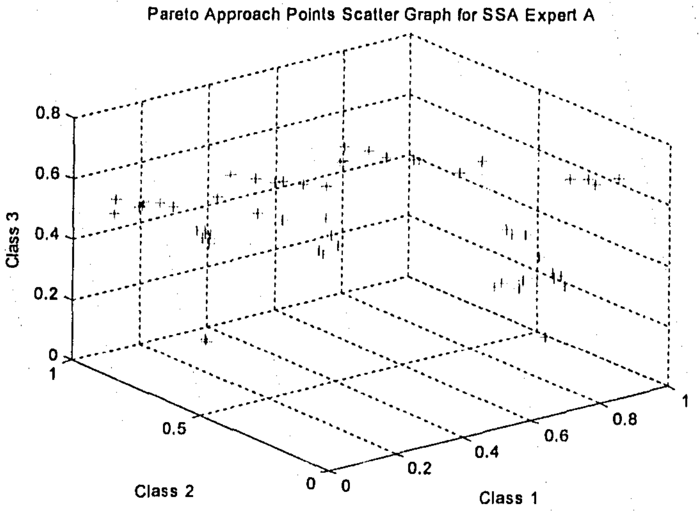


FIGURE 12.7 THE 3D SCATTER PLOT FOR THE SSA APPROACH

For class 1

And rule 1

{IMST is Medium}$_{0.005810}$ AND {PUBM is Low}$_{0.045474}$

IF {And rule 1}$_{0.885987}$ THEN class 1

For class 2

And rule 1

{MDEC is 4}$_{0.248939}$ AND {MDEC is 10}$_{0.330854}$ AND {MAXP is 0}$_{0.150592}$

And rule 2

{MWDC is 16}$_{0.040358}$ AND {CBO is Low}$_{0.234855}$

Or rule 2

{MDEC is 10}$_{0.234767}$ OR {MWDC is 16}$_{0.932697}$ OR {CBO is Low}$_{0.085431}$ OR {WMC1 is High}$_{0.486816}$

And rule 3

{INCL is 2}$_{0.356554}$ AND {MNL2 is High}$_{0.170931}$ AND {PROM is Low}$_{0.407681}$

Or rule 3

{ALOC is Medium}$_{0.300554}$ OR {INCL is 2}$_{0.910444}$ OR {MNL1 is Medium}$_{0.569300}$ OR {MNL2 is High}$_{0.685231}$ OR {PROM is Low}$_{0.659878}$

IF {And rule 1}$_{0.097055}$ OR {And rule 2}$_{0.052668}$ OR {And rule 3}$_{0.358051}$ OR {Or rule 2}$_{0.992928}$ OR {Or rule 3}$_{0.228498}$ THEN class 2

For class 3

And rule 1

{INCL is 3}$_{0.039691}$ AND {INCL is 8}$_{0.019651}$

Or rule 1

{TYPE is 3}$_{0.389817}$ OR {ADEC is 7}$_{0.353241}$ OR {INCL is 3}$_{0.407295}$ OR {INCL is 8}$_{0.427132}$ OR {MAXL is 4}$_{0.350110}$

And rule 2

{CHLD is 3}$_{0.078406}$

Or rule 2

{ATOK is High}$_{0.552006}$ OR {CHLD is 3}$_{0.676342}$ OR {ATCO is

Low}$_{0.657739}$ OR {ATCO is Medium}$_{0.904957}$ OR {ATTR is Medium}$_{0.667247}$
And rule 3
{ATOK is Low}$_{0.035967}$ AND {ADEC is 3}$_{0.008391}$
Or rule 3
{ATOK is Low}$_{0.277025}$ OR {ADEC is 3}$_{0.118408}$ OR {LCOM is High}$_{0.715947}$ OR {WMC1 is High}$_{0.494564}$
IF {And rule 1}$_{0.350167}$ OR {And rule 2}$_{0.452347}$ OR {And rule 3}$_{0.106970}$ OR {Or rule 1}$_{0.271821}$ OR {Or rule 2}$_{0.984087}$ OR {Or rule 3}$_{0.398385}$ THEN class 3

Table 12.12 Rules obtained in the SSA Pareto approach for data split #6

After we obtain the pruned models, we combine these models to classify the dataset. As we have done in the APR approach, we obtain the confusion matrix following the combination of pruned models (Table 12.13).

```
Training data confusion matrix
  1  0  2
  0  6  5
  0  2 204
Classification accuracy: 0.959091
Testing data confusion matrix
  0  0  4
  0  2  7
  3  5 125
Classification accuracy: 0.869863
```

Table 12.13 Confusion matrix after combining models

12.3.2.4 Merging of SSA and APR approaches to obtain the final rule set

In Sections 12.3.2.2 and 12.3.2.3, we obtained a set of rule sets from the APR and SSA approaches. For theAPR approach, the models work well when the training methods are applied separately but, after the combination of models, the performance of the combination is not promising. The performance of separate models is worse with the SSA approach than with the APR approach, but the combined results are much better. In this section, we attempt to combine the APR and SSA approaches to see if there is any improvement. In Table 12.14, the combined results for usability expert A's dataset are displayed.

APR APR APR

Training data confusion matrix

0	0	1
0	3	1
1	5	209

Classification accuracy: 96.4%

Testing data confusion matrix

0	1	0
1	1	4
2	5	132

Classification accuracy: 91.1%

APR APR SSA

Training data confusion matrix

1	**0**	**0**
0	**6**	**7**
0	**2**	**204**

Classification accuracy: 95.9%

Testing data confusion matrix

0	**1**	**3**
0	**2**	**7**
3	**4**	**126**

Classification accuracy: 87.7%

APR SSA APR

Training data confusion matrix

0	0	0
0	4	5
1	4	206

Classification accuracy: 95.5%

Testing data confusion matrix

0	0	0
1	2	5
2	5	131

Classification accuracy: 91.1%

APR SSA SSA

Training data confusion matrix

1	0	1
0	6	5
0	2	205

Classification accuracy: 96.4%

Testing data confusion matrix

0	1	4
0	2	7
3	4	125

Classification accuracy: 87.0%

SSA APR APR

Training data confusion matrix

1	0	3
0	3	1
0	5	207

Classification accuracy: 95.9%

Testing data confusion matrix

0	0	4
1	1	4
2	6	128

Classification accuracy: 88.4%

SSA APR SSA

Training data confusion matrix

1	**0**	**2**
0	**6**	**7**
0	**2**	**202**

Classification accuracy: 95.0%

Testing data confusion matrix

0	**0**	**4**
0	**2**	**7**
3	**5**	**125**

Classification accuracy: 87.0%

SSA SSA APR

Training data confusion matrix

1	0	3
0	4	5
0	4	203

Classification accuracy: 94.5%

Testing data confusion matrix

0	0	4
1	2	5
2	5	127

Classification accuracy: 88.4%

SSA SSA SSA

Training data confusion matrix

1	0	2
0	6	5
0	2	204

Classification accuracy: 95.9%

Testing data confusion matrix

0	0	4
0	2	7
3	5	125

Classification accuracy: 87.0%

Table 12.14 Confusion matrix after combination of APR and SSA (expert A)

From Table 12.14, it is clear that the combinations of APR APR SSA and SSA APR SSA are superior to simply using the APR rule combination, and similar to the simple combination of the SSA models. From the view of classification accuracy, the results in

Table 12.14 are not good. For example, the combination of APR APR SSA, the base line of the classification is 93.2% in testing data, while the classification accuracy here is 87.7%. There is a difference between the two accuracy rates of 5.5%. But our results did review the existence of the minority class (class 2) instead of just focusing on the majority one (class 3). The goal of this research is focusing on the classification rate on minority class. So, in this sense, this classification result is good. In Tables 12.15 and 12.16, we show the APR SSA combination results for experts D and V.

APR APR APR
Training data confusion matrix

1	1	0
1	2	1
4	20	190

Classification accuracy: 87.7%
Testing data confusion matrix

0	0	0
0	1	1
11	11	122

Classification accuracy: 84.2%

APR APR SSA
Training data confusion matrix

2	**0**	**0**
4	**15**	**15**
0	**8**	**176**

Classification accuracy: 87.7%
Testing data confusion matrix

1	**2**	**1**
7	**6**	**9**
3	**4**	**113**

Classification accuracy: 82.2%

APR SSA APR
Training data confusion matrix

0	0	0
3	6	7
3	17	184

Classification accuracy: 86.4%
Testing data confusion matrix

0	0	0
4	3	3
7	9	120

Classification accuracy: 84.2%

APR SSA SSA
Training data confusion matrix

2	0	1
4	18	20
0	5	170

Classification accuracy: 86.4%

SSA APR APR
Training data confusion matrix

0	1	5
2	2	1
4	20	185

Classification accuracy: 85.0%
Testing data confusion matrix

0	1	1
0	0	1
11	11	121

Classification accuracy: 82.9%

SSA APR SSA
Training data confusion matrix

2	**4**	**18**
4	**11**	**7**
0	**8**	**166**

Classification accuracy: 81.4%
Testing data confusion matrix

3	**5**	**4**
6	**3**	**7**
2	**4**	**112**

Classification accuracy: 80.8%

SSA SSA APR
Training data confusion matrix

0	0	5
3	6	6
3	17	180

Classification accuracy: 84.5%
Testing data confusion matrix

0	0	1
4	3	2
7	9	120

Classification accuracy: 84.2%

SSA SSA SSA
Training data confusion matrix

2	3	14
4	15	13
0	5	164

Classification accuracy: 82.3%

Testing data confusion matrix

0	1	0
11	7	11
0	4	112

Classification accuracy: 81.5%

Testing data confusion matrix

0	2	3
11	6	10
0	4	110

Classification accuracy: 79.5%

Table 12.15 Confusion matrix after combination of APR and SSA (expert D)

APR APR APR

Training data confusion matrix

0	0	0
2	12	28
14	28	136

Classification accuracy: 67.3%

Testing data confusion matrix

0	0	0
2	9	20
12	25	78

Classification accuracy: 59.6%

APR APR SSA

Training data confusion matrix

6	**7**	**4**
7	**26**	**41**
3	**7**	**119**

Classification accuracy: 68.6%

Testing data confusion matrix

7	**9**	**6**
5	**19**	**29**
2	**6**	**63**

Classification accuracy: 61.0%

APR SSA APR

Training data confusion matrix

0	0	0
0	1	4
16	39	160

Classification accuracy: 73.2%

Testing data confusion matrix

0	0	0
0	0	1
14	34	97

Classification accuracy: 66.4%

APR SSA SSA

Training data confusion matrix

0	2	1
13	25	24
3	13	139

Classification accuracy: 74.5%

Testing data confusion matrix

0	1	1
12	21	20
2	12	77

SSA APR APR

Training data confusion matrix

0	0	0
2	12	28
14	28	136

Classification accuracy: 67.3%

Testing data confusion matrix

0	0	0
2	9	20
12	25	78

Classification accuracy: 0.59.6%

SSA APR SSA

Training data confusion matrix

8	**7**	**6**
5	**26**	**41**
3	**7**	**117**

Classification accuracy: 68.6%

Testing data confusion matrix

9	**10**	**7**
4	**18**	**29**
1	**6**	**62**

Classification accuracy: 61.0%

SSA SSA APR

Training data confusion matrix

0	0	0
0	1	4
16	39	160

Classification accuracy: 73.2%

Testing data confusion matrix

0	0	0
0	0	1
14	34	97

Classification accuracy: 66.4%

SSA SSA SSA

Training data confusion matrix

0	3	0
13	24	25
3	13	139

Classification accuracy: 74.1%

Testing data confusion matrix

0	2	4
12	20	18
2	12	76

Classification accuracy: 67.1%	Classification accuracy: 65.8%

Table 12.16 Confusion matrix after combination of APR and SSA (expert V)

In Tables 12.14, 12.15, and 12.16, we have highlighted the optimal combinations. We find that the combinations of APR APR SSA and SSA APR SSA are superior to simply using the APR rule combination, and similar to the simple combination of the SSA models; however these two combinations are more stable than the simple SSA combination.

12.4 Discussions

In this part of the thesis, we proposed a new model for a rule-based fuzzy classification system. The focus of this system is to improve performance in skewed data. We compared the OA approach and the Pareto-based multi-objective optimization approaches, represented here as APR and SSA approaches. From the experimental results on synthetic data and the software metric dataset, we find that Pareto-based multi-objective optimization approaches are more effective than traditional approaches, such as OA, in modeling skewed datasets. By comparing the results of the combination of APR and SSA models, we are able to say that combining SSA and APR models in specific ways can offer better performance than simply using one kind of model. In our experiments, we find that the combinations of APR APR SSA and SSA APR SSA are the best choices.

Part VI Conclusions and Future Work

Chapter 13 Comparisons of Approaches Used

13.1 Reed Muller Models vs. Cascade Fuzzy Neural Networks

In Chapter 8 and Chapter 10, we performed experiments on the Boston housing data, Abalone dataset, Machine CPU dataset, and Auto-MPG dataset, using the Reed-Muller Binary Decision Tree (RMBDT) and the Cascade OR/AND neuron-based Fuzzy Neural Network (COAFNN). From these results, we obtained the following facts:

- The two modeling structures have similar performance on simple datasets, such as Machine CPU and Auto-MPG.

- When confronted with complex datasets, such as the Boston housing and Abalone datasets, COAFNN performs better than RMBDT. The reason is that COAFNN provides more structural choice in building the models.

- For the structural optimization process, the RMBDT model needs only one parameter as the depth of the tree, while the COAFNN model is defined by two parameters: number of layers and number of inputs to each layer. The optimal structural parameter for RMBDT is easier to find, compared to finding optimal structural parameters for COAFNN. COAFNN is the most complex structure of the three modeling structures we researched: there are more possible models for COAFNN to check, with numerous combinations of the two structural parameters.

- COAFNN outperforms RMBDT in the interpretability of the models.

13.2 Reed Muller Models vs. Parallel Fuzzy Neural Networks

In order to compare the results for the Reed-Muller Binary Decision Tree (RMBDT) structure with those for the Parallel OR/AND neuron-based Fuzzy Neural Network (POAFNN) structure, we carried out the experiments on the software metric data with the models with depth of 2 to 7. Considering the confusion matrix of the classification, we present the results of experts A, D, and V's confusion matrix mean value and standard deviation value, in Table 13.1.

For Expert A dataset		
For training data		
1.683± 0.983	0.000± 0.000	0.000± 0.000
0.000± 0.000	2.717± 1.075	0.333± 0.475
0.783± 0.524	5.867± 1.578	208.617± 1.914
For testing data		
0.333± 0.475	0.000± 0.000	0.183± 0.504
0.183± 0.431	0.683± 0.725	1.333± 1.244
1.017± 0.930	5.733± 1.471	36.533± 2.119
For Expert D dataset		
For training data		
6.017± 1.864	1.250± 1.144	0.250± 0.474
1.550± 1.534	6.833± 2.345	0.767± 1.079
3.167± 1.317	12.167± 2.656	188.000± 3.805
For testing data		
1.667± 1.188	1.717± 1.166	0.933± 1.483
1.500± 1.321	2.533± 1.546	1.317± 1.157
3.100± 1.633	10.500± 3.689	122.733± 4.021
For Expert V dataset		
For training data		
6.683± 3.000	0.950± 1.171	1.067± 1.163
3.100± 1.884	10.750± 2.955	4.333± 2.334
8.617± 2.256	31.983± 4.023	152.517± 4.466
For testing data		
2.367± 1.687	1.600± 1.509	1.117± 1.530
3.333± 2.549	5.733± 2.350	4.517± 2.228
5.900± 2.608	22.983± 3.680	98.450± 4.597

Table 13.1 Statistical analyses on confusion matrices for RMBDT

From Table 13.1, we see that the RMBDT tends to push the samples towards the majority class (class 3), but it is able to review the existence of the minority classes (class 1 &2). When applied on the software metric dataset, we find that RMBDT performs better than the parallel OR/AND neuron-based fuzzy neural network using the single objective optimization method (OA metric), but not better than the POAFNN models trained with the SSA or APR Pareto-based multi-objective optimization process. RMBDT is able to reveal the existence of the minority classes (classes 1 & 2), while the single objective optimization method places great focus on the majority class (class 3) and virtually ignores the appearance of the minority classes.

When comparing RMBDT with the Pareto approach in POAFNN, we find the following:

- RMBDT results are inferior to the results in the Pareto approach when comparing the individual classes separately. Considering Table 13.1 and Tables 12.8 &12.11, we find that the APR and SSA Pareto approach results in Expert A's usability data outperform the RMBDT results in classes 1 & 2, but are lower than RMBDT results in class 3. As our goal is focusing on improving performance in the minority classes, this demonstrates (proves) that Pareto in POAFNN is the better solution.

- We find that the combined results in POAFNN are slightly better than the results in RMBDT. (Refer to Tables 12.14, 12.15, 12.16 and Table 13.1.)

- In terms of extracting knowledge, POAFNN supports the interpretability of the models, while RMBDT does not.

13.3 Cascade Fuzzy Neural Networks vs. Parallel Fuzzy Neural Networks

Here we present the experimental results of COAFNN on the software metric dataset. The experiments are based on models with 3 layers and 5 inputs to each layer. Similar to the experimental setup in Chapter 12, we randomly divided the dataset into training and testing splits 10 times. For each split, we carried out 5 experiments. Based on these 50 experiments, we obtained the statistical analysis results in Table 13.3.

For Expert A dataset		
For training data		
1.620± 0.805	0.000± 0.000	0.000± 0.000
0.020± 0.141	2.640± 1.306	0.240± 0.517
0.860± 0.639	6.560± 1.740	208.060± 1.856
For testing data		
0.060± 0.240	0.020± 0.141	0.060± 0.240
0.120± 0.328	0.220± 0.418	0.740± 1.026
1.320± 0.957	5.560± 1.514	137.900± 1.753
For Expert D dataset		
For training data		
5.020± 1.879	0.920± 0.986	0.100± 0.364
1.960± 1.261	6.800± 2.914	1.460± 1.313
3.720± 1.278	13.480± 2.597	186.540± 5.230
For testing data		
1.660± 1.136	1.160± 1.131	0.280± 0.701
2.460± 2.375	1.380± 1.028	1.500± 1.282
2.180± 1.380	11.260± 3.337	124.120± 3.921
For Expert V dataset		
For training data		
5.780± 2.574	1.300± 1.488	1.340± 1.154
3.080± 2.069	8.340± 3.729	2.460± 1.432
9.140± 2.907	35.760± 3.378	152.800± 4.408
For testing data		
2.380± 1.483	1.640± 1.588	1.060± 1.185
2.900± 1.930	2.760± 1.408	2.340± 1.955
6.720± 2.090	24.200± 3.188	102.000± 4.305

Table 13.2 Statistical distribution of the classes on confusion matrixes for COAFNN

Comparing the results in Table 13.2 with Tables 12.8, 12.11, 12.14, 12.15, and 12.16, we come to the following conclusions:

- Similar to RMBDT, COAFNN has lower performance on individual class classification.
- COAFNN models outperform POAFNN models trained by the OA single objective optimization approach.
- From the viewpoint of the combination of POAFNN Pareto approaches, the two structures do not differ greatly.

- In terms of knowledge extraction, the two structures are based on fuzzy neural networks, so both have the ability to generate rules from models.

- COAFNN and POAFNN are two methods for gaining better performance in both accuracy and interpretability for skewed datasets. The construction of COAFNN models is achieved by a separate process performed on each class. The same number of models will be generated after training. We can combine the results with the maximum value comparison, when considering the classification results. The POAFNN Pareto approach predicts the results on the same model at a specific time, and generates the final class using threshold value. After each class has been classified, the final classification results are obtained by combining the results using maximum value comparison. These two structures provide us with different methods for solving the classification problems for skewed datasets.

- When there are a large number of classes in the dataset, POAFNN is still able to reach the optimal models much more quickly than COAFNN is able to.

- COAFNN will automatically generate the models. POAFNN produces a set of models and the user makes a selection using their own criteria, after the training completes.

Chapter 14 Conclusions and Future Directions

In this thesis, we introduced two categories of logic-driven models, the Reed-Muller Binary Decision Tree (RMBDT) and the Fuzzy Neuron Network (FNN), and we presented the evolutionary development environment. Although RMBDTs do not come with a great deal of interpretability, their structures offer a significant level of accuracy and in this sense are worth exploring, The FNNs discussed here are based on the OR/AND neurons. Making use of the Cascade OR/AND neuron-based FNNs (COAFNNs), we investigated their abilities to extract domain knowledge in a rule-based format. We used the Parallel OR/AND neuron-based FNN (POAFNN) in various classification and knowledge-based models to solve the problems that are characterized by skewed data. In order to properly address the skewness of the data, we adopted multi-objective optimization techniques. The experimental results showed that the Pareto-based multi-objective optimization approach outperforms the single-objective optimization approach. In order to achieve rule bases of higher accuracy, we experimented with different optimization measures considered together, and feel we made some progress along this line of research..

The comparative analysis of the results led us to a number of general conclusions

- All three approaches presented in this study offer some interesting modeling capabilities, producing similar levels of accuracy of the resulting models

- Performance of white-box models is slightly worse than that of black-box models

- The RMBDT models contain rules represented with the use of the exclusive-OR aggregation, and this leads to a far less interpretable character of the rules than those used in other models discussed in this thesis. Rules generated from COAFNN and POAFNN models are expressed in the form of AND/OR terms, which are easier to comprehend. Based on the experiments, we conclude that the extracted rules are quite simple and easy to understand. Furthermore, accuracy of the models is desirable, and we noted that the increased simplicity and understandability of the rules (i.e., simpler rules) results in a fairly limited deterioration in the accuracy of the models.

- The classification abilities of COAFNN and POAFNN on skewed datasets are very similar. These two approaches confront the essence of the problem in different ways. When we use COAFNN models, we construct models separately; rules are generated after the models have been pruned. By selecting the Pareto-based multi-objective optimization approach, a number of POAFNN models could be built during the training process, and we are able to select appropriate ones from these models, to generate rules.

- The training speed of POAFNN using the Pareto approach is faster than that of COAFNN, when the number of classes is large.

- For classification involving skewed data, POAFNN is a better choice than the RMBDT and COAFNN models.

We have also used other data models, such as Fuzzy Decision Trees, Product of Maxterm Fuzzy Neural Networks, and Sum of Minterm Fuzzy Neural Networks. In these models, promising results were also obtained in extracting knowledge from the raw data. Future issues worth exploring along this line are summarized as follows:

- Combination of different modeling structures

  We have tried different kinds of models that are useful in extracting knowledge from the raw dataset. The next step in improving the performance of the modeling system could be the attempt to combine 2 or more different modeling structures to form a new modeling system. With a proper combining technique, it would be possible to find models that could provide improved accuracy and interpretability.

- Evaluation of rule confidence levels

Because of time constraints, we did not do a great deal of work on how to extract rules with a higher confidence level, with the modeling structures we proposed above. This topic could provide an exciting continuation of the research in this area.

References to Part II

[2_1] D. W. Aha, D. Kibler, M. K. Albert, Instance-Based Learning Algorithms, *Machine Learning*, vol. 6, pp. 37-66, 1991.

[2_2] T. Bäck, U. Hammel, and H.-P. Schwefel. Evolutionary computation: comments on the history and current state. *IEEE Trans. on Evolutionary Computation*, 1(1):3–17, 1997.

[2_3] A. G. Barto, R. S. Sutton, and C. W. Andersson, Neuron like adaptive elements that can solve difficult learning control problems. *IEEE Trans. Systems, Man, and Cybernetics* 13: 834-846, 1983.

[2_4] D. E. Bell, R. L. Keeney, and H. Raiffa (Eds.), *Conflicting objectives in decision*. John Wiley & Sons, 1977.

[2_5] L. Chambers, *The practical handbook of genetic algorithms : applications*, 2nd ed. Chapman & Hall/CRC, Florida, 2001.

[2_6] P. Clark and T. Niblett, The CN2 Induction Algorithm. *Machine Learning*, vol. 3, pp. 261-283, 1989.

[2_7] C. A. Coello, D. A. VanVeldhuizen, and G. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Boston, MA: Kluwer Academic Publishers, 2002.

[2_8] T. M. Cover and P. E. Hart, Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, vol. 13, pp. 21-27, 1967.

[2_9] L. Davis, *Handbook of Genetic Algorithms*. van Nostrand Reinhold, New York, 1991.

[2_10] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, 2001[2_11] G. B. Fogel and D. W. Corne, *Evolutionary computation in bioinformatics*. Morgan Kaufmann Publishers, San Francisco, 2003.

[2_12] D.E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[2_13] P. Hart, The Condensed Nearest Neighbor Rule. *IEEE on Information Theory*, vol. 14, pp. 515-516, 1968.

[2_14] F. Herrera, M. Lozano and J. L. Verdegay, Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. *Artificial Intelligence Review*, Volume 12, Issue 4, pp. 265 – 319, August 1998.

[2_15] http://www.ics.uci.edu/~mlearn/MLRepository.html

[2_16] R. P. Lippmann, An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, vol. 3, no. 4, pp. 4 - 22, April 1987.

[2_17] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed.* Heidelberg, Germany: Springer-Verlag, 1996.

[2_18] S. Mitra, K. M. Konwar, S. K. Pal: Fuzzy decision tree, linguistic rules and fuzzy knowledge-based network: generation and evaluation. *IEEE Transactions on Systems, Man, and Cybernetics*, Part C 32(4): 328-339 (2002).

[2_19] F. Neelamkavil, *Computer Simulation and Modeling*. John Wiley & Sons, 1987.

[2_20] C. Olaru., L. Wehenkel, A complete fuzzy decision tree technique, *Fuzzy Sets and Systems*. Volume 138 , Issue 2, Pages: 221 - 254, 2003.

[2_21] S. K. Pal, S. Mitra and P. Mitra, Rough Fuzzy MLP: Modular evolution, rule generation and evaluation, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, pp. 14-25, 2003.

[2_22] W. Pedrycz, Fuzzy neural networks and neurocomputations, *Fuzzy Sets and Systems*, v.56 n.1, p.1-28, May 1993.

[2_23] W. Pedrycz, *Fuzzy control and fuzzy systems, 2nd extended edition*. Taunton, Somerset, England : Research Studies Press ; New York : Wiley, 1993.

[2_24] W. Pedrycz, F. Gomide, *An Introduction to Fuzzy Sets, Analysis and Design*. MIT Press, 1998.

[2_25] W. Pedrycz and M. Reformat, Rule-Based Modeling of Nonlinear Relationships. *IEEE Transactions on Fuzzy Systems*, Vol.5, No.2, pp.256-269, May 1997.

[2_26] B. W. Porter, R. Bareiss, and R. C. Holte, Concept Learning and Heuristic Classification in Weak-Theory Domains, Artificial Intelligence, vol. 45, nos. 1 and 2, 1990.

[2_27] J.R. Quinlan, Discovering rules by induction from large collections of examples. In D. Michie (Ed.), *Expert systems in the micro electronic age*. Edinburgh University Press, 1979.

[2_28] J. R. Quinlan, Induction of Decision Trees, Machine Learning, vol. 1, pp. 81-106, 1986.

[2_29] S. Salzberg, A Nearest Hyperrectangle Learning Method. *Machine Learning*, vol. 6, pp. 277-309, 1991.

[2_30] H. P. Schwefel, *Evolution and optimum seeking*. John Wiley & Sons, 1995.

[2_31] M. Umanol, H. Okamoto, I. Hatono, etc. Fuzzy decision trees by fuzzy ID3 algorithm and its application to diagnosis systems, *Fuzzy Systems, IEEE World Congress on Computational Intelligence. Proceedings of the Third IEEE Conference on*, Page(s):2113 - 2118 vol.3, June 1994.

[2_32] R.A. Vivanco, N.J. Pizzi, Finding optimal software metrics to classify software maintainability using a parallel genetic algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference*, June 26-30, Seattle, USA, pp. 1388-1399, 2004.

[2_33] D. Wettschereck and T. G. Dietterich, An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms. *Machine Learning*, Vol. 19, No. 1, pp. 5-28, 1995.

[2_34] D. Wettschereck, A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm. *The Proceedings of the 7th European Conference on Machine Learning*, pp. 323-335, 1994.

[2_35] B. Widrow and R. Winter, Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition. *IEEE Computer Magazine*, pp. 25-39, March 1988.

[2_36] B. Widrow and M. A. Lehr, 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415-1441, September 1990.

[2_37] L. A. Zadeh, Fuzzy sets. *Information and Control* 8: 338-353, 1965.

References to Part III

[3_1] A. E. A. Almaini, *Electronic logic systems*. Prentice-Hall, 1994.

[3_2] A.E.A. Almaini. and N. Zhuang, Using genetic algorithms for the variable ordering of Reed-Muller binary decision diagrams. *Microelectronics Journal*, 26 (1995) Pages: 471-480.

[3_3] C.H. Chang and B.J. Falkowski, Flexible optimization of fixed polarity Reed-Muller expansions for multiple output completely and incompletely specified Boolean functions. *Proceedings of the IEEE/IEICE Asia and South Pacific conference on design automation*, ASP-DAC'95, Makuhari, Chiba, Japan, Aug. 1995, pp. 335–340.

[3_4] Y.L. Csank, M.A. Perkowski, and I. Schaefer, Canonical restricted mixed-polarity exclusive-OR sums of products and the efficient algorithm for their minimization. *IEEE Proc.-Comput. Digit. Tech.*, 1993, 140, (1), pp. 69–77.

[3_5] L. Davis, *Handbook of Genetic Algorithms*. van Nostrand Reinhold, New York, 1991.

[3_6] R. Drechsler, *Evolutionary Algorithms for VLSI CAD*. Kluwer Academic Publisher, Dordrecht, 1998.

[3_7] L.J. Eshelman, J.D. Scahffer, Real-coded genetic algorithms and interval-schemata. In: L. Darrel Whitley (ed.), *Foundations of Genetic Algorithms 2*, San Mateo: Morgan Kaufmann, 1993, 187-202.

[3_8] J.L. Fernández-Villacañas Martín, M. Shackleton, Investigation of the importance of the genotype–phenotype mapping in information retrieval. *Future Generation Computer Systems*, 19, 2003,55–68.

[3_9] D.E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[3_10] D. Green, *Modern Logic Design*. Addison Wesley, 1986.

[3_11] U. Kebschull, E. Schubert and W. Rosenstiel, Multilevel logic synthesis based on functional decision diagrams. In: *Proc. European Conf. on Design Automation, EDAC'92 (1992)*, pp. 43–47.

[3_12] K.M. Kim, B.J. Lee, K. Lyou, G.T. Park, Design of a binary decision tree using the genetic algorithm and K-means algorithm for recognition of the defect patterns of cold mill strip. *Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE '99. 1999 IEEE International*, Volume: 2 , 22-25 Aug. 1999, Pages:1081 - 1085.

[3_13] *G.S. Lehal, C. Singh;* A Gurmukhi script recognition system. *Pattern Recognition, 2000. Proceedings of the 15th International Conference on*, Volume: 2 , 3-7 Sept 2000, Pages:557 - 560.

[3_14] L. McKenzie, A.E.A. Almaini,, J.F. Miller, and P. Thomson, Optimisation of Reed-Muller logic functions. *Int. J. Electron.*, 1993, 75, (3), pp. 451–466.

[3_15] L. McKenzie, L. Xu and A. Almaini, Graphical representation of generalised Reed-Muller expansions. In: Proc. *IFIP WG 10.5 Workshop On Applications of the Reed-Muller Expansion in Circuit Design* , pp. 181–187, 1993.

[3_16] W. Pedrycz, F. Gomide, *An Introduction to Fuzzy Sets; Analysis and Design*. MIT Press, 1998.

[3_17] A. Sarabi, P.F. Ho, K. Iravani, W.R. Daasch and M.A. Perkowski, Minimal multi-level realisation of switching functions based on Kronecker functional decision diagrams. In: *Proc. Int. Workshop on Logic Synthesis*, pp. 3a1–3a6, 1993.

[3_18] E.C. Tan, P.K.K. Loh, Processor-farm model for parallel computation of fixed-polarity Reed-Muller expansions. *TENCON 2000. Proceedings*, Volume: 2 , 24-27 Sept. 2000 Pages:1 - 3 vol.2.

[3_19] R.A. Vivanco, N.J. Pizzi, Finding optimal software metrics to classify software maintainability using a parallel genetic algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference*, June 26-30, Seattle, USA, 1388-1399 (2004)

[3_20] L. Xu, A.E.A. Almaini, J.F. Miller and L. McKenzie, Reed-Muller Universal logic module networks. In: *IEE Proc. Pt E, Computers Digital Techniques* 140 (1993), pp. 105–108 (2).

[3_21] L. Xu and L. McKenzie, *Constructing* Reed-Muller universal logic module networks from Reed-Muller binary decision diagrams. *Microelectronics Journal*, 27 (1996) Pages: 23-36.

[3_22] L. Xu and L. McKenzie, Multi-level optimisation of fixed polarity Reed-Muller expansions using Reed-Muller binary decision diagrams. In: Proc. *IEE Colloquium on Synthesis and Optimisation of Logic* Systems (1994), pp. 3/1–3/4.

[3_23] M. Yoshikawa, H. Shindo, R. Nishii,S. Taaaka, A fully automated design of binary decision tree for land cover classification, Geoscience and Remote Sensing Symposium. *IGARSS '95. 'Quantitative Remote Sensing for Science and Applications', International*, Volume: 3 , 10-14 July 1995 Pages:1921 - 1923 vol.3, 1995.

References to Part IV

[4_1] R. L. Ashenhurst, The decomposition of switching functions. *Proc. Int. Symp. Theory of Switching*, Harvard University, Cambridge, Massachusetts (1959), 74-116.

[4_2] K. V. S. Bhat, Comments "on simplification of fuzzy switching functions," *IEEE Trans. Systems, Man & Cybemet.* SMC-10, October, 1980, 637-640.

[4_3] J. C. Biocha, T. Ibaraki, Kazuhisa Makino, Minimum self-dual decompositions of positive dual-minor Boolean functions. *Discrete Applied Mathematics*, 96, 1999, 307-326.

[4_4] L. Davis, *Handbook of Genetic Algorithms*. van Nostrand Reinhold, New York, 1991.

[4_5] L. J.Eshelman, J.D. Scahffer, Real-coded genetic algorithms and interval-schemata. In:

[4_6] L. Darrel Whitley (ed.), *Foundations of Genetic Algorithms 2*. San Mateo: Morgan Kaufmann, 1993, 187-202.

[4_7] J.L. Fernández-Villacañas Martín, M. Shackleton, Investigation of the importance of the genotype–phenotype mapping in information retrieval. *Future Generation Computer Systems*, 19, 2003,55–68.

[4_8] L. Fu, *Neural Networks in Computer Intelligence*, McGraw-Hill, 1994.

[4_9] D.E. Goldberg, *Genetic Algorithms in Search, Optimization& Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[4_10] K. Hirota, W. Pedrycz, OR/AND neuron in modeling fuzzy set connectives, *IEEE Trans. on Fuzzy Systems*, 2, 1994, 151-161.

[4_11] K. Hirota, W. Pedrycz, A distributed model of fuzzy set operators, *Fuzzy Sets and Systems*, 68, 1994, 157-170.

[4_12] A. Kandel, S. C. Lee, *Fuzzy Switching and Automata: Theory and Applications*, Crane, Russak & Co., Inc., New York, 1979.

[4_13] A. Kandel, Inexact switching logic, *IEEE Trans. Systems Man and Cybemet.*, SMC-6,1976, 215-219.

[4_14] A. Kandel, On the minimization of incompletely specified fuzzy functions, *Inf. Control*, 26, 1974, 141-153.

[4_15] C. G. Looney, *Pattern Recognition Using Neural Networks*, Oxford University Press, 1997

[4_16] Z. Michalewicz, *Genetic Algorithms+ Data Structures =Evolution Programs*, Springer, Berlin, 1994.

[4_17] W. Pedrycz, *Fuzzy Control and Fuzzy Systems (2nd edition)*, Taunton, NY: Research Studies Press/J. Wiley, 1993

[4_18] W. Pedrycz, Fuzzy neural networks and neurocomputations, *Fuzzy Sets and Systems*, v.56 n.1, p.1-28, May 1993.

[4_19] W. Pedrycz, Neurocomputations in relational systems, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Volume 13, Issue 3, pp. 289-297, March 1991

[4_20] W. Pedrycz, A. Rocha, Fuzzy-set based models of neurons and knowledge-based networks, *IEEE Trans. on Fuzzy Systems*, Volume 1, Issue 4, pp. 254-266, Nov. 1993.

[4_21] W. Pedrycz, A. Rocha, Knowledge-based neural networks, *IEEE Trans. on Fuzzy Systems*,1, 1993, 254-266.

[4_22] W. Pedrycz, F. Gomide, *An Introduction to Fuzzy Sets; Analysis and Design*. MIT Press, 1998.

[4_23] G. Rudolph, Convergence analysis of canonical genetic algorithms, *IEEE Trans. Neural Networks*, 5, 1994, 96- 101.

[4_24] G. W. Schwede, A. Kandel, Fuzzy maps, *IEEE Trans. Systems, Man and Cybernet.*, SMC-7, Sept. 1977, 669-674.

[4_25] H. J. Zimmermann, P. Zysno, Latent connectives in human decision making, *Fuzzy Sets and Systems*, 4, 1980, 37-51.

References to Part V

[5_1] D. E. Bell, R. L. Keeney, and H. Raiffa (Eds), *Conflicting objectives in decision*, John Wiley & Sons, 1977.

[5_2] P. Clark and R. Boswell, Rule induction with CN2: Some recent improvements, *Proceedings of the Fifth European Working Session on Learning*, pp. 151-163, 1991.

[5_3] C. A. Coello, A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[5_4] C. A. Coello, *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*, PhD thesis, Department of Computer Science, Tulane University, New Orleans, LA, April 1996.

[5_5] C. A. Coello, D. A. Van Veldhuizen and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academic Publishers, New York, 2002.

[5_6] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[5_7] M. Farina, *Cost-effective Evolutionary Strategies for Pareto Optimal Front Approximation in Multiobjective Shape Design Optimization of Electromagnetic Devices*, PhD Thesis, Department of Electrical Engineering, University of Pavia, Italy, 2001.

[5_8] C. M. Fonseca and P. J. Fleming, An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[5_9] J. Guddat, F. G. Vasquez, K. Tammer and K. Wendler, *Multiobjective and Stochastic Optimization Based on Parametric Optimization*. Akademie-Verlag, Berlin, 1985.

[5_10] R. Kohavi AND F. Provost, Glossary of terms, *Journal of Machine Learning*, 30, 2/3, 271–274. Editorial,1998.

[5_11] K. Miettinen, *Nonlinear Multiobjective Optimization*. Boston: Kluwer, 1999.

[5_12] Y. Sawaragi, H. Nakayama, T. Tanino, *Theory of Multiobjective Optimization*, Academic Press, 1985

[5_13] J. D. Schaffer, Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 93–100, Lawrence Erlbaum, Hillsdale, New Jersey, 1985.

[5_14] P. Serafini (Editor), *Mathematics of Multi Objective Optimization*, Springer-Verlag, Wien-New York, 1985.

[5_15] D. A. Van Veldhuizen and G. B. Lamont, Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art, *Evolutionary Computation*, 8(2):125-147, 2000.

[5_16] E. Zitzler, *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.

[5_17] E. Zitzler, M. Laumanns, L. Thiele, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review, *IEEE Transactions on Evolutionary Computation* 7(2), 2003, 117-132.

[5_18] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Evolutionary Algorithm, *IEEE Transactions on Evolutionary Computation*, 3(4):257-271, 1999.