

# **UNCERTAINTY-BASED PATTERN CLASSIFICATION BY MODULAR NEURAL NETWORKS**

*A THESIS*

*submitted by*

**MANISH SARKAR**

*for the award of the degree of*

**DOCTOR OF PHILOSOPHY**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS  
CHENNAI 600036, INDIA**

**OCTOBER 1998**

## THESIS CERTIFICATE

This is to certify that the thesis entitled Uncertainty-Based Pattern Classification by Modular Neural Networks, submitted by **Manish Sarkar**, to the Indian Institute of Technology, Madras, for the award of the degree of Doctor of Philosophy, is a **bonafide** record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Madras 600036

(B. YEGNANARAYANA)

Date:

Place: Madras 600036

(DEEPAK KHEMANI)

Date:

# ACKNOWLEDGEMENTS

First and foremost, I wish to place on record my gratitude to all the members of *Neural Networks Laboratory* (alias Microprocessor Laboratory), Department of Computer Science & Engineering, IIT Madras. I am grateful to them for maintaining a stimulating research environment in the laboratory. Discussions and **brainstorming** sessions with various members of the laboratory like Appan, Devarajan, Gayathry, **Hashim**, Ilango, Murthy, Nallu, Neeharika, **Pavan**, Poongudi, Prakash, Raghu, **Ramkrishna** Reddy, Sanjay, Sridharan, **Usha** and Vijaya during the course of my research was very productive. I am grateful to the institute for extending to me the facilities of the Neural Networks Laboratory without which my research would have been stillborn.

I express my gratitude to Prof. B. Yegnanarayana and Dr. Deepak Khemani who have inspired me during my association with them. I am thankful to my doctoral committee members for their time and effort in reviewing the progress in this work. I would like to thank Dr. **Sukhendu** Das for the helpful discussions that I had with him.

I am indebted to the Department of Atomic Energy, Government of India, for supporting me in my research work with the Dr. K. S. Krishnan Senior Research Fellowship.

I express my gratitude to the Department of Computer Science and Engineering for giving me an opportunity to carry out my research. I shall remember **Anitha**, **Hariram**, **Hemant**, Iqbal, Joshi, Karna, **Mathew**, Murthy, Sain, Sankara **Raman**, Sudha, Sudhakar, Tamil, Yaji for their kind cooperation and friendship.

It has been real fun staying at IIT Madras. The sylvan surroundings, "quark", the endless sessions at Velacheri and Taramani tea shops will be etched into my memory forever. I shall never forget the constructive/destructive chats that I had with Sids, Bijon and Prabir while sipping tea at midnight.

I cherish the great time I had with the **Tapti** Hostel people; **Soumen** for our countless hours of discussion on every topic in the universe, Sanjay for all the fun and excitement we had together, Arunachalam for his magic with mathematics, Sids without whom life would have been a bore.

Finally, I would never have been able to survive the  $N$  years this endeavour took without moral support ( $+$  perturbation) of my good friends staying outside the institute. These are the people who made my life a controlled stochastic process.

# ABSTRACT

The objective of this study is to demonstrate the significance of incorporating *a priori* knowledge of the problem in a pattern classification task. The issues of identification, representation and application of knowledge in the form of fuzzy, rough and probabilistic uncertainties are addressed to develop a new pattern classification methodology. This thesis demonstrates the significance of modular classification approach to deal with uncertainties effectively in pattern classification tasks. The performance of the proposed approach is illustrated for the opening bid problem in the game of Contract Bridge.

Building classifiers involves capturing the similarity among the training patterns and assigning labels for the group of similar patterns. Capturing the similarity among patterns becomes complicated when a training pattern belongs to more than one class, i.e., the output classes are overlapping. Thus *fuzzy uncertainty* appears in form of similarity and overlap. Due to the lack of details, two input patterns may appear similar whereas the class labels may not be same. The *one-to-many* relationship between the inputs and outputs results in *rough uncertainty*. If the occurrence of the training patterns in the neighborhood region is small, then due to *probabilistic uncertainty* assigning the class labels is difficult. Thus building classifiers essentially involves dealing with fuzzy, rough and probabilistic uncertainties. In the opening bid problem of Contract Bridge game, the input is a hand pattern and the output is the class label for the input hand. In this problem, obtaining a particular hand pattern is probabilistic. The output classes are fuzzy. The absence of unique class labels for input hands creates rough uncertainty.

In this thesis artificial neural networks are employed as classifiers. Experimentally it was observed that it is difficult to deal with the issues in uncertainties for the opening bid problem. Hence, modular neural networks are explored. Modular approach partitions the classification task into three subclassification tasks, solves each subclassification task, and eventually integrates the results to obtain the final classification result. In other words, partitioning of the classification task is carried out such that each subproblem can be solved in a module by exploiting the local uncertainties and the results of all the modules can be combined by exploiting the global uncertainties.

The performance of each module can be improved by giving importance to the features based on their class discrimination capability for the output classes present in the module. Since both roughness and fuzziness are present and the input features are discrete, the uncertainty in assigning class labels for a given pattern based on each feature is treated as *rough-fuzzy uncertainty*. The more important a feature is for classification, the less is the rough-fuzzy uncertainty associated with that feature. A *rough-fuzzy entropic measure* is proposed to quantify the importance of each feature. Using the importance measure, the input hands are biased to generate modified feature vectors corresponding to each

module.

One approach of assigning class labels for the modified feature vectors is through direct classification. It involves partitioning the modified feature space of a module into several fuzzy output classes. Feedforward neural networks are used to obtain the class labels. Backpropagation learning algorithm with *fuzzy objective functions* are used to train the networks. The networks are configured optimally using *evolutionary programming*. After training if a new input pattern is presented to the network, then the network yields the output as the fuzzy membership value of the input to the output classes.

An alternative approach to assign the class labels on the modified feature vector is clustering. In this approach, modified feature vectors are clustered, and each cluster is labelled with class labels. Since the clusters are fuzzy, the modified feature vectors are clustered using an evolutionary programming-based fuzzy clustering algorithm. The labelling of the clusters is complicated because two patterns from the same cluster may belong to entirely different classes. The labelling of the clusters is done using a *fuzzy-rough neural network*. It captures the fuzzy uncertainty present in the clusters and rough uncertainty between the clusters and the class labels. If a new input pattern is presented to the network after training, it yields the output as a class confidence value in terms of *fuzzy-rough membership* value corresponding to the input pattern. In the opening bid problem, experimentally it was decided to use feedforward neural networks with backpropagation algorithm to construct the module for the first level bids and fuzzy-rough neural networks to construct the remaining two modules for the second and third level bids.

When the original classification task is distributed among modules, the modules have been trained and configured to deal with the uncertainties locally. But the final class labels, indicated by the outputs of the modules, may be conflicting. To arrive at the classification result from the conflicting outputs, *Sugeno's fuzzy integral* is used. The outputs of the modules are treated as evidence, and they are fused in a nonlinear fashion based on their importance. The importance of each evidence is determined using the fuzzy-roughness associated with the evidence. The final class label of an input is the output class corresponding to the maximum value of the fuzzy integral.

The main contribution of the thesis are: (1) Demonstrating the significance of uncertainty in pattern classification problems, (2) providing a review on issues in *uncertainty-driven* pattern classification tasks, (3) application of modular neural networks to deal with fuzzy, rough and probabilistic uncertainties, (4) use of rough-fuzzy sets to determine the importance of each feature for classification, (5) development of backpropagation learning algorithms based on various fuzzy objective functions, (6) proposing rough-fuzzy membership functions and fuzzy-rough membership functions to construct fuzzy-rough neural networks, and (7) use of fuzzy-rough sets in fuzzy integral to measure the importance of each module.

**Keywords:** Classification, uncertainty, modular neural networks, feedforward neural networks, fuzzy sets, rough sets, evolutionary programming, clustering, fuzzy-rough membership functions, fuzzy-rough neural networks and fuzzy integral.

# CONTENTS

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
LIST OF FIGURES	iiix
LIST OF TABLES	x
NOTATIONS	xii
ABBREVIATIONS	xiv
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Background: Problem Solving with Uncertainty . . . . .	1
1.2 Issues in Pattern Classification . . . . .	2
1.3 Scope: Study of Uncertainties in Opening Bid Problem in Bridge Game . .	4
1.4 Proposed Approach for Capturing the Reasoning Process in Opening Bid Problem . . . . .	6
1.5 Organization of the Thesis . . . . .	7
<b>2 MODELING PATTERN CLASSIFICATION PROBLEMS</b>	<b>11</b>
2.1 Introduction . . . . .	11
2.2 A Review on Pattern Classification . . . . .	12
2.2.1 How Human Classification Mechanism is Mimicked on Machines . .	12
2.2.2 Process Description . . . . .	16
2.2.2.1 Symbolic process description . . . . .	17
2.2.2.2 Numerical process description . . . . .	18
2.2.3 Feature Analysis . . . . .	19
2.2.4 Structure Analysis . . . . .	20
2.2.5 Abstraction: Search for Structure . . . . .	22
2.2.5.1 Deterministic classifiers: Crisp rule base . . . . .	23
2.2.5.2 Statistical classifiers . . . . .	25
2.2.5.3 Fuzzy classifiers . . . . .	31

2.2.5.4	Rough classifiers . . . . .	37
2.2.5.5	Hybrid classifiers . . . . .	38
2.2.6	Generalization . . . . .	46
2.2.7	Conclusion . . . . .	50
2.3	Modular Classifiers . . . . .	53
2.3.1	Background of Modular Classifiers . . . . .	<b>53</b>
2.3.2	Advantages of Modular Classifiers . . . . .	54
2.3.3	Issues in Modular Approach . . . . .	55
2.3.4	Types of Modular Classifiers . . . . .	56
2.4	Opening Bid Problem in Contract Bridge as a Pattern Classification Problem	62
<b>3</b>	<b>PRELIMINARY STUDIES ON BIDDING PROBLEM USING ARTIFICIAL NEURAL NETWORKS</b>	<b>64</b>
3.1	Introduction . . . . .	64
3.2	Representation of Opening Bid Problem on Machines . . . . .	66
3.2.1	Data Generation and Collection . . . . .	66
3.2.2	Representation of Input Patterns . . . . .	67
3.3	Studies on Network Architecture and Training . . . . .	71
3.4	Summary . . . . .	79
<b>4</b>	<b>IMPORTANCE OF INPUT FEATURES IN CLASSIFICATION: ROUGH-FUZZY SET THEORETIC APPROACH</b>	<b>81</b>
4.1	Introduction . . . . .	81
4.2	Background of Fuzzy K-Nearest Neighbors Algorithm . . . . .	83
4.3	Proposed Method . . . . .	85
4.3.1	Criterion Function . . . . .	85
4.3.2	Possibilistic K-Nearest Neighbors Algorithm . . . . .	89
4.3.3	Optimization Technique and Weight Update . . . . .	90
4.4	Results and Discussion . . . . .	91
4.5	Summary . . . . .	98
<b>5</b>	<b>DESIGN OF CLASSIFIER MODULES THROUGH DIRECT CLASSIFICATION</b>	<b>102</b>
5.1	Introduction . . . . .	102
5.2	Feedforward Neural Network Classifiers: Backpropagation Learning Algorithm with Fuzzy Objective Functions . . . . .	103
5.2.1	Architecture of Feedforward Neural Networks . . . . .	104
5.2.2	Training of Feedforward Neural Networks . . . . .	104
5.2.3	Testing of Feedforward Neural Networks . . . . .	111
5.2.4	Results and Discussion . . . . .	112

5.3	Configuration of Feedforward Neural Networks Using Evolutionary Programming-Based Hybrid Technique . . . . .	115
5.3.1	Evolutionary Programming in Network Configuration . . . . .	118
5.3.2	Implementation Issues . . . . .	119
5.3.2.1	Fitness function . . . . .	119
5.3.2.2	Replication of parents . . . . .	119
5.3.2.3	Mutation . . . . .	121
5.3.3	Results and Discussion . . . . .	124
5.4	Summary . . . . .	129
6	<b>DESIGN OF CLASSIFIER MODULES THROUGH CLUSTERING</b>	132
6.1	Introduction . . . . .	132
6.2	Evolutionary Programming-Based Fuzzy Clustering . . . . .	133
6.2.1	Background of Fuzzy K-Means Clustering . . . . .	135
6.2.2	Embedding Evolutionary Programming in Fuzzy Clustering . . . . .	138
6.2.3	Implementation Issues . . . . .	140
6.2.3.1	Fitness function . . . . .	140
6.2.3.2	Replication of parents . . . . .	140
6.2.3.3	Mutation . . . . .	141
6.2.3.4	Modified fuzzy K-means algorithm . . . . .	143
6.2.4	Results and Discussion . . . . .	143
6.3	Fuzzy-Rough Neural Networks . . . . .	149
6.3.1	Root of Fuzzy-Rough Neural Networks . . . . .	150
6.3.2	Architecture of Fuzzy-Rough Neural Networks . . . . .	150
6.3.3	Training and Testing of Fuzzy-Rough Neural Networks . . . . .	152
6.3.4	Results and Discussion . . . . .	153
6.4	Summary . . . . .	154
7	<b>FUSION OF CLASSIFICATION RESULTS</b>	157
7.1	Introduction . . . . .	157
7.2	Background . . . . .	159
7.2.1	Fuzzy Measure . . . . .	159
7.2.2	Fuzzy Integral . . . . .	159
7.3	Modular Networks with Proposed Fusion Technique . . . . .	162
7.3.1	Architecture of Modular Networks . . . . .	162
7.3.2	Training of Modular Networks . . . . .	162
7.3.2.1	Training of subnetworks . . . . .	163
7.3.2.2	Pattern matching . . . . .	164
7.3.3	Testing of Modular Networks . . . . .	167



7.4	Results and Discussion . . . . .	169
7.5	Summary . . . . .	171
<b>8</b>	<b>SUMMARY AND CONCLUSIONS</b>	<b>173</b>
8.1	Summary of the Thesis . . . . .	173
8.2	Contribution of the Thesis . . . . .	176
8.3	Conclusion of the Thesis . . . . .	176
8.4	Issues Related to This Thesis Work for Further Study . . . . .	177
 <b>APPENDIX</b>		
<b>A</b>	<b>CONTRACT BRIDGE GAME: ISSUES</b>	<b>179</b>
<b>B</b>	<b>EVOLUTIONARY PROGRAMMING AND ROUGH SETS: BASICS</b>	<b>182</b>
B.1	Background of Evolutionary Programming . . . . .	182
B.2	Background of Rough Sets . . . . .	183
<b>C</b>	<b>ROUGH-FUZZY MEMBERSHIP FUNCTIONS</b>	<b>186</b>
C.1	Basics of Rough-Fuzzy Sets . . . . .	187
C.2	Definition of Rough-Fuzzy Membership Functions . . . . .	187
C.3	Properties of Rough-Fuzzy Membership Functions . . . . .	187
<b>D</b>	<b>FUZZY-ROUGH MEMBERSHIP FUNCTIONS</b>	<b>191</b>
D.1	Background of Fuzzy-Rough Sets . . . . .	192
D.2	Definition of Fuzzy-Rough Membership Functions . . . . .	192
D.3	Properties of Fuzzy-Rough Membership Functions . . . . .	192
 <b>BIBLIOGRAPHY</b>		 <b>198</b>
 <b>PUBLICATIONS</b>		 <b>210</b>

# LIST OF FIGURES

<b>1.1</b>	A typical modular neural network with $S$ modules . . . . .	<b>7</b>
<b>1.2</b>	Flow of ideas across the thesis . . . . .	<b>8</b>
<b>2.1</b>	Relationships among different topics discussed in pattern classification . . . . .	<b>13</b>
<b>2.2</b>	Different steps involved in pattern classification . . . . .	<b>16</b>
<b>2.3</b>	A three layered feedforward neural network . . . . .	<b>29</b>
<b>2.4</b>	A typical radial basis function neural network . . . . .	<b>30</b>
<b>2.5</b>	A typical probabilistic neural network . . . . .	<b>31</b>
<b>2.6</b>	Fuzzy membership functions for fuzzy sets and fuzzy numbers . . . . .	<b>32</b>
<b>2.7</b>	Possibilistic vs. constrained fuzzy and crisp membership assignments . . . . .	<b>35</b>
<b>2.8</b>	Psychological uncertainty . . . . .	<b>52</b>
<b>2.9</b>	Types of modular classifiers . . . . .	<b>56</b>
<b>2.10</b>	Construction of modular classifiers based on the concept of problem decomposition . . . . .	<b>58</b>
<b>2.11</b>	Construction of modular classifiers based on the concept of class decomposition . . . . .	<b>58</b>
<b>2.12</b>	Different varieties of modular classifiers . . . . .	<b>60</b>
<b>3.1</b>	Input representations for opening bid problem . . . . .	<b>70</b>
<b>4.1</b>	Fuzzy K-nearest neighbors algorithm . . . . .	<b>84</b>
<b>4.2</b>	Flow diagram to determine the weightages of the features . . . . .	<b>91</b>
<b>4.3</b>	Algorithm to determine the importance of input features using rough-fuzzy entropy . . . . .	<b>92</b>
<b>4.4</b>	Artificially generated input data set for the first experiment . . . . .	<b>94</b>
<b>4.5</b>	Change of total rough-fuzzy entropy with the number of iterations for the data set shown in Fig. 4.4 . . . . .	<b>95</b>
<b>4.6</b>	Artificially generated input data set for the second experiment . . . . .	<b>96</b>
<b>4.7</b>	Change of total rough-fuzzy entropy with the number of iterations for the data set shown in Fig. 4.6 . . . . .	<b>96</b>
<b>4.8</b>	Change of total rough-fuzzy entropy for the data set with first level bids. . . . .	<b>99</b>
<b>5.1</b>	A typical fully connected feedforward neural network . . . . .	<b>104</b>
<b>5.2</b>	Top: No. of iterations vs. fuzzy mean square error of an FFNN. Bottom: No. of iterations vs. fuzzy cross entropy of an FFNN . . . . .	<b>114</b>
<b>5.3</b>	Configuration of feedforward neural networks using evolutionary programming . . . . .	<b>120</b>
<b>5.4</b>	Spread vs. diversity of a population of networks . . . . .	<b>125</b>

	Top: No. of generations vs. fuzzy mean square error of an FFNN. Bottom: No. of generations vs. no. of hidden nodes . . . . .	127
5.6	Two equivalent networks with different genotype representations . . . . .	130
6.1	Fuzzy K-means algorithm . . . . .	137
6.2	Evolutionary programming-based fuzzy clustering algorithm . . . . .	139
6.3	Modified fuzzy K-means algorithm . . . . .	144
6.4	Top: Eight different Gaussian distributions are used to generate a data set artificially. Bottom: Clustered output by the proposed clustering algorithm	146
6.5	Top: No. of iterations vs. fitness curve for the best set of clusters in the proposed clustering algorithm. Bottom: No. of iterations vs. no. of clusters for the best member of the population. . . . .	147
6.6	Clustered output by fuzzy K-means algorithm for the data set shown in Fig. 6.4. . . . .	148
6.7	A typical fuzzy-rough neural network with three input nodes, four hidden nodes and two output nodes . . . . .	151
6.8	Equivalent sets of clusters with different genotype representations . . . . .	155
7.1	A modular network with $S$ modules or subnetworks . . . . .	163
7.2	Training of the proposed modular neural network . . . . .	168
7.3	Testing of the proposed modular neural network . . . . .	169
B.1	Rough sets in one and two dimensional domains. . . . .	184

# LIST OF TABLES

2.1	Relative merits of artificial neural networks, fuzzy logic, evolutionary computation and rough sets . . . . .	39
2.2	Variations in players' bids . . . . .	63
3.1	Distribution of hand patterns . . . . .	68
3.2	Sample hands generated by the shuffling program . . . . .	69
3.3	Mean square error of the 1-Level network after training . . . . .	72
3.4	Bids made by the 1-Level network . . . . .	73
3.5	The bids made by the 1-Level and 2-NT networks for some hands . . . . .	75
3.6	Mean square error of the 2-NT network after training . . . . .	78
3.7	Bids made by the 2-NT network . . . . .	78
3.8	Classification performance of FFNNs with BP algorithm for first level bids	80
3.9	Classification performance of FFNNs with BP algorithm for second level bids	80
3.10	Classification performance of FFNNs with BP algorithm for third level bids	80
4.1	Importance of features against the number of iterations for the data set shown in Fig. 4.4 . . . . .	93
4.2	Importance of features against the number of iterations for the data set shown in Fig. 4.6 . . . . .	97
4.3	Classification performance of FFNNs with BP algorithm for first level bids. The inputs are modified feature vectors . . . . .	98
4.4	Classification performance of FFNNs with BP algorithm for second level bids The inputs are modified feature vectors. . . . .	99
4.5	Classification performance of FFNNs with BP algorithm for third level bids. The inputs are modified feature vectors . . . . .	99
5.1	Classification performance of FFNNs with the BP algorithm for first level bids. Both crisp and fuzzy objective functions are used . . . . .	113
5.2	Classification performance of FFNNs with the BP algorithm for second level bids. Both crisp and fuzzy objective functions are used . . . . .	113
5.3	Classification performance of FFNNs with the BP algorithm for third level bids. Both crisp and fuzzy objective functions are used . . . . .	115
5.4	Classification performance of FFNNs with the BP algorithm for first level bids. Classification results for both configured and nonconfigured FFNNs are shown . . . . .	128

5.5	Classification performance of FFNNs with the BP algorithm for second level bids. Classification results for both configured and nonconfigured FFNNs are shown . . . . .	128
5.6	Classification performance of FFNNs with the BP algorithm for third level bids. Classification results for both configured and nonconfigured FFNNs are shown . . . . .	128
6.1	Relationship between the parameters used in fuzzy-rough neural networks and input space . . . . .	152
6.2	Comparative classification performance of FFNNs and FRNNs for first level bids . . . . .	154
6.3	Comparative classification performance of FFNNs and FRNNs for second level bids . . . . .	154
6.4	Comparative classification performance of FFNNs and FRNNs for third level bids . . . . .	154
7.1	Final classification results for opening bid problem . . . . .	171

# NOTATIONS

## English Symbols

$C_c$	Number of classes
$D$	cth output class
$d$	Distance between two clusters
$\varepsilon$	Euclidean distance between two vectors
$e$	Mean square error
$F$	Average dispersion
$\mathcal{F}$	Fuzzy clusters
$f$	Fuzzy integral
	Output function for hidden and output nodes of a feedforward neural network
	Fitness of a member of a population
	Fuzzy measure
	Number of equivalence classes or number of clusters or number of hidden nodes
$\mathcal{H}$	: Entropy
$h$	· Output of the modules in a modular network
$K$	Number of clusters
$M_{hc}$	Hard classification
$M_{fc}$	Fuzzy classification
$M_{pc}$	Possibilistic classification
$m$	Mean
$N$	: Dimension of input pattern vectors
$\mathcal{N}$	: Normal random number
$n$	: Number of patterns
$net$	· Activation value
$o_p$	Output of a node for pth input
$P$	Probability
$P$	Probability density
$q_k$	Number of clusters in the kth class
	Index of fuzziness
	Set of real numbers
	Radius of a cluster
	Number of modules
$S_k$	Number of patterns from kth class
$T$	Temperature
$t$	Target or desired output of a network
$U$	Fuzzy partition of X
$\mathcal{U}$	Uniform random number
$V$	Total fuzzy hypervolume
$\mathcal{W}$	OWA operator

## English Symbols (Continuation)

$\mathbf{w}$	: Weight vector
$X$	: Universal set
$\mathbf{x}$	: Input pattern
$\mathbf{y}$	: Modified feature vector

## Greek Symbols

$\Xi$	: Set of outputs generated by all modules
$\Sigma$	: Covariance matrix
$\Omega$	: Subset of $\Xi$
$\alpha, \beta$	: Constants lying in between 0 and 1
$\eta$	: Learning rate constant in feedforward neural networks
$\theta$	: Parameters used in parametric estimation
$\iota_A(\mathbf{x})$	: Rough-fuzzy membership function of $\mathbf{x}$ for the output class $A$
$\kappa$	: Bandwidth of a membership function
$\lambda$	: Constant used in Sugeno's measure
$\mu_A(\mathbf{x})$	: Fuzzy membership value of $\mathbf{x}$ in the set $A$
$\nu$	: Number of <b>offsprings</b>
$\xi$	: Output of a module
$\sigma$	: Standard deviation or width of a cluster
$\tau_A(\mathbf{x})$	: Fuzzy-rough membership function of $\mathbf{x}$ for the output class $A$
$\phi$	: Empty set
$\psi$	: Output vector from all classifier modules
$\omega$	: Weights given in OWA operators

## Miscellaneous Symbols

$ \mathbf{x} $	: Cardinality of $\mathbf{x}$
$\ \mathbf{x}\ $	: Distance between the origin and $\mathbf{x}$
$\overline{A}$	: Upper approximation of the set $A$
$\underline{A}$	: Lower approximation of the set $A$
$[\mathbf{x}]_R$	: Equivalence class in $X/R$ that contains $\mathbf{x}$
$\acute{\mathbf{x}}$	: Transpose of $\mathbf{x}$
$\hat{\mathbf{x}}$	: <b>Offspring</b> of $\mathbf{x}$

# ABBREVIATIONS

ANN	:	Artificial Neural Network
BP	:	Backpropagation
EC	:	Evolutionary Computation
EP	:	Evolutionary Programming
FFNN	:	Feedforward Neural Network
FKM	:	Fuzzy K-Means
FRNN	:	Fuzzy-Rough Neural Network
FKNN	:	Fuzzy K-Nearest Neighbours
FMLE	:	Fuzzy Modification of Maximum Likelihood Estimation
KNN	:	K-Nearest Neighbours
OWA	:	Ordered Weighted Average
PKNN	:	Possibilistic K-Nearest Neighbours



# Chapter 1

## INTRODUCTION

Human beings apply their classification ability to perceive patterns in natural scenes, stock market analysis, mental processes and in many other fields. It would be possible to build a new breed of intelligent machines if the human classification mechanism can be successfully emulated on machines. However, this goal **appears** to be difficult. One major reason behind it is the presence of uncertainties at different stages of the classification process. Presence of uncertainties may affect the classification process. The problem is made simpler by ignoring the uncertainties at every stage of the pattern classification process, but it results in an inevitable loss of information. The objective of this study is to demonstrate that incorporation of the knowledge in form of the uncertainties indeed enables us to design better pattern classifiers. In this thesis, the role fuzzy, rough and probabilistic uncertainties in a given classification task are discussed. This work illustrates the use of modular classification approach to handle the uncertainties efficiently in complex pattern classification tasks. The modular approach breaks the classification task into several subclassification tasks, solves each subclassification task, and eventually integrates the subclassification results to obtain the final classification result. In other words, partitioning of the classification task is carried out such that each subproblem can be solved in a module by exploiting the local uncertainties and the results of all the modules can be combined by exploiting the global uncertainties. The performance of the proposed approach is illustrated for the opening bid problem in Contract Bridge game.

### 1.1 Background: Problem Solving with Uncertainty

**Importance of pattern recognition:** Intelligence implies the ability to think, reason, learn and memorise. It is generally related to the human cognitive process. The fact that the human cognition process is marvellously efficient and effective poses a question to the scientists: Can some of the functions and attributes of the human reasoning be emulated on a machine? The reasoning can be for the tasks like classification, grouping,

and prediction. The issues involved in these reasoning tasks are discussed in the field of pattern recognition.

**Exploitation of uncertainties may improve the pattern recognition process:**

Many pattern recognition tasks in real life involve uncertainties at various stages. For instance, the input data to a pattern recognition system may have uncertainties due to randomness in the system generating the data or due to errors in the measurement of the data. Uncertainties may also arise in the selection and extraction of the features from the input data. The output of a recognition task may be vague too. Finally, the knowledge captured in the form of cause and effect relation is generally soft because the relationship between the input features and the output can be imprecise or only partially correct. However, human reasoning process is able to deal with these uncertainties effortlessly to obtain satisfactory solutions to many pattern recognition problems. Moreover, the decisions based on the soft relations or constraints seem to be robust against small variations in the parameters or features at every stage. In fact, these variations or uncertainties may be helping the human beings with updating the acquired knowledge, and thus, helping in the process of learning.

**Unsolved aspects and objective:** For solving pattern recognition problems on a machine, normally crisp quantities are derived from the uncertain data or information available at every stage. The problem itself is solved using an algorithm consisting of **unambiguous** sequence of processing steps. It is likely that the performance of a pattern recognition system may improve significantly in terms of accuracy, robustness and learning ability, if the system is designed to deal with ambiguities at different stages of processing the information. This requires identification of the sources of uncertainties and capturing the uncertainties in a suitable form for incorporating them for solving the pattern recognition problem on a machine. The objective of this study is to demonstrate the significance of incorporating the knowledge of the uncertainties for some real world pattern recognition problems.

## **1.2 Issues in Pattern Classification**

**Pattern classification is chosen for the study:** There are several pattern recognition tasks, which are relevant for this study, such as pattern classification, pattern storage, pattern clustering, associative memory recall and pattern mapping. This thesis considers pattern classification tasks for discussion throughout the study. The task of pattern clas-

sification is defined as a search for structures in a pattern set, and subsequent labelling of the structures into categories such that the degree of association is high among the structures of the same category and low between the structures of different categories [Bez81]. Pattern classification finds extensive applications in script recognition, face recognition, speech recognition, speaker recognition, ECG analysis, radar and sonar signal detection, weather forecasting, data mining, etc. [TG74] [Fuk89] [BP92].

Presence of uncertainty in pattern classification: In a pattern classification task, the input data is generated by a source, and the data is measured by a set of sensors. The sensed data is used to extract some relevant features, which in turn are used to associate class labels corresponding to the problem. To implement the pattern classification task on a machine, one needs to characterize the associated uncertainties at different stages. Some of the uncertainties may be identified as resolution, probabilistic, fuzzy and rough uncertainties. Resolution uncertainty is due to sensors, probabilistic uncertainty is due to the randomness in physical system generating the data. Fuzzy uncertainty [KY95] [PM86] is due to the vagueness in the human interpretation of the data at the feature level, class labels or may be at some intermediate levels. Rough uncertainty [Paw91] is due to incomplete information or knowledge at various stages.

Example: Let us consider an example of a pattern classification task to illustrate these uncertainties. Suppose an artificial vision system analyzes a digital image of a dice. Based on the evidence gathered, the system might suggest that the top face of the dice is either a 5 or 6, but cannot be more specific. This kind of uncertainty, known as resolution uncertainty, arises from the limitations (for example, sensor resolution) of the evidence gathering system. Again, randomness is involved when the outcome of a dice is predicted before the dice is cast. This uncertainty, which arises due to the chance or randomness, is called probabilistic uncertainty. On the other hand, if one is asked to interpret the top face of the dice as, say high, the uncertainty appears due to vagueness. This is called fuzzy uncertainty. In this example, rough uncertainty is absent.

Issues: The issues involved in modeling human classification mechanism on a machine are

- (a) Identification of the uncertainties involved in the chosen classification task.
- (b) Representation of the problem with uncertainties on a machine. It involves representation of the input, output and the knowledge of the problem at various stages.
- (c) Development of a methodology to exploit the uncertainties for solving the classifi-

cation problem.

### 1.3 Scope: Study of Uncertainties in Opening Bid Problem in Bridge Game

**Contract Bridge Problem is chosen to illustrate the efficacy of the proposed approach:** In order to demonstrate the significance of the uncertainties for solving a pattern classification problem on a machine, the "opening bid" problem in the game of Contract Bridge has been chosen. Contract Bridge [Khe88] is a card game played in two stages (for more details see the Appendix A). The aim is to maximize the points gained, which depend directly upon the number of tricks a side can win. In the first stage, both the players of each side make bids. Finally, through a bid the player stakes a claim for the denoted number of tricks. In practice the first few bids are used by the players to convey information about their hands. In the second stage, the cards are played out to see if the highest bidder can fulfill the contract.

In Contract Bridge a player makes a bid to convey information about the thirteen cards in his hand. The bid made by the first player in the game is called the 'opening bid'. He makes one of the permitted bids based only on the patterns of the cards in his hand, as he has no *a priori* knowledge of the cards in the hands of the other three players. In the opening bid problem, the input is the distribution of the thirteen cards in the player's hand, and the output is the legal bid the player makes. It is assumed that the bid is to be made according to standard conventions, so that no artificial conventions are involved. The aim is to capture the human reasoning process in the opening bid problem based on the real input-output pairs of the data collected from players of the Bridge game.

**Reason for choosing Contract Bridge Game:** One reason for choosing this particular problem for illustration is that it is easy to collect the data. Moreover, the input is the crisp data of the thirteen cards pattern, and hence, there is no resolution uncertainty in the problem. In addition, there is no noise in the input representation. In many pattern classification problems, preliminary processing of the data (e.g., speech signal) is essential to extract parameters or features. This in turn may result in loss of information at the input stage itself. Expert behavior in games, on the other hand, does not depend on any of these sensory interactions. This is particularly important if one is to generate faithful reproductions of human cognition. This is relevant if the objective is not only to attempt a task typical of humans, but to also try and mimic the human way of doing it. When the goal is to emulate human expertise, one has to be careful in selecting those areas that

can best be modeled without too many simplifying assumptions [KR89].

In spite of the simplicity in the representation, the opening bid problem is still very complex. For example, all the hands are not equally likely, and hence, learning all the hands equally well is not possible. This is true especially, since the hands corresponding to the higher level bids are very rare. Therefore, one problem is how to learn the rare hands along with the frequently available hands. In addition, for a given hand, the same player may make a different bid at a different time, which illustrates the variability in his reasoning process. This variability is present because the player changes his strategy based on his experience, vulnerability, etc. Since it is impossible to quantify the influence of these subjective qualities, two hands may appear same or similar, although they are not if the unaccounted features are also considered. Two hands with same or similar patterns may be classified to two different classes. This implies the presence of rough uncertainty in the bidding process. The output bid can be fuzzy. For example, the bid can be neither completely one “**Diamond**” nor one “**Spade**”. In this thesis the following issues are addressed:

1. How to represent the input hand pattern on a machine and how to interpret the output bid.
2. How to develop methods to effectively model and classify the input hand pattern to an output bid. In particular, how to take care of the rough, fuzzy and probabilistic uncertainty while modeling the bidding system.

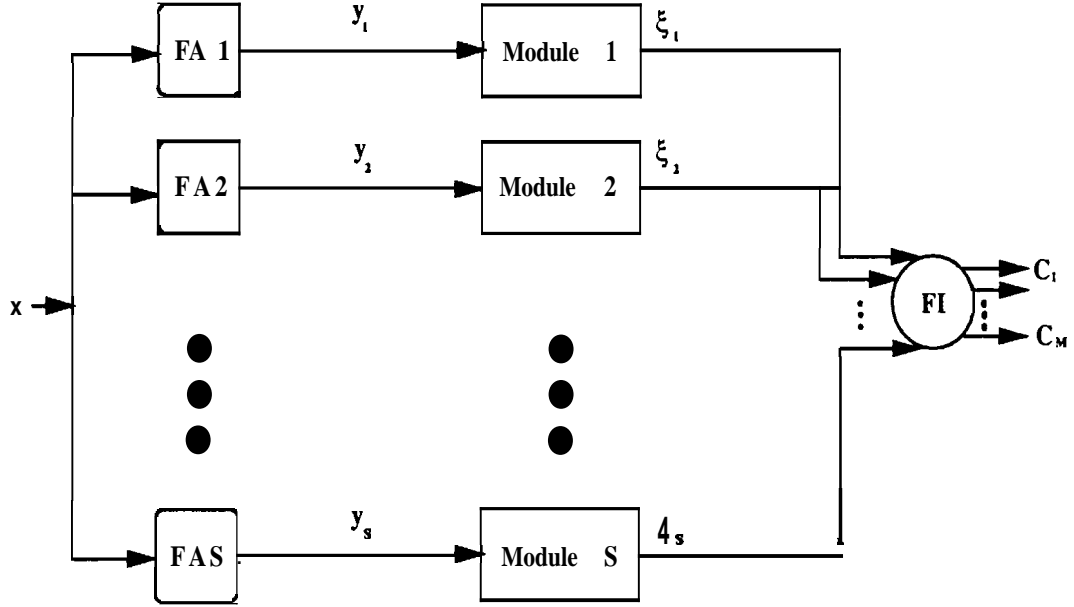
In Bridge game, hand patterns containing seven cards or longer suit constitute less than **5%** of the total possible number of hands. Therefore, to make the problem simpler, the study is kept limited only up to third level bids. However, it must be noted that this work is not intended to solve the bidding problem. Rather it illustrates the development of a pattern classification methodology based on the uncertainties associated with the given classification process.

Portability to other problems: The uncertainty-based pattern classification methodology may also be relevant for problems in vision, speech and other decision making fields, where a large part of the information is lost in representing the problem on a machine.

## 1.4 Proposed Approach for Capturing the Reasoning Process in Opening Bid Problem

The goal of this study is to develop a pattern classification technique based on uncertainties at different stages to capture the human reasoning process. Initially, we survey various existing techniques for pattern classification. We focus on the role of uncertainties in these classification techniques. Then an attempt is made to build a feedforward neural network [Yeg98] for the opening bid problem. Artificial neural network is chosen as a tool because it offers various advantages like incremental learning, robustness, universal approximation capability, etc. However, experimentally it has been found that a single monolithic neural network model may not be suitable for the complete classification task. Therefore, based on domain specific knowledge, the monolithic classifier is broken into several modules such that equiprobable classes and overlapping classes are kept in the same module. It aids, 1) to learn hands with less frequent patterns and highly frequent patterns equally well, 2) to deal with fuzziness among the close classes locally, and 3) to deal with roughness within each module locally. A post-processor treats the fuzzy and rough uncertainties globally, and it combines the results from all the modules to yield the final classification result.

Following the above track, the input representation has been fine tuned separately for each module using the concept of rough-fuzzy sets [DP92]. Each module can be a classifier that relies on the principle of direct classification or classification through clustering [Bez81]. In the direct classification approach, the whole feature space is directly analysed to delineate the output classes. Classification through clustering approach involves initial clustering of a subset of patterns from the original feature space, and subsequent partitioning of the whole feature space based on the clusters obtained. Following the first approach, a feedforward neural network is used for each module. These networks are trained by backpropagation algorithm with fuzzy objective functions. Thereafter, each such network is configured using evolutionary programming [Fog95] technique. Following the second approach, i.e., classification through clustering, the input data set is optimally classified using evolutionary programming-based fuzzy clustering technique. Next, using these clusters a fuzzy-rough neural network is evolved to establish the input-output relationship. Thus, several modules are constructed either by the direct classification method or by the clustering method. The evidence supplied by these modules are aggregated by a post-processor which is based on fuzzy integral. Finally, a modular network consisting of feature analysers, subclassifier modules and a post-processor (Fig. 1.1) is obtained. The



**Fig. 1.1:** A modular network with  $S$  different modules for the opening bid problem. Initially, the input feature vector  $(\mathbf{x})$  is modified separately for each module through a feature analyser (FA). The modified feature vector  $\mathbf{y}_i$ ,  $i = \{1, 2, \dots, S\}$  is fed to the module connected to the  $i$ th feature analyser. Each module can be a feedforward neural network or a fuzzy-rough neural network. The outputs of all the modules, i.e.,  $\{\xi_1, \xi_2, \dots, \xi_s\}$ , are fused by a fuzzy integrator (FI) to obtain the final classification result.

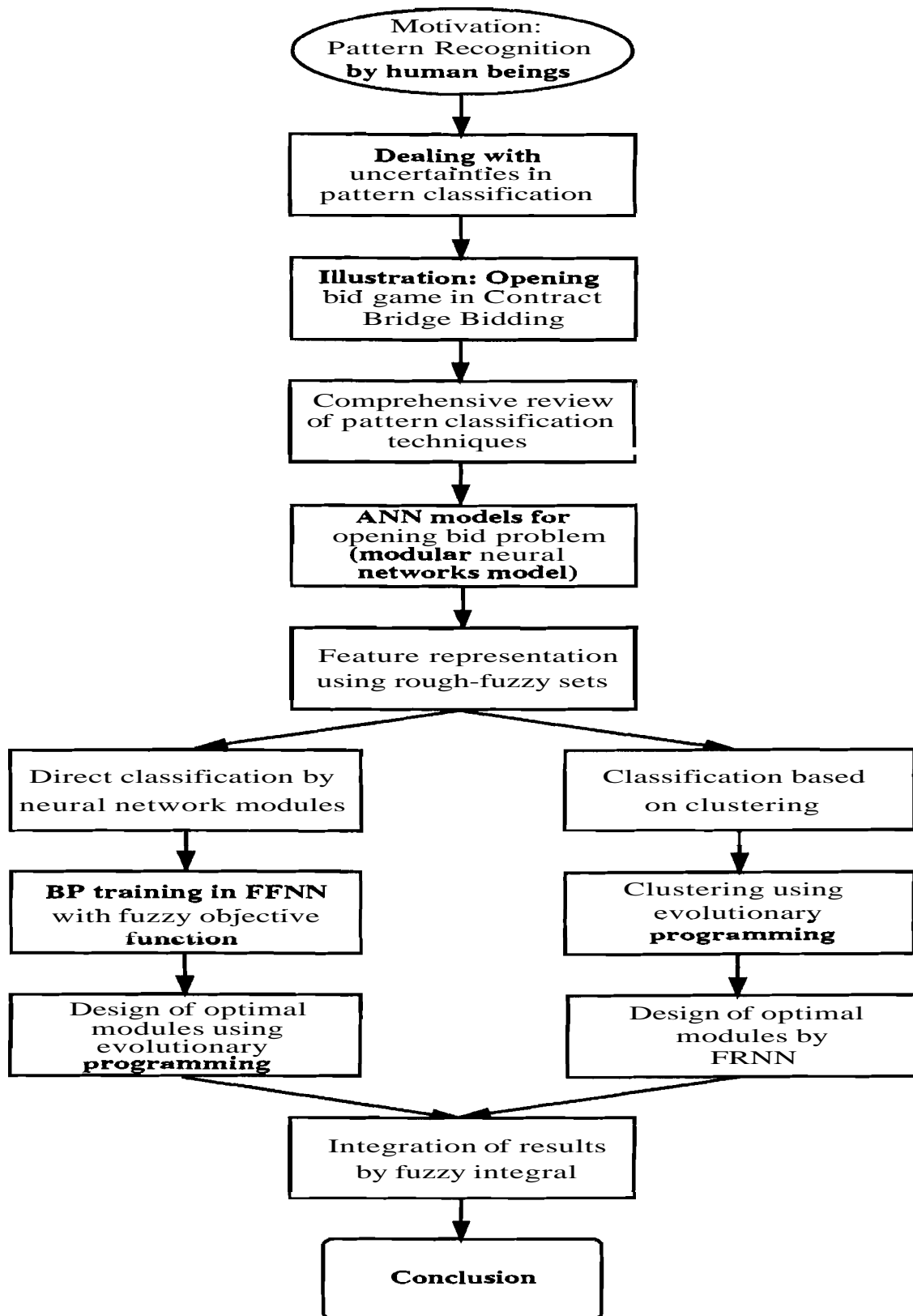
flow of the ideas described in this thesis is depicted in Fig. 1.2.

## 1.5 Organization of the Thesis

The organization of the rest of the thesis is as follows:

**Chapter 2** is devoted to review different classification stages involved in pattern classification. It also delineates different paradigms used for the classification stages. Specifically, attention has been paid on the manipulation of the various uncertainties present in the classification process. In a complex pattern classification task, modular approach is an attractive approach to handle the uncertainties efficiently. Later part of this chapter reviews several varieties of modular classifiers.

**Chapter 3** explores the possibility of capturing the implicit relationship in bidding a Bridge hand using an artificial neural network. Issues like the role of uncertaini-



**Fig. 1.2:** Flow of ideas across the thesis. The terms ANN, BP, FFNN and FRNN mean artificial neural networks, backpropagation, feedforward neural networks and fuzzy-rough neural networks, respectively.



ties, input representation, possible architectures for the network are studied. It was found experimentally that it is difficult to train a monolithic neural network for the opening bid problem. This chapter suggests the use of a modular neural network for attacking the bidding problem. The opening bid classification problem is partitioned into three subclassification tasks, and one module is assigned for each subclassification task.

**Chapter 4** focuses on fine tuning the input representation for a module based on the class discriminatory capability of the features for the output classes present in the module. Since both roughness and fuzziness are present in the opening bid problem, a rough-fuzzy *entropic* measure is proposed to quantify the rough-fuzzy uncertainty associated with each feature. The rough-fuzzy entropy corresponding to each feature is iteratively minimized to quantify the class discriminatory capability of the feature. These quantified values are used to derive the modified feature vectors for each module.

**Chapter 5** describes one approach of capturing the relationship between the modified feature vector and the output classes of a module through direct classification. It involves partitioning the modified feature space of a module into several decision regions or output classes. The boundary between any such two regions is fuzzy. This chapter employs feedforward neural networks to capture the relationship between the modified feature vector (crisp) and the output classes (fuzzy) present in each module. Backpropagation learning algorithm with fuzzy objective functions are used to train the networks. In addition, evolutionary programming-based technique is applied to configure each network optimally.

**Chapter 6** examines clustering-based approach to capture the input-output relation in each module. In this approach clustering of the modified feature vectors is followed by labelling of each cluster with a class label. This chapter proposes a technique to construct a classifier module in presence of fuzzy and rough uncertainties. The modified feature vectors are clustered using an evolutionary *programming-based* fuzzy clustering algorithm. The relationship between a cluster and the output class labels are estimated through the fuzzy-rough membership functions associated with each input pattern. Using the fuzzy-rough membership functions, a *fuzzy-rough* neural network is constructed to relate the input and output.

**Chapter 7** combines the information supplied by all the modules using a fusion technique based on Sugeno's fuzzy-integral. In the earlier chapters, the original classification

task is distributed among small modules, and the modules have been trained and configured to deal with the uncertainties locally. Here, the outputs of all the modules are treated as evidence, and they are fused in a non-linear fashion based on their importance. The importance of each evidence is determined using the *fuzzy-roughness* associated with the evidence. The final class label of an input is the output class corresponding to the maximum fuzzy integral value.

**Chapter 8** concludes the thesis, by summarising the work and indicating the future directions of using uncertainties in modular neural network classifiers.

# Chapter 2

## MODELING PATTERN CLASSIFICATION PROBLEMS

### 2.1 Introduction

In the last few years, there has been a large upswing in research activities in the problems of pattern classification [DH79] [Fuk89] [Bow84]. Although far away from human classification ability, machine classification techniques attempt to mimic the human classification mechanism in several stages. In this chapter we discuss the functions performed by these stages, the uncertainties pertaining to them, and the working principle of the various techniques used for these stages. This discussion includes some uncertainty driven techniques like statistical, fuzzy and rough approaches. When the classification problem is complex, one greedy approach to solve the problem is modular classification approach. Following the principle of 'divide and conquer, modular approach breaks the classification task into several subclassification tasks, solves each subclassification task, and eventually integrates the subclassification results to obtain the final classification result. In other words, partitioning of the classification task is carried out such that each subproblem can be solved in a module by exploiting the local uncertainties, and the results of all the modules can be combined by exploiting the global uncertainties. Later part of this chapter summarises the architecture and working principle of some of the existing modular classifiers.

The organization of the chapter is as follows: In section 2.2 we describe how machines are used to mimic human classification mechanism. It reviews several methods to perform classification process on a machine. Section 2.3 analyses issues and architectures of modular classifiers. Section 2.4 attempts to frame the Contract Bridge opening bid problem as a pattern classification problem.

## 2.2 A Review on Pattern Classification

A pattern is a description of an object [TG74]. A pattern can be a *concrete* item which can be recognized by human sensory organs, like eyes and ears [TG74]. Image pattern, speech pattern, hand pattern of a Bridge player, etc., are the examples of concrete items. On the other hand, a pattern may be an *abstract* item, like a pattern of thought process, which we can recognize with our sensors like eyes and ears closed [TG74]. The task of pattern classification is defined as a search for the structures in a pattern set, and subsequent labelling of the structures into categories such that the degree of association is high among the structures of the same category and low between the structures of different categories [Bez81] [KY95]. In this chapter we address the pattern classification problem based on the concrete patterns only.

In section 2.2.1 we describe how machines are used to mimic human classification mechanism. Section 2.2.2 discusses several methods to represent the classification process on a machine. It also describes the basic five stages involved in a machine-based pattern classification technique. Section 2.2.3 analyses several aspects of the first stage, i.e., feature extraction stage. Section 2.2.4 reviews a few methods used for the second stage, i.e., how to interpret the structures present in the input data set. Section 2.2.5 discusses various current methodologies to discover the structure present inside the data. Issues involved in the last stage, i.e., generalization, are discussed in section 2.2.6. The relationships among the topics discussed in this section are illustrated in Fig. 2.1.

### 2.2.1 How Human Classification Mechanism is Mimicked on Machines

Let us take a real life pattern classification example. Suppose, one is asked to determine whether a particular person is a European or an Asian. He cannot do it unless he has already seen a set of European and Asian people, or someone has told him about the difference explicitly. While observing a set of European or Asian people, he gathers some experience about them. In other words, gathering experience means, based on certain characteristics of these people, he extracts some common property from them. For instance, he watches their height, eye color, etc., and based on that he realizes most of the Europeans are tall and their eye colours are not black. The opposite is true for the Asians. Now, he tries to find whether a new person is tall or his eye color is black, and based on that he can determine that person's identity (assuming that the person can come only from any one of those two classes). If he can decide the identity of most of

# Pattern Classification on Machines

---

- **Mimicking Human Classification on Machines**

- **Process Description**

- Symbolic Process Description
- Numeric Process Description
  - \* Relational data
  - \* Object data

- **Feature Analysis**

- Preprocessing
- Feature Selection
- Feature Extraction

- **Structure Analysis**

- Direct Classification
- Classification Through Clustering

- **Abstraction: Search for the Structure**

- Search During Training
  - \* Supervised
  - \* Reinforcement
  - \* Unsupervised
- Search During Testing

- **Classifier Types**

- Deterministic Classifiers: Crisp Rule Base System
- Statistical Classifiers
  - \* Parametric
  - \* Nonparametric
  - \* Semiparametric
- Fuzzy Classifiers
  - \* Fuzzy relation
  - \* Fuzzy pattern matching
  - \* Fuzzy clustering
  - \* Fuzzy K-nearest neighbors
- Rough Classifiers
- Hybrid Classifiers
  - Neuro-Fuzzy
    - \* Neuro-Rough
    - \* Neuro-Evolutionary
  - Fuzzy-Rough
    - \* Fuzzy-Evolutionary
    - \* Rough-Evolutionary

- **Generalization**

Fig. 2.1: Relationships among different topics discussed in the context of pattern classification (section 2.2).

persons he encounters, then he is called *intelligent*. It is because his ability to extract the common property out of these two classes is good.

In order to mimic the above mentioned human classification mechanism on a machine, several instances (e.g., a set of Europeans and Asians) of the problem are collected. Care should be taken to collect the sample data randomly from the whole population. To represent these instances on a machine, typical properties, termed as *features* (e.g., height and eye color), are extracted from each instance. These features represent the given problem in a higher dimensional feature space. Feature extraction is a **difficult** task as less number of features may not be sufficient to represent the problem, whereas too many features can affect the classifier system (this phenomenon is known as *curse of dimensionality* [Bis95]). In addition, we do not know how many features are sufficient, or which feature is necessary (e.g., in the above example color of each person's cloth need not to be noted). The features form some structures in the feature space. The interpretation of the structures depends on the pattern recognition task and situation. For instance, in recognizing English characters, twenty six different class structures are present. On the other hand, in distinguishing English characters from Arabic characters, only two structures are interpreted [Fu68]. Now, the task of pattern classifier is to search the structure. This search becomes complicated because of the presence of uncertainties associated with the structure. Thus, the whole pattern classification process involves manipulation of the information supplied by the instances. The instances contain the information about the process generating them, and the extracted features reflect this information. The structures present inside the features represent the information in an organized manner so that the relationship among the variables in the classification process can be identified. Finally, in the last step, a search process recognizes the information from the structure. Now, if a new pattern is encountered, the machine detects the structure in which the input pattern belongs, and based on the structure the pattern is classified. Therefore, once the structure is found, the machine is capable of dealing with new situations to some extent. The ability to deal with new situations can be evaluated by testing the classifier with several new examples, for which we know the answers for comparison. If the performance of the pattern classifier with this so called test data is good, then we say that the machine has *generalized* well.

An important assumption in the pattern classification task (for humans as well as machines) is that nature is by and large stable—what is known yesterday is true for today and tomorrow. In other words, it means that to some extent today's experience is valid

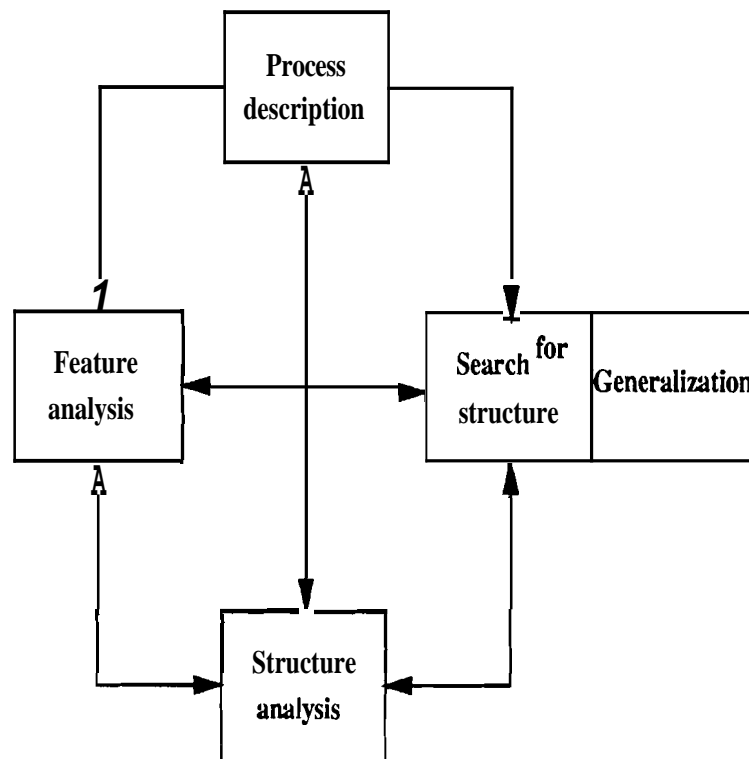
tomorrow. This assumption is essential; otherwise, there would have been no meaning of gathering experience through learning and using this experience further for generalization.

Realizing the pattern classification task by a machine becomes complicated due to various uncertainties. A few of them are known, among them fewer we can model. Some of them are

- **Resolution uncertainty:** Caused by inaccurate measurement in measuring instruments.
- **Probabilistic uncertainty:** Caused by randomness in physical systems.
- **Fuzzy uncertainty:** Caused by vagueness in human reasoning.
- **Rough uncertainty:** Caused by our incomplete knowledge about the classification system we are trying to model.

Although all the above four are uncertainties, they are basically different from each other. The fuzzy uncertainty differs from the probabilistic uncertainty and resolution uncertainty, because it deals with situations where set-boundaries are not sharply defined. The probabilistic uncertainty and resolution uncertainty are not due to the ambiguity about the set-boundaries; rather about the **belongingness** of elements or events to crisp sets [PB94]. Specifically, fuzziness deals with vagueness between the **overlapping** sets [Bez94] [KY95], while probability concerns the likelihood of randomness of the phenomenon. On the other hand, rough sets deal with coarse nonoverlapping concepts [DP90] [DP92]. Both roughness and fuzziness do not depend on the occurrence of the event; whereas probability does. Fuzziness lies in the subsets defined by the linguistic values, like tall, big, whereas indiscernibility is a property of the referential itself, as perceived by some observers, not of its subsets [DP92]. In fuzzy sets each granule of knowledge can have only one membership value to a particular class. However, rough sets assert that each granule may have different membership values to the same class. Fuzzy sets deal with overlapping classes and fine concepts; whereas rough sets deal with nonoverlapping classes and coarse concepts. In a pattern classification problem, all or some of the above uncertainties may be present.

For simplicity, most of the pattern classification problems can be decomposed into five different stages. **From** an abstract point of view, the division of the classification problem into five different stages may seem to be quite arbitrary. The entire process can be viewed as a single mapping from the object space to the decision space. Optimum mapping is the one for which the probability of error is minimum. In practice, this leads to a very



**Fig. 2.2:** Relationships among different steps involved in pattern classification. In many cases certain stages can be skipped. For instance, from process description we can go to generalization directly without going through feature analysis and structure analysis.

complicated calculation, which is in fact currently impossible to solve [Dud70]. In many cases depending upon the given problem certain stages can be skipped (Fig. 2.2). For instance, in a few applications it is possible to consider the raw input data as features, and hence, there is no need of any separate feature analysis stage. The five stages are discussed in the next five subsections.

### 2.2.2 Process Description


The first stage concerns the way the classification process will be represented. The following are some of the methods for this stage:



### 2.2.2.1 Symbolic process description

The classification process can be represented in terms of different symbols. In the decade of the 1980s symbolic approach became a dominant theory to explain how intelligence is produced and how it can do certain useful tasks. Using this method it is possible to write programs that work with symbols rather than numbers. Symbols can be equal or not equal, and that is the only relationship defined between the symbols. Hence, it is not possible to know if one symbol is less than another symbol. Of course, in symbol processing programs, the symbols do get represented by numbers. Besides the use of symbols, the symbol processing programs consists of a large number of rules. The most significant outcome of the symbolic approach is the development of the *knowledge-based expert systems* [LS95]. It tends to capture the higher level human reasoning functions in the form of a set of *if-then rules* or *knowledge*. A typical set of two symbolic rules in chromosome identification can be

If the input is  then the output class is  $C_1$

If the input is  then the output class is  $C_2$

If a new chromosome is encountered, then the structure of the chromosome is matched with the *if* part of each rule. The class label of the new chromosome is the class label corresponding to the *then* part of the matched rule.

Eventhough in symbolic approach symbols can only be equal or not equal, and there are no other relations defined for them, quite often “symbolic” programs end up using integers or **reals** as part of the program, and it is called symbolic anyway eventhough by the strictest standard doing so no longer makes the program completely symbolic, only partly symbolic. Drawbacks of the symbolic approach are the following: (a) It does **not** take care of pattern variability, (b) it needs large number of rules when the inputs are continuous, and (c) it does not employ efficient learning mechanisms to acquire the rules.

Symbolic approach is useful in *syntactic pattern recognition* technique [Fu82]. This a p

**proach** deals with the patterns which are rich in structural information, i.e., the patterns that contain most of their information in their structures, rather than in numeric values. In this approach the input patterns are divided into several parts, and one symbol, called primitive, is assigned to each part. Each primitive has no direct relationship to the structure of the pattern. A pattern is represented by the knowledge about how primitives must be combined to make up the entire pattern, and how primitives are related to each other. In the syntactic methods, building classifiers consist of constructing rules for combining primitives to obtain the structure of a given object. The methods are formulated around the concept of formal languages with each primitive represented as a terminal symbol and a grammar inferred for each pattern class. When a new pattern comes, it is represented as a set of primitives, and the primitives are matched against the antecedent parts of the rules to determine the output class. The pattern grammar used for these rules can be made more effective by using stochastic grammar (in presence of randomness) or fuzzy grammar (in presence of vagueness). It should be noted that existence of a recognizable physical structure is essential for the success of the syntactic approach. Syntactic pattern recognition research has been largely confined to pictorial patterns, which are characterized by recognizable shapes, such as characters, chromosomes, finger prints, etc. Many of the major problems associated with the design of a syntactic pattern recognition system have been only partially developed. For instance, not much progress has been achieved in deriving general training algorithms for syntactic systems [TG74] [PM86] [Fu68].

#### 2.2.2.2 Numerical process description

The most familiar choice of representing objects or patterns are by numbers. Unlike symbolic approach, in numerical approach, the information about the classification process is extracted from the following types of numerical data:

- **Object data:** Object data can be the numerical representation of some physical entities, like images and hand patterns of a card game.
- **Relational data:** It may happen that, instead of an object data set, we have access to a set of numerical relationships between pairs of objects; that is, the relationship represents the extent to which the objects are related. For example, in numerical taxonomy, the relationship between species families can be assigned by human experts. Here, we do not have any access to the object or species, rather the relationship among them. Relational data are found in diverse application fields,

e.g., cognitive maps, influence diagrams, etc. [Bez96].

In this review, we shall discuss the other stages of pattern classification process assuming the representation is numeric. In addition, the word "data" will always imply the object data.

### 2.2.3 Feature Analysis

Feature analysis can be defined as a method that is used to explain and improve the data collected during the process description. It consists of the following three steps:

**Preprocessing:** A preprocessor is used in this step to perform some or all of the following functions: (a) Strengthening features, i.e., edges, specific frequencies, etc., (b) provide invariance to translations, rotations, and scale changes of the input data, (c) noise suppression, and (d) formatting the processed data for acceptance by the recognition device [Vig70].

**Feature selection:** Feature selection seeks a small number of features by obtaining a subset from the original set, either by discarding poor features or selecting good ones. Feature selection can take place by minimizing some objective function. The choice of the objective function may be classifier independent, or it may be based on particular classifier's accuracy to judge whether a feature subset is superior to another subset. The former approach is known as *absolute feature selection* approach or *filter* approach and the later approach is known as *performance-dependent feature selection* approach or *map-per* approach [BL98] [SB97] [WAM97]. The objective functions are carefully designed so that interclass distance of the input data set decreases, but the intraclass distance increases [DH79]. The distances may be Euclidean, Mahalanobis or some other standard distance measure, or it may be the distance between two probability distributions. The objective function like mean square error is based on Euclidean distance, and the objective functions like entropy and divergence are based on the distance between two probability distributions [Fuk89]. It is also possible to transform or rotate the axes of the input data set so that the interclass distance decreases and the intraclass distance decreases. Principal component analysis and Karhunen Loeve transform are based on this idea [DH79]. Note that in these cases the transformation of axes is actually a linear mapping. A natural extension of this scheme is nonlinear mapping [Fuk89]. Till now, nonlinear mapping has not become popular as it is very difficult to handle.

**Feature extraction:** It deals with developing some new features based on the already

selected features. From preprocessing and feature selection steps, the designer of classifier obtains the features that he knows or suspects are important. These may prove to be inadequate, or may provide a format not suitable for the decision mechanism. For example, in statistical feature extraction, a sample set of preclassified patterns is analyzed, and the statistical information collected from this sample set is used to augment the known feature list and to reformat the feature profile [Vig70].

The presence of noise, missing attributes, missing attribute values, etc., can make the feature analysis difficult [FSW97]. In addition, the presence of uncertainties can make this stage more complex. For instance, the input features may be vague in terms like tall, almost 5, etc. The features that are present may be insufficient for a particular class. This creates rough uncertainty in the classification task.

Practically, the methods by which initial features are obtained are often intuitive and **empirical**, drawing upon the designer's knowledge about and experience with the problem. The main guideline here is that the features should be invariant to (or, at least insensitive to) irrelevant variations, such as limited amount of translation, rotation, scale change, etc., while emphasizing differences that are important for distinguishing between patterns of different types. These features are ranked to select only the most important features (feature selection), and then some new features may be augmented with the extracted features (feature extraction).

Feature analysis serves several functions. Firstly, by reducing the input patterns to its essential features, the memory requirement to store the input patterns can be reduced. Secondly, by reducing the input data to more independent features, a considerable amount of invariance to exact form is obtained. Finally, by extracting more than the minimum number of features required, a degree of invariance to noise and background may be achieved [DL97].

#### **2.2.4 Structure Analysis**

The structure (spatial) present in the feature space represents certain common properties of the feature. Building classifiers may be impossible if such common property is not present in the data; then even a look-up table scheme would be sufficient. For instance, it is impossible to capture any common property from a set of names while classifying them into two classes, below 50 years and above 50 years. Moreover, the feature selection may not be proper when the feature is deep hidden due to many surface features. In this case

there may not exist any structure at all. For example, if we represent a large dimensional parity problem by a string of 0 and 1, then each character of the string does not carry any common property, which can be utilized to classify the input strings to even or odd parity. On the other hand, if we extract a feature that represents the sum total of all 0's and 1's present in the string, then there is certainly a structure. It is because, when this sum total is odd, it represents an odd parity, and when it is even it represents an even parity.

Interpretation of the structure in the feature space depends on the method followed in classification. Let  $X$  denote the original population of the data from which the example data set  $\mathcal{X}$  has been drawn, i.e.,  $\mathcal{X} \subset X$ . A  $C$ -class classification can be carried out in the following two ways:

1. **Direct classification:**  $X$  is used to train the classifier, i.e., to delineate the output classes in  $X$  into different decision regions [Bez81]. Therefore, the structure in the space  $X$  is directly analysed.
2. **Classification through clustering:** This technique involves initial grouping or clustering of  $X$ , and subsequent partitioning of the set  $X$  based on the obtained structure. Therefore, the structure in the training data space  $X$  is analysed. Later, based on this structure, the structure of the space  $X$  is analysed.

In the classification through clustering, the structure is analysed in the space formed by  $X$ . Since this step needs clustering as a prerequisite, we will discuss the basic concepts of clustering in brief. Clustering can be defined as follows [Bez81] [DJ87] [Har75]: Given a set  $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  of feature vectors, find an integer  $K$  ( $2 \leq K < n$ ) and a  $K$  number of partitions of  $X$  which exhibit categorically homogeneous subsets. There exists many clustering algorithms. Among them, the simplest and popular one is  $K$ -means clustering algorithm [DH79]. It starts with  $K$  random initial cluster centers. The algorithm considers each input pattern sequentially, and assigns the input pattern to the nearest (in Euclidean distance sense) cluster center. After the assignment is over for all the input patterns, each cluster center is updated so that it becomes the mean of the patterns that are associated with it. Same procedure is repeated for several iterations until there is no appreciable change in the position of the cluster centers. After clustering, each pattern belongs to only one cluster, and a structure evolves in the training set  $X$ . Most of the clustering algorithms assume that  $K$  is known *a priori*. To find the approximate value of  $K$  for a given set of data, various methods based on cluster validity exist [DJ87].

Presence of uncertainties may make the boundaries of the classes or clusters overlapping. It may also happen that the same cluster represent patterns from more than one class. It is because the relation between the input structure and the output class labels is one-to-many. Thus, uncertainties make the structure analysis difficult, and these difficulties are manifested in the next stage.

### 2.2.5 Abstraction: Search for Structure

This stage involves exploring structures using all the available information so that the obtained structure can be used for classifying a new sample with unknown class. Mathematically, let  $X$  denote the feature space from which  $X$  has been drawn, i.e.,  $X \subset X$ . A classifier for  $X$  is a device or means whereby  $X$  itself is partitioned into  $C$  decision regions. Explicit representation of these regions depends on the nature of  $X$  (i.e., data), the way in which the regions are formed (i.e., structure), and the model we choose for searching [Bez81].  $X$  is often employed to "train" the classifier, that is, to delineate the decision regions in  $X$ . The resulting structure may enable the machine to classify subsequent observations rapidly and automatically. The training method adopted in this stage can take place mainly in the following two ways:

1. **Supervised:** In this process a known set of input-output pairs is used to teach the classifier system first how to classify, and then let the system go ahead freely classifying other new patterns. In this case we usually need some *a priori* information, i.e., a training set consisting of a set of input-output pairs, to form the basis of teaching.
2. **Reinforcement:** In many classification problems, it is not possible to obtain a known set of input-output pairs. The output may be known only partially. This partial information may state that the actual output is "too high" or "50% correct". Unlike the supervised learning, here the teacher signal only says how bad a particular output is, and provides no hints on what the right answer should be [LL96].

It is possible, but not necessary, to conduct the search by first clustering the patterns in  $X$ . The clustering can take place in supervised or unsupervised fashion. In the supervised clustering, all the training data from a particular class are collected, and the clustering is carried out on this set of data. In the unsupervised clustering, the clustering is done on the whole training set.

In some search operations, there is no training. The search operation is left for the testing phase. Hence, the testing becomes time consuming, and thus, it makes these algorithms unsuitable for online testing. Some of these algorithms are called *lazy algorithms* [WAM97].

When a new input comes, it is classified based on in which part of the structure it falls. However, the boundaries among the different parts of the structure may be ambiguous. Due to this uncertainty, classifiers can be of the following types:

#### 2.2.5.1 Deterministic classifiers: Crisp rule base

If the boundaries of the different parts of the structure are not ambiguous, then the test pattern can be classified without any uncertainty. It happens in *deterministic classifiers*. An example of deterministic classifiers is a *crisp rule base system*.

Instead of representing knowledge in a relatively declarative, static way (as a set of things that are true), crisp rule base systems represent knowledge in terms of a set of *if-then rules*, a set of facts, and some interpreter controlling the application of the rules when the facts are given. A typical rule in the rule base is

If the input is 23, then the output class is  $C_2$

Note that this rule is different from the symbolic rule describe in section 2.2.2.1. Here the input is a number, but in a symbolic rule the input must be a symbol. There are numerous techniques to construct a rule base from a set of data. Among them one important approach is *evolutionary computation* (EC) theoretic approach [Fog94b] [BHS97] [Fog98]. EC is a technique to encompass a variety of population-based problem solving techniques that mimic the natural process of Darwinian evolution. Current research in the evolutionary computation has resulted in powerful and versatile problem solving mechanisms for global searching, adaptation, learning and optimization in a variety of pattern recognition domains. The main avenues for research in evolutionary computation are *genetic algorithms* [Hol75] [Gol89] [Dav91], *genetic programming* [Koz92], *evolutionary strategies* [Sch81] and *evolutionary programming* [FOW66] [Fog91b] [Fog95]. Genetic algorithms deal with chromosomal operators, while genetic programming stresses on operators of more general hierarchical structures. Evolutionary strategies emphasize behavioral changes at the level of the individuals, whereas evolutionary programming focuses on behavioral changes at the level of the species. The common factor underlying **all** these approaches is the emphasis on an ensemble of solution structures, and the evaluation

and evolution of these structures through specialized operators that mimic their biological counterparts, in response to an ever changing environment. Specifically, all of them maintain a population of trial solutions, impose random changes to those solutions, and incorporate the use of selection to determine which solutions are to be maintained into future generation and which are to be removed from the pool of trials.

From a mathematical point of view, all the EC techniques are *controlled, parallel, stochastic search and optimization* techniques. There are, two different training approaches for exploiting these optimization techniques to evolve the classification rules. In one method (also known as *Pitt's approach* in genetic algorithm community [Mic92]), each element of the population represents one complete classification rule set. Consequently, the complete population is an ensemble of many rule sets. In the process of evolution the rules compete among themselves, the weak individual dies, the strong survives and reproduces. In the other approach (also known as *Michigan approach* in genetic algorithm community [Mic92]), the whole population represents only one rule base, i.e., each member of the population represents a single rule. The second method is more time and space efficient. But, it needs, (a) delicate *credit assignments*, for which a heuristic method should distribute positive or negative credits among the members of the population, and (b) the members of the population, i.e., different parts of the network, to cooperate with each other so that they can build the complete rule base [Mic92] [WC96]. For both the training approaches, generally supervised or reinforcement learnings are used. The difficulties that most of the EC algorithms face are the optimal balance in exploration and exploitation, and premature convergence. EC keeps a balance between what already has worked best and exploring possibilities that might evolve into something even better [CHL96]. But, the balance is not optimal in practice. Moreover, in spite of the in-built *stochasticity*, EC algorithm may get stuck in local minima. This kind of premature convergence takes place mainly when all the new offsprings are similar to the existing offsprings (thus virtually stops the exploration of new space in the search domain). In order to reduce these problems, there are various strategies like modification of EC operators, increasing population diversity, etc.

Crisp rule base can be applied to a classification problem provided the input features are discrete. To accommodate the continuous features, one approach is to discretize the input feature with inevitable loss of information. When the input features are distorted due to noise and measurement errors, the variation in the input features increases rapidly. If discretization is carried out on this feature set, then the distortion may become very



high, and eventually the classification performance may decrease significantly. Note that EC does not use any uncertainty inherent to the problem; it introduces probabilistic uncertainty externally to make the search operation more efficient.

#### 2.2.5.2 Statistical classifiers

In many classification problems, which deal with measuring and interpreting physical events, statistical considerations become important in pattern recognition because of the randomness in the pattern generation process. Here the randomness comes from the physical process which generates it. For instance, in the Asian-European example, if we have certain statistical ideas about the occurrences of the persons in the two different classes, we can derive a classification technique which is optimal in the sense that, on the average, its use yields lowest probability of committing errors, provided the cost of misclassification is equal for all the classes and no cost is associated with correct classification. This statistically optimal classification technique is a generally accepted standard for classifiers where the outputs come from  $[0, 1]$  and sum of them is equal to 1. Henceforth, these kinds of classifiers will be called *probabilistic classifiers*. One such classification technique is *Bayes classification* technique. The assumption needed for using the Bayes classifier is that the feature vectors are random vectors whose conditional density function depends on its class. Let the class conditional probability density function,  $p(\mathbf{x}|i)$ , along with *a priori* probability ( $P_i$ ) of each class be known. The Bayes classification rule states that the input  $\mathbf{x}$  belongs to the class  $i$  with probability

$$p(i|\mathbf{x}) = \frac{P_i p(\mathbf{x}|i)}{\sum_{i=1}^C P_i p(\mathbf{x}|i)} \quad i = 1, 2, \dots, C \quad (2.1)$$

The Bayes classifier is defined as the classifier which computes  $p(i|\mathbf{x})$ , where  $i = 1, 2, \dots, C$ . The output class label is determined by maximum *a posteriori* probability. That is, the class label is  $c$ , if  $p(c|\mathbf{x}) = \max_i \{p(i|\mathbf{x})\}$ . While implementing Bayes classifiers, in many cases, we do not have any idea about the input distribution. There are the following three methods to estimate the class conditional distribution of the inputs.

**Parametric estimates:** In this approach a functional form  $p(\mathbf{x}|i, \theta)$  for the probability density  $p(\mathbf{x}|i)$  is assumed, where  $\theta$  is a parameter vector. The parameter vector is then optimized by fitting the model to the data set. It leads to the parametric estimation of the Bayes classifier. *Unsupervised maximum likelihood classifier* and *supervised*

maximum likelihood *classifier* are the two classifiers that are based on this idea [Bez96]. The drawback of this method is that the chosen form of parametric function may not be able to provide a good representation of the true probability density, and the number of parameters in the model grows with the size of the data set.

**Nonparametric estimates:** In many pattern classification problems the classification of an input pattern is based on the training data, where the respective sample size of each class is small, and possibly not representative of the actual probability distributions. In these situations, nonparametric models are attractive as they do not assume any particular form of the density function.

There are mainly two types of nonparametric classifiers. One type consists of procedures for estimating the density function  $p(\mathbf{x}|i)$  from the sample patterns. The approach based on Parzen Window falls in this category [DH79]. Another type consists of procedures for directly estimating the *a posteriori* probabilities  $p(i|\mathbf{x})$ . It is accomplished by collecting a set of correctly classified samples, and by classifying each new pattern using the evidence of the nearby sample observations. One such approach, popularly known as K-nearest neighbours (KNN) algorithm [DK82], is used as a simple nonparametric supervised method for the assignment of a class label to the input pattern based on the class labels represented by the K-closest (in some distance sense) neighbors of the input. It can be shown that the error rate of 1-NN (i.e.,  $K=1$ ) is bounded above by no more than twice the optimal Bayes error rate, and moreover, when  $K$  increases with infinite number of training data, the error rate approaches the Bayes optimal rate asymptotically [CH67]. This algorithm is also a typical example of lazy algorithm mentioned in section 2.2.5. The advantage of this algorithm is that it does not need any a priori knowledge about the structure present in the training data. Like other nonparametric techniques, in KNN also the number of samples is greater than the number that would be required if we know the form of the density. The demand for a large number of samples grows exponentially with the dimension of the feature vector. Consequently, when the number of test data is large or  $K$  is large, KNN takes large amount of time.

**Semiparametric estimates:** This approach is a compromise between parametric and non-parametric approaches, and it tries to enjoy the advantages of both parametric and non-parametric approaches [Bis95]. It allows a general class of functional forms in which

the number of adaptive parameters can be varied independently from the size of the data set. Artificial neural networks can be regarded as typical examples of this approach.

The Bayes classifier gives optimal classification performance for the probabilistic classifiers [Bez96], provided the parameters of the input distribution are estimated from the inputs collected over the whole input space. In practice, the parameters of input distribution are estimated based only on a finite number of training data. As a result, the performance of the resultant Bayes classifier is no longer optimal, but its performances approaches the optimal one as the number of input data is made very large (theoretically, it is infinity). Nevertheless, the Bayes classifier, based on a finite number of training samples, is used to compare the performance of the other probabilistic classifiers.

Artificial neural network (ANN) [Arb95], a semiparametric model, needs special attention, and in what follows we will describe it in detail. An ANN is an interconnected assembly of simple processing units or nodes, whose functionality is loosely based on the biological neuron. The processing ability of the network is stored in the inter-unit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns [BL96] [Rip96]. Some of the advantages of using ANNs are [RY95]

1. Any continuous input-output function can be captured by ANNs.
2. ANN models can learn the statistical distributions underlying the input patterns. Hence, ANN-based classifiers do not need to know the input probability distribution explicitly.
3. Certain ANN models can act as constraint satisfaction models. Such networks can be used to represent different domain-specific constraints [RY96], [RY97] [RPY97].
4. Information stored in an ANN is not represented locally, rather it is distributed over the entire network through synaptic weights. Hence, ANNs are fault tolerant in the sense that even if some connections are snapped or some of the processing elements are damaged, performance of the networks is not affected significantly.
5. ANNs take care of pattern variability. Moreover, ANNs do not need any input-output rule to be known.
6. ANNs can learn incrementally, and hence, they do not need a huge data storage.
7. Other advantages like parallel computation, robustness, etc., make ANNs attractive.

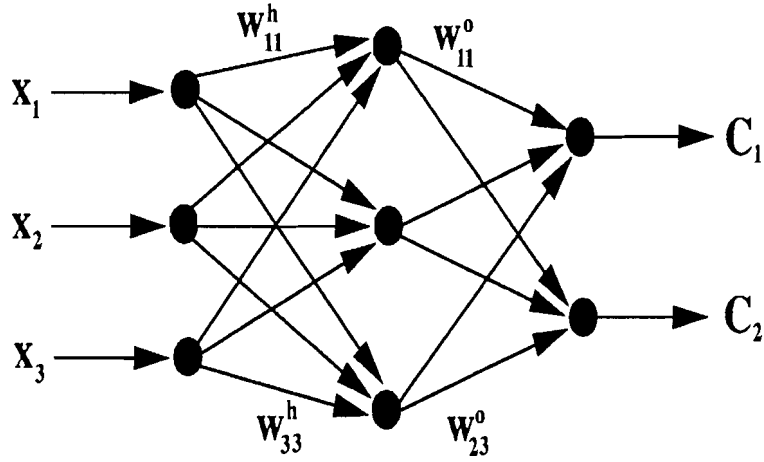
Based on the architectures, ANNs can be classified into the following three groups: 1) Feedforward, 2) feedback, and 3) feedforward and feedback. Although all these three types of networks can be used for classification, generally feedforward networks are used for classification. Here, we shall describe the following three feedforward neural networks that are useful for classification.

**Feedforward neural networks with backpropagation algorithm:** A feedforward neural network (FFNN) with backpropagation (BP) algorithm consist of elementary processors arranged in a distributive fashion so that the whole network can classify patterns in an autonomous manner. Specifically, the network consists of several layers where each layer contains several processing units (Fig 2.3). There is a complete connection between the layers, but there is no connection within the layers. The input-output relation is captured through the change of weights associated with the connections. Given a training set of input-output pairs  $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$ , the backpropagation algorithm provides a supervised procedure for changing the weights in an FFNN to classify the given input patterns correctly. It uses supervised learning mechanism implemented in two phases. In the forward phase, the input pattern  $\mathbf{x}_i$  is propagated from the input layer to the output layer, and as a result, it produces an actual output  $\mathbf{o}_i$ . Then, in the second phase, the error signal resulting from the difference between  $\mathbf{y}_i$  and  $\mathbf{o}_i$  is backpropagated from the output layer to the previous layer to update the weights. The weight updating continues until the error becomes very small [SY96]. Note that the classification mechanism adopted here is direct classification. The advantage of this method is that it can partition the input space  $X$  even when the class boundary is very complicated. But the disadvantage with this approach is that it takes a long time to train, and in many cases the training may not converge.

**Radial basis function neural networks:** A radial basis function neural network [HH93] is a three-layer network (Fig. 2.4), whose output nodes form a linear combination of the basis functions computed by the hidden layer nodes. The basis functions in the hidden layer produce a localized response to input stimulus. Hence, they produce a significant nonzero response only when the input falls within a small localized region of the input space. Popularly used basis function is of the following type:

$$o_j^h = \exp \left[ -\frac{(\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)}{2\sigma_j^2} \right] \quad (2.2)$$

where  $o_j^h$  is the output of the  $j$ th hidden node,  $\mathbf{x}$  is the input pattern of dimension  $N$ ,  $\mathbf{m}_j$  and  $\sigma_j^2$  are the center and variance (assume that the variance is same along all dimensions)



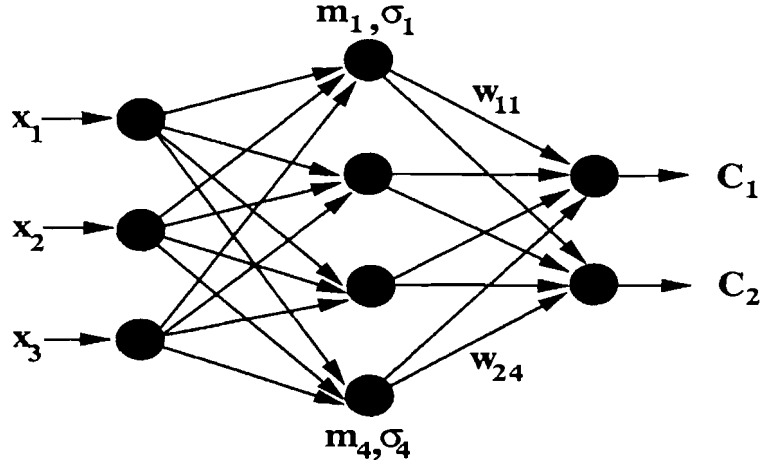
**Fig. 2.3:** A three layered feedforward neural network. It has three input nodes, three hidden nodes and two output classes. The input is  $\mathbf{x}$  and the output is the class confidence values for the classes  $C_1$  and  $C_2$ .

of the Gaussian functions of the  $j$ th hidden node, respectively. The hidden node outputs are in the range from zero to one such that the closer the input is to the center of the Gaussian, the larger is the response of the node. The output layer node equations are given by

$$o_k^o = \frac{\sum_j o_k^h w_{kj}}{\sum_j w_{kj}} \quad (2.3)$$

where  $o_k^o$  is the output of the  $k$ th output node and  $w_{kj}$  is the weight from the  $j$ th hidden node to the  $k$ th output node. The class label of the input  $\mathbf{x}$  is assigned as  $c$  where  $o_c^o = \max\{o_1^o, o_2^o, \dots, o_C^o\}$ . In radial basis function neural networks, the parameters used in the hidden layer are generally obtained through supervised clustering. The weights between the hidden and output layer are learned in a supervised fashion using **Widrow-Hoff** learning rule [Hay94]. Note that here classification is carried out through clustering. The advantage of this method is that the training is fast. However, if the estimated number of clusters and the cluster structure are not close to the original one, the classification result may be poor.

**Probabilistic neural networks:** A probabilistic neural network is a three layered feed-forward network [Spe90] [RY98] consisting of an input layer, a pattern layer and a summation layer (Fig. 2.5). The input layer contains  $N$  nodes to accept an  $N$ -dimensional input pattern. The pattern layer consists of  $C$  pools of pattern units, where  $K$ th pool contains  $S_k$  number of pattern units. Here,  $C$  is the number of classes, and  $S_k$  is the number of training patterns for the class  $C_k$ . Each node in the pattern layer is connected from every



**Fig. 2.4:** A typical radial basis function neural network. It has three input nodes, four hidden nodes and two output classes. The input is  $\mathbf{x}$  and the output is the class confidence value corresponding to the classes  $C_1$  and  $C_2$ .

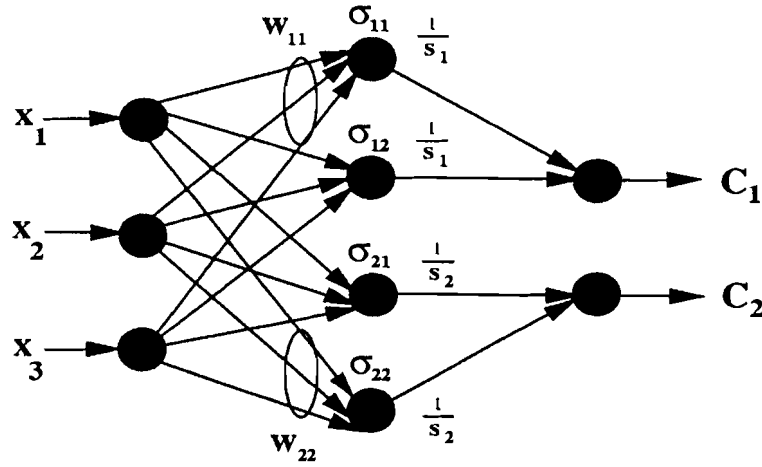
node in the input layer. The summation layer consists of  $C$  number of summation units, one unit for each pool in the pattern layer. Pattern units of the  $k$ th pool in the pattern layer are connected to the corresponding  $k$ th summation unit in the summation layer.

Training of the network consists of storing each training vector  $\mathbf{x}_j$  ( $1 \leq j \leq S_k$ ) of the class  $C_k$  as the weight  $\mathbf{w}_{kj}$  connecting the input layer and the  $j$ th pattern unit in the  $k$ th pool of the pattern layer. The connection weight from each pattern unit in the  $k$ th pool and the summation unit for the  $k$ th class is assigned as  $\frac{1}{S_k}$ . Note that the training is a one-pass supervised algorithm, and hence, it is trivial in this case. For any input vector  $\mathbf{x}$ , output of the  $j$ th pattern unit belonging to the  $k$ th pool is  $\frac{1}{\sqrt{(2\pi)^N \sigma_{kj}^2}} \exp\left(-\|\mathbf{x} - \mathbf{w}_{kj}\|^2 / 2\sigma_{kj}^2\right)$ , where  $\sigma_{kj}$  is a smoothing parameter for the Gaussian activation function of that unit. Output of the  $k$ th summation unit is [RY95]

$$o_k^o = \frac{1}{S_k} \sum_{j=1}^{S_k} \frac{\exp\left(-\|\mathbf{x} - \mathbf{w}_{kj}\|^2 / 2\sigma_{kj}^2\right)}{\sqrt{(2\pi)^N \sigma_{kj}^2}} \quad (2.4)$$

The class label of the input  $\mathbf{x}$  is assigned as  $c$  where  $o_c^o = \max\{o_1^o, o_2^o, \dots, o_C^o\}$ .

The problem with this network is that the testing time is very high. To alleviate it, supervised clustering scheme can be adopted. But, if the estimated number of clusters and the cluster structure are not close to the original one, then the classification result may decrease significantly.

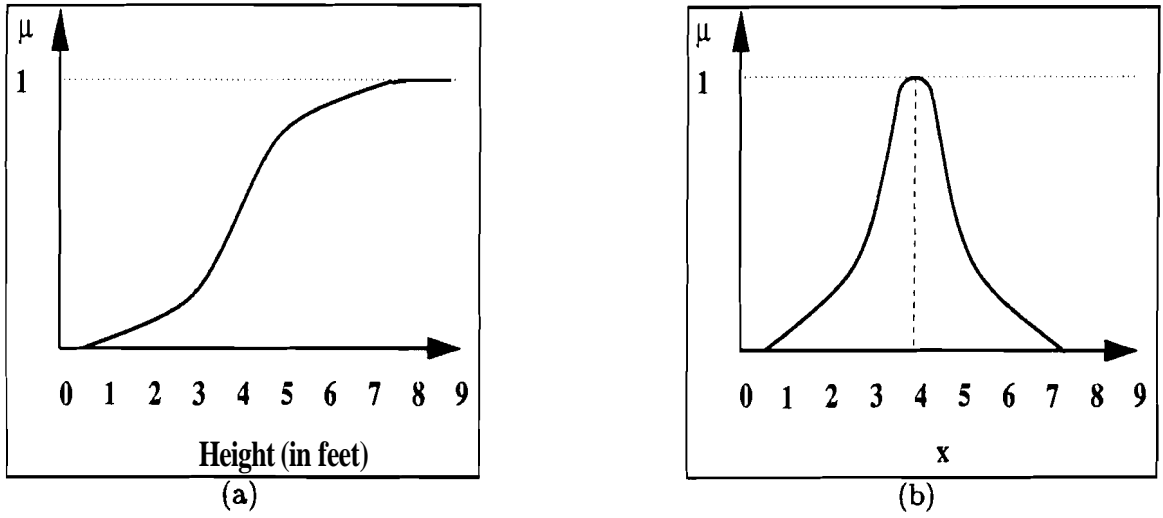


**Fig. 2.5:** A typical probabilistic neural network. It has three input nodes, four hidden nodes and two output classes. The input is  $\mathbf{x}$  and the output is the class confidence value corresponding to the classes  $C_1$  and  $C_2$ .

There exist various other interesting neural networks classifiers like **Hopfield** networks, **Kohonen's** self-organization map (SOM) networks [Hay94] [RY95], adaptive resonance theory (ART) networks [Hay94], etc.

### 2.2.5.3 Fuzzy classifiers

The concept of fuzzy sets was first introduced by L. Zadeh in 1965 [Zad65], as a mathematical way to represent the vagueness present in the human reasoning. Fuzzy sets can be considered as a generalization of classical set theory. In the classical set, an element of the universe either belongs to or does not belong to a set. That is, the belongingness of the element is crisp—it is either yes (in the set) or no (not in the set). In fuzzy sets, the belongingness of the element can be anything in between yes or no; for instance, a set of tall persons. We cannot identify a person as tall in a *yes/no* manner, as there does not exist any well-defined boundary for the set tall [PP96] [ESY92]. A fuzzy set is mathematically a mapping (known as membership function) from the universe of discourse to  $[0, 1]$ . The higher the membership value of an input pattern to a class, the more is the belongingness of the pattern to the class [DP80] [Kan82] [Kan86] [KF93]. Therefore, any concept that uses fuzzy sets requires the membership function to be defined. This function is usually designed by taking into consideration the requirements and constraints of the problem. Fig. 2.6(a) shows one possible membership function for the set tall. There are many other possible membership functions for the set tall. Nonuniqueness



**Fig. 2.6:** Fuzzy membership functions for (a) fuzzy set *tall* and (b) fuzzy number *close to 4*.

of membership functions may raise a question: How does a designer know which one to use? In fact, the designer can obtain the membership function from an expert (subjective computation) or from the data (objective computation) [PP96] [Bez81] [BP92] [Bez96]. Following the idea of fuzzy sets, the concept of crisp numbers has been generalized to fuzzy numbers [KG85] (Fig. 2.6(b)). The reasoning with fuzzy sets and fuzzy numbers is known as *fuzzy logic* [Kos93].

Since many classical pattern recognition techniques are based on conventional set theory, fuzzy sets can be fruitfully used to generalize these techniques. In traditional two-state classifiers, where a class  $A$  is defined as a subset of a universal set  $X$ , any input pattern  $x \in X$  can either be a member or not be a member of the given class  $A$ . This property of whether or not a pattern  $x$  of the universal set belongs to the class  $A$  can be defined by a *characteristic function*  $\mu_A : X \rightarrow \{0, 1\}$  as follows:

$$\mu_A(x) = \begin{cases} 1 & : \text{if and only if } x \in A \\ 0 & : \text{if and only if } x \notin A \end{cases}$$

In real life situations, however, boundaries between the classes may be overlapping. Hence, it is uncertain whether an input pattern belongs totally to the class  $A$ . To take care of such situations, in fuzzy sets [Bez81] the concept of characteristic function has been modified to membership function  $\mu_A : X \rightarrow [0, 1]$ .

The training of a C-class classifier for a set of input patterns  $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$



is basically an assignment of the membership values  $\mu_c(\mathbf{x}_i)$  on each  $\mathbf{x}_i \in X$ ,  $\forall c = 1, 2, \dots, C$ ,  $\forall i = 1, 2, \dots, n$ . If the membership values are crisp, then  $X$  is partitioned into  $C$  subgroups during the training process. In the fuzzy context,  $C$  subgroups of  $X$  are the set of values  $\{\mu_c(\mathbf{x}_i)\}$  that can be conveniently arranged on a  $C \times n$  matrix  $U = [\mu_c(\mathbf{x}_i)]$ . Based on the characteristic of  $U$ , classification can be of the following three types [PB95]:

1. Crisp classification:

$$M_{hc} = \left\{ U \in \mathbb{R}^{Cn} \mid \mu_c(\mathbf{x}_i) \in \{0, 1\} \forall c, \forall i; \sum_{c=1}^C \mu_c(\mathbf{x}_i) = 1; 0 < \sum_{i=1}^n \mu_c(\mathbf{x}_i) < n \forall c \right\} \quad (5-a)$$

2. Constrained **fuzzy** classification:

$$M_{fc} = \left\{ U \in \mathbb{R}^{Cn} \mid \mu_c(\mathbf{x}_i) \in [0, 1] \forall c, \forall i; \sum_{c=1}^C \mu_c(\mathbf{x}_i) = 1; 0 < \sum_{i=1}^n \mu_c(\mathbf{x}_i) < n \forall c \right\} \quad (5-b)$$

3. Possibilistic classification:

$$M_{pc} = \left\{ U \in \mathbb{R}^{Cn} \mid \mu_c(\mathbf{x}_i) \in [0, 1] \forall c, \forall i; 0 < \sum_{i=1}^n \mu_c(\mathbf{x}_i) < n \forall c \right\} \quad (5-c)$$

It is obvious  $M_{hc} \subset M_{fc} \subset M_{pc}$ . The implementation of the crisp classifiers has been discussed in the context of crisp rule base system (section 2.2.5.1). If we replace the membership values by probabilities, then the constrained fuzzy classifiers become the probabilistic classifiers (discussed in section 2.2.5.2). The interpretations of the output (say  $\mathbf{a}$ ) for the two classifier models are different. The probabilistic interpretation means that *the probability that the input pattern belongs to the class  $C_c$  is  $\mathbf{a}$* . On the other hand, the fuzzy interpretation is as follows: *The grade of membership of the input pattern to the class  $C_c$  is  $\mathbf{a}$* . The first statement implies that if we take the same input pattern  $n$  times,  $\alpha n$  times it belongs to the class  $C_c$  [Ped90] [Bez94]. In contrast, the second statement expresses that the input is close to the center of the class prototype  $C_c$  with a degree  $\mathbf{a}$ , that is, the input is similar to the class  $C_c$  with a degree  $\mathbf{a}$ . The user is not interested in repeating the experiments but in the class assignment. Therefore, in many cases it may be appealing to consider the output of the classifiers as fuzzy membership values rather than *a posteriori* probabilities. In these cases fuzzy sets can be used to implement the classifiers based on the constrained fuzzy classification and possibilistic classification.

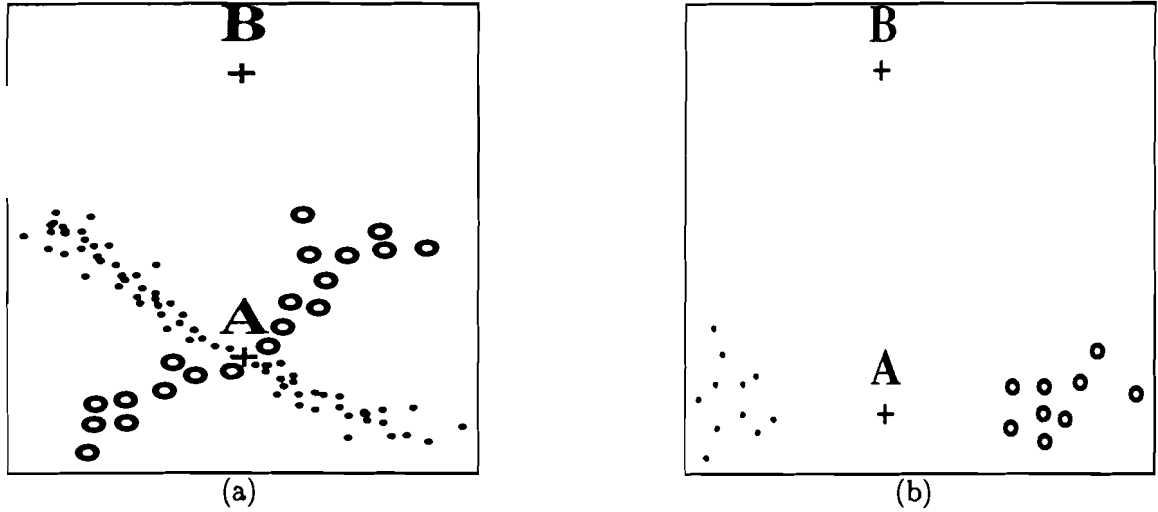
Fig. 2.7 examines the role played by crisp, constrained fuzzy and possibilistic classification approaches in a 2-class problem. Here, both the patterns **A** and **B** are equidistant

from the two classes. In crisp classification, the membership value of A in one class will be 1, and in the other class it will be 0. It is true for the pattern B also. Obviously, this kind of membership assignment does not reflect the actual classification situation as A and B partially belong to both the classes. In constrained fuzzy membership assignment, both the patterns A and B will be assigned the membership values equal to 0.5. Although this membership assignment is better than the crisp counterpart, it fails to consider the pattern A as a more typical one than the pattern B. It is because, here the membership assignment is a relative one, and it depends on the membership values to both the classes. In possibilistic membership assignment, the pattern A will receive equal membership values to both the classes. It is true for B also. But, the membership of B to any class will be always less than that of pattern A. Therefore, the possibilistic assignment may not be summed up to one, and thus, it can distinguish between equal evidence and ignorance [Zad78] [KK93] [Sha76] [DP88]. This property of the possibilistic assignment makes it attractive compared to the crisp membership assignment and constrained fuzzy membership assignment.

Broadly speaking, there are the following four ways to apply the fuzzy classification techniques:

- **Fuzzy relation approach:** The input and output of any classifier system is supposed to be related by some relation. If there is no such relations, it is impossible to build any classifier. On the other hand, if there exists any relation, in a crisp case, any two points (one from input space and another from output space) from the input-output space are either related or not. In fuzzy relations, these two points can be related with a varying degree. The value of the degree is expressed as a membership value that lies in between 0 and 1. Therefore, the fuzzy relation subsumes the crisp relation. The search for a structure involves discovering the fuzzy relation. One scheme to realize the fuzzy relation is construction of a fuzzy rule base system [DHR93]. A fuzzy rule base system consists of a set of fuzzy if-then rules like

If a person is very tall and *very* fair, then he is a European with high confidence.  
 If a person is very tall and *fair*, then he is an African with low confidence.



**Fig. 2.7:** (a) The crisp membership values of pattern **A** and **B** in both the classes are either 0 or 1. The constrained fuzzy membership values of the pattern **A** and **B** in both the classes are about 0.5, which does not consider the fact that **B** is much less representative of either class than **A** is. (b) The crisp membership values of pattern **A** in both the classes are either 0 or 1. The constrained fuzzy membership values of the pattern **A** in both the classes are about 0.5. On the other hand, possibilistic membership values of the pattern **A** in both the classes are 1 as it belongs to both the classes completely.

where the terms tall, fair are called fuzzy linguistic values. The fuzzy rule-base system is useful where it is difficult or impossible to model the given classification system with classical approach. In this case a set of fuzzy if-then rules along with the fuzzy linguistic values are collected from experts. If a new input comes, the input is matched against the if part of each if-then rule, and the response of each rule is obtained through fuzzy implication. The response of each rule is weighed according to the extent to which each fuzzy rule fires. The response of all the fuzzy rules for a particular output class are combined to obtain the confidence with which the input is classified to that class. The final class label can be determined by taking the class with maximum confidence. It can be observed that there is no learning associated with the fuzzy-rule base system. Consequently, the designer has to rely completely on the expert's opinions to build the rule base, which may

be difficult in some cases.

- a **Fuzzy pattern matching approach:** A slightly different way of classification is the information fusion approach offered by fuzzy integrals. Here, a decision to associate an input pattern to a class is accomplished through the fusion of the information coming from several sources in form of features. Fuzzy integral combines the objective evidence supplied by the features in a nonlinear way with the importance of that feature set for recognition purpose. Instead of treating each feature identically, it stresses those features or sets of features which provide the most evidence toward the determination of class memberships. Therefore, it results in a convenient framework to produce different nonlinear classification rules for different classes within the same problem and with the same over all feature set [KQ88] [Gra96]. However, when the number of features is  $N$ , this technique may need  $O(2^N)$  computations.
- a **Fuzzy clustering approach:** Fuzzy clustering is similar to the conventional clustering as described in section 2.2.4. However, unlike the conventional one, in fuzzy clustering each input pattern belongs to all the clusters with different degrees or membership values. Thus, each cluster is a fuzzy set. **If** the sum of memberships of a pattern to all the clusters is equal to one, then it is called *constrained fuzzy clustering*. **If** the sum is not necessarily equal to one, then it is called *possibilistic clustering*. There is fuzzy K-means clustering algorithm which realizes the constrained fuzzy clustering method. Modifications of this algorithm form possibilistic angle are known as possibilistic K-means algorithm [KK93] and mixed K-means clustering algorithm [PPB97]. If the value of  $K$  is not close to the actual number of clusters, then the clustering result may be far away from the reality. To know the approximate number of clusters present in the data set, various indices like partition coefficient and entropy indices [Bez81], Xie-Beni index [XB91], Fukuyama-Sugeno index [FS89] [PB95], fuzzy hypervolume [GG89], etc., exist. The clustering algorithms can be used to group the input data set. Then a class label (crisp or fuzzy) is assigned to each cluster. Thus, a classifier can be constructed through fuzzy clustering.
- **Other approaches:** Among the other methods, fuzzy K-nearest neighbors algorithm [KGG85] and fuzzy decision trees are popular. In the conventional K-nearest neighbors algorithm, each neighbor is considered equally important to assign the class label to the input sample. However, when two classes overlap each other, a

more typical neighbor should be given more weightage. In fuzzy K-nearest neighbors algorithm, this philosophy is implemented. Thus, fuzzy K-nearest neighbors algorithm subsumes the conventional K-nearest neighbors algorithm, and in many cases the first one becomes more powerful than the later one. Like the conventional crisp K-nearest neighbors algorithm, the fuzzy counterpart also suffers from the problem of long testing time.

#### 2.2.5.4 Rough classifiers

In any classification task the aim is to form classes of objects which are not noticeably different. These *indiscernible* or indistinguishable objects can be viewed as basic building blocks (concepts) used to build up a knowledge base about the real world. For instance, if the objects are classified according to color (red, black) and shape (triangle, square and circle), then the indiscernible objects are red triangles, black squares, red circles, etc. Thus, these two attributes make a *partition* in the set of objects and the universe becomes coarse. If two red triangles with different areas belong to different classes, it is impossible for anyone to classify these two red triangles based on the given two attributes. This kind of uncertainty is referred to as rough uncertainty [Paw82] [PBSZ95] [Paw95]. The rough uncertainty is formulated in terms of *rough sets* [Paw82] [PBSZ95]. Obviously, the rough uncertainty can be completely avoided if we can successfully extract the essential features so that distinct feature vectors are used to represent different objects. But, it may not be possible to guarantee as our knowledge about the system generating the data is limited [SS93].

Let us consider a 2-class problem where each input pattern has only one feature. Two input patterns  $x_1$  and  $x_2$  are called related if  $x_1 = x_2$ . This is obviously an equivalence relation. From linear algebra we know that this equivalence relation partitions the input space into (say  $m$ ) equivalence classes. If all the patterns from an equivalence class (say  $[x]$ ) have the same label (let  $C_1$ ), then we can allot a single rule to describe the input-output relationship for all the patterns that belong to the equivalent class. The rule is

If the input is  $x$ , then the output class label is  $C_1$

Thus, by partitioning the input space into  $m$  equivalence classes, it is possible to obtain a rule base consisting of  $m$  deterministic rules. However, in presence of rough uncertainty, i.e., when more than one pattern from the same equivalence class carries a different label

(let  $C_2$ ), a one-to-many relationship exists between the equivalence class and the class labels. Hence we cannot use the deterministic rules any more. One possible way to describe the input-output relationship is to construct nondeterministic rules such as [Paw91]

If the input is  $x$ , then the output class label is  $C_1$  with confidence factor  $r_1$

If the input is  $x$ , then the output class label is  $C_2$  with confidence factor  $r_2$

where  $r_1$  and  $r_2$  can be determined from the input data. Note that here more than one rule is present with the same  $\#$  part. Let  $\underline{R}(C_1)$  represent the set of all equivalence classes, where each equivalent class contains patterns only from the class  $C_1$ . Let  $\overline{R}(C_1)$  represent all the equivalence classes, where each equivalent class contains some pattern from  $C_1$ . In  $\overline{R}(C_1)$  some equivalence class may contain patterns from classes other than  $C_1$  as well. Now, one simple scheme of assigning the value of  $r_1$  is  $r_1 = \frac{R(C_1)}{\overline{R}(C_1)}$ . Similarly,  $r_2$  can be assigned. This concept can be extended for an input with more than one feature. Thus, in general, any classification problem can be mapped onto two sets of rules—one set is deterministic and another set is nondeterministic. If there is no rough uncertainty, then the nondeterministic rules do not exist. In other words, the deterministic and nondeterministic rules are needed for the equivalence classes where the class labelling is not unique. When a new input comes, the input is matched with the  $\#$  part of each deterministic rule. A rule fires if there is a match. The class label corresponding to the input is decided by the rule that fires. If the input does not match with the  $\#$  part of any deterministic rule, then the input is matched with the  $\#$  part of the nondeterministic rules. The class label is decided based on the confidence factor associated with each nondeterministic rule.

One problem with the rough approach is that it is mainly applicable when the number of equivalence classes is small. When the features are continuous, the number of equivalent classes may be very high. To circumvent this problem, continuous features are usually transformed to discrete features with inevitable loss of information. Moreover, approaches based on rough set cannot be used where the input features or the output classes are fuzzy.

#### 2.2.5.5 Hybrid classifiers

Table 2.1 summarises the relative merits and demerits of artificial neural networks, fuzzy logic, rough sets and evolutionary computation techniques for pattern classification tasks. To exploit the rough and fuzzy uncertainties present in a classification process, it is beneficial to incorporate the concepts of rough sets and fuzzy logic in the framework of neural

networks. This kind of model is useful, because these three methods approach the design of classifiers from quite different angles. Neural networks supply the brute force method necessary to accommodate and interpret large amount of input data. Rough sets and fuzzy logic provide a structural framework that utilises and exploits these low level results. For efficient implementation of this kind of hybrid model, we need a good search and optimization strategy. For this purpose, one can use evolutionary programming, which represents a potentially powerful pathway to machine learning and self organization [AH95]. In what follows, we are focussing on these hybrid techniques from the pattern classification angle.

**Table 2.1:** Relative merits of artificial neural network (ANN), fuzzy logic (FL), evolutionary computation (EC) and rough set (RS). The symbols B, SB, SG and G represent bad, somewhat bad, somewhat good and good, respectively [JSM97]

Property	ANN	FL	EC	RS
Mathematical modeling	B	SG	B	SG
Learning ability	G	B	S	G
Knowledge representation	B	G	SB	G
Expert knowledge	B	G	B	S
Nonlinearity	G	G	G	B
Optimization ability	S	G	B	G
Fault tolerance	G	G	G	B
Uncertainty tolerance	G	G	G	G
Real-time operation	SG	G	SB	G

**Neuro-Fuzzy classifiers:** It has been recognized that the areas of neural networks and fuzzy logic are strongly interconnected [LL96]. An important connection between ANN and fuzzy logic-based systems is that both of them can approximate continuous functions [JSM97]. Use of fuzzy concepts in ANNs is also supported by the fact that the psycho-physiological process involved in the human reasoning does not employ precise mathematical formulation [PM86]. There are the following two approaches to fuse these

two approaches:

- **Fuzzy-neural networks:** This type of classifier consists of an ANN equipped with the capability of handling fuzzy information. Specifically, fuzziness can be incorporated in an ANN at the following levels: (a) At output and target levels, (b) at input level, and (c) at each neuron level in terms of weight value, basis function and, output function. The appropriate level of incorporation of the theory of fuzzy logic depends on the given problem.
- **Neural-fuzzy systems:** This type of classifier consists of a fuzzy system augmented by ANNs to enhance the flexibility, speed and adaptability of the fuzzy system. For instance, neural networks can be used to tune the membership values or fuzzy rules. Thus, neural network learning can reduce the development time and cost while improving the performance of a fuzzy system.

While training an ANN for a classification task, we generally use crisp target values, which can be either zero or one. This kind of target assignment can be generalized by exploiting fuzzy sets, where target values can be anything in between zero and one. In [PM92] ANN outputs are interpreted as fuzzy membership values, and using this idea the conventional mean square error objective function has been extended to various fuzzy objective functions. The learning laws are derived by minimizing the fuzzy objective functions in a gradient descent manner. It has been found that incorporation of fuzziness in the objective functions leads to better classification rate.

ANNs adopt numerical computations for learning. But numerical quantities suffer from lack of representative power [Pao89]. There are many applications where information cannot be obtained in terms of numerical values. Instead it is possible to represent the information in linguistic values only [LL95] [KL79]. In [WM97] Wang *et al.* have proposed fuzzy basis functions to design a radial basis function network [Hay94], which can accept both numerical inputs as well as fuzzy linguistic inputs. In [Ped92] Pedrycz has proposed an ANN model based on fuzzy logical connectives. Instead of using linear basis functions, he has utilized fuzzy aggregation operators. In [PR93] and [HP94], this technique has been extended to a more general one where inhibitory and excitatory characteristics of the inputs are captured by employing direct and complemented, i.e., negated input signals. The advantage of this approach is that problem-specific fuzzy *a priori* knowledge can be incorporated into the network easily. In [IFT93] Ishibuchi *et al.* have proposed an ANN learning algorithm where expert's *a priori* knowledge, in terms of fuzzy *if-then* rules, can



be exploited to learn the information supplied by the numerical data.

Fuzzy logic can be employed to speed up the training of an ANN. In [CAMC92] a fuzzy rule base is used to dynamically adapt the learning rate and momentum parameters of a feedforward neural network with backpropagation learning algorithm. In a similar approach [COB92], Choi *et al.* have proposed an incremental updating scheme to control the value of vigilance parameters of ART networks.

The difficulty in constructing fuzzy rule base systems is that the membership function, number of rules and precedent parts of the rules can be supplied only by the experts. In many cases, it is difficult to get an expert, and in some cases, even for the experts it becomes difficult to construct the rules. This problem can be reduced if ANNs learning mechanism can be incorporated in the fuzzy rule base systems to construct fuzzy neural systems. Since ANNs can approximate continuous functions, ANNs (for example, feedforward neural networks with backpropagation algorithm) can be used to realise membership functions with arbitrary shape. If the membership function has a regular shape like bell shape or triangular shape, it can even be modeled by a simple neuron with sigmoidal function [LL96]. Fuzzy OR and fuzzy AND need min and max operators. Since they are nondifferentiable, it is difficult to learn them. Hence, the concept of differential minimum and maximum operators, i.e., *softmin* and *softmax*, have been introduced [LL96]. Thus, fuzzy logic connectives, i.e., fuzzy NOT, fuzzy OR and fuzzy AND, are modeled by ANNs. Keller *et al.* have proposed [KKR92] *fuzzy inference network*, where membership functions and fuzzy logic connectives are implemented through ANNs. This network is made more powerful by incorporating learning facilities. The resultant network is called *fuzzy aggregation network* [KT92].

In many cases, it is possible to view the same classifier as a neural network and a fuzzy rule base system. For instance, feedforward neural networks with backpropagation algorithm and radial basis function network can be seen as fuzzy rule base systems [BCR97] [JS93] [HHMS96]. In these networks the output functions present in the hidden nodes act as the membership functions for some linguistic values.

**Neuro-Rough classifiers:** In ANNs one critical problem is to determine how many input units are necessary. Obviously, it depends on the number of features present in the input data. Using rough sets, it may be possible to decrease the dimension of the input data without losing any information. A set of features is sufficient to classify all the input

patterns if the rough ambiguity for this set of features is equal to zero. If we know the amount of rough ambiguity present, then using it as a criterion we can select a proper set of features from the given set of data [PWZ88]. In [PBSZ95] it is claimed that for a classification task the number of hidden units needed in a feedforward neural network is equal to the minimal number of features required to represent the data set without increasing the rough uncertainty. One way to accelerate the training of a network is to initialize the weights of the networks in such a manner that the initial decision region is closer to the desired one. For that, a set of training data is collected, and the knowledge extracted from them through rough sets is used to initialize the ANN [BMP97].

**Neuro-Evolutionary classifiers:** From a mathematical point of view, all the evolutionary computation (EC) techniques are *controlled, parallel, stochastic search and optimization* techniques. Since different learning techniques used in ANNs hinge on the optimization of various objective functions, it is possible to employ EC for learning weights, learning network architectures, learning the learning laws, input feature selection, etc. [Yao93]. For instance, in a feedforward neural network, gradient-based local search methods [Hay94] can be substituted by EC for weight training [SF95] [MTH89] [Has95]. In some cases, a more ambitious approach may be to exploit local search methods like gradient descent, and global search methods like EC, simultaneously [RF96]. The advantages of local search methods are better accuracy and fast computation. The disadvantages of the local search methods are stagnation at the suboptimal solutions and sensitiveness to the initialization. On the other hand, EC is a global search method which can avoid local optima, and does not have the initialization problem. However, EC can suffer from extremely slow convergence before arriving at the accurate solution. This is because, EC uses minimal *a priori* knowledge, and does not exploit available local information [RF96]. In fact, in the search space EC is good for exploration, whereas the gradient descent is good for exploitation. Therefore, by utilizing both of them, merits of both methods, i.e., speed, accuracy, reliability and fast computation, can be achieved. Yao *et al.* [YL97] have proposed one such method to evolve the topology (weights and architecture) of a feedforward neural network, where they exploit both evolutionary programming and backpropagation algorithm, simultaneously. In EC techniques, the whole population set evolves over and over again, generations after generations. At the last generation, the network which has the highest fitness is considered to be the desired optimal network for the given task. Instead of choosing a single network as the desired network, in [YL98]

all the networks in the population are considered as the desired networks. The final result is obtained by combining all the individuals in the last generation to make best use of all the information contained in the whole population. This result confirms the fact that a population contains more information than a single individual, and EC is used to exploit that information. In particular, in [YL98] the outputs of all the networks in the population are combined, and the output class is determined by a majority voting method. In [WC96] [Whi96] [BZ95], EC-based techniques are used successfully to optimally configure radial basis function networks so that the networks generalize well. In another development, Angeline *et al.* [ASP94] have used EC to configure recurrent neural networks. It should be noted that, gradient-based approach is not useful here as it needs the objective function to be differentiable. Using EC, Jockusch *et al.* [JR94] have introduced a training strategy for self-organizing maps [Koh89] [Koh90], where it is possible to find the number of output units for the self-organizing maps automatically. In their scheme the training of the self-organizing maps is less likely to be stuck in local optima. Currently, researchers are working on different evolutionary methods which can be utilized to learn weights, architectures and learning laws, simultaneously [Yao93]. The search space for these problems are prohibitively large, and they need a large computing time. These drawbacks may be reduced if parallel machines are used to implement the search operation [BR94], or the search operation is made more efficient and less time consuming by using adaptive EC operators [SP94] [LD95].

**Fuzzy-Rough classifiers:** Rough set theory was proposed as a mathematical tool in order to deal with inexact, noisy or incomplete information. It aims to provide a formal framework for automated transformation of data into classification knowledge. The starting premise is that the universe is coarse and some object in the data set may become indiscernible, resulting in a partition of the universe. In contrast, fuzzy set theory provides an effective means of handling uncertainty in various systems, including those in the application of rough set theory. But the premise of granularity in knowledge is absent in fuzzy set theory, and the focus is on the fact that concepts in the universe of discourse tend to be gradual rather than crisp. Therefore, rough sets are a calculus of partitions, while fuzzy sets are a continuous generalization of set-characteristic functions [DP92] [Slo92]. Hence, it is possible to integrate roughness and fuzziness, and the resultant model of uncertainty is expected to be stronger than either. These hybrid notions develop in a natural way when a linguistic category, denoting a set of objects, must be approximated

in terms of already existing labels, or when the indiscernibility relation between objects no longer obeys the ideal laws of equivalence, and the relation is a matter of degree.

Direct application of rough sets to minimize a fuzzy rule base classifier was proposed by Tanaka et al. [TI92] [TIS92]. They extracted a fuzzy rule base for a given medical classification problem. They also proposed a method to reduce significantly the number of input variables of the fuzzy rule base. The advantage of their method is that the inconsistency of the test data and experts' diagnoses can be clarified, and inconsistent data can be removed. In addition, unlike the rule base generated by rough approach (see section 2.2.5.4), the fuzzy rule base is applicable to inputs with continuous features.

**Fuzzy-Evolutionary classifiers:** Currently, the combination of EC and fuzzy logic is taking place mainly along the following two directions [HHVLV94] [CHL96] [HM97]:

1. The use of fuzzy logic-based techniques for either improving EC behavior and modeling EC components, or to manage problems in an imprecise environment, where the imprecision is modeled by fuzzy sets.
2. The application of ECs in various optimization and search problems involving fuzzy systems.

Several techniques related to fuzzy logic have been used for improving EC behavior and modeling EC components (like using fuzzy connectives to design crossover in genetic algorithm, fuzzy population diversity measure, etc. [CHL96]). These techniques concern different parts of EC development in the following ways:

1. Expert knowledge (represented as fuzzy rules) is used to compute dynamically the EC parameters. The aim is to obtain suitable exploitation/exploration relationships throughout the EC execution. In this way, a knowledge base is used for controlling the evaluation process and for avoiding the undesired behavior, like premature convergence.
2. A fuzzy stop criterion forces the EC to reach optimal solutions with a user-defined accuracy.

There are two main directions for applying ECs in a fuzzy environment. The first one exploits an EC to manage fuzzy valued variables. In the second approach, the variables consist of associated fuzzy sets, and hence, the fitness is actually a fuzzy valued fitness.

The first proposal considers variables with fuzzy values in the representation, and the second proposal considers nonfuzzy value variables but with a fuzzy evaluation (e.g., fuzzy fitness).

In [BH94] [FS93] [YKSS95] attempts are made to develop EC-based clustering techniques. These algorithms are less prone to get stuck in local minima. Moreover, they do not suffer from the initialization problems as observed in the fuzzy K-means algorithm. These clusters are used to construct fuzzy rules. In [INYT95] [NIT96] genetic algorithm is used to evolve a fuzzy rule base system suitable for a given classification task. A good bibliography on fuzzy-evolutionary techniques is provided in [OC97].

**Rough-Evolutionary classifiers:** One major difficulty in a classification problem is to find the optimal number of features. It can be viewed in rough set domain as to discover the minimum number of features necessary for the classification problem without increasing the rough uncertainty associated with the classification task. Although it is possible to apply brute force method to find all possible combinations of the features, and subsequently take the best one as the optimal one, it may involve large amount of computation. This problem may be reduced if we use EC to find the minimum number of features [PBSZ95]. Here, rough set theoretic measure acts as a guideline to choose the correct set of features [Wro95] [HPA<sup>+</sup>97].

In addition to the above techniques, hybrid techniques like neuro-fuzzy-rough, **neuro-fuzzy-evolutionary**, fuzzy-rough-evolutionary, neuro-fuzzy-rough-evolutionary, etc., are also possible. These integration techniques are based on partnerships, in which each of the partners contributes a distinct methodology for solving the problem [AH95]. There are several other attractive paradigms which can be used for the hybrid techniques. For instance, *artificial ant system* [DMC96], *cultural evolution* [Bel89], *immunity net* [HC96], *cellular automata* [TM87] and *DNA computing* [Adl94] seem to be attractive and viable approaches.

### 2.2.6 Generalization

A great deal of episodic evidence has been presented in the literature to support the claim that, once a classifier has been trained on a sufficient number of samples, it can then label a new and previously unseen input. It should be noticed that, without the ability to generalize, much of the cases for using classifiers would simply collapse. A simple look up table would suffice if one were interested merely in constructing a classifier that could reproduce the known input-output pairs [Nee96] [Vid97]. The training of a classifier with a data set leads the classifier to learn the input-output relationship. When a test set is applied, this relationship is extended so that it holds for the test data. However, there can be more than one relationship if the training data sequence, classifier size, learning algorithm, etc., are varied. All these relationships may be valid on the given training set, however, when extended on a different test set many of them may not be valid. Hence, the task of the generalization is to choose the relationship which holds for most of the test data [HKP91]. In order to choose the best relationship, we should be able to impose certain constraints on all possible input-output relationships. It can be accomplished if there is a structure present in the feature space, and the relationship between this structure and the class labels is not random. Since the parameters for classifier design depend on the structure in the feature space and the input-output relationship, the generalization capability of a classifier is largely influenced by the following three factors:

1. **Training data:** This refers to how well the training set consisting of input-output pairs represent the input-output relationship. Obviously, if the input-output relationship is noisy or random, then the generalization ability of the classifier becomes very poor.
2. **Size and structure of the classifier:** If the size of the classifier is large, then it needs a large number of parameters. It may lead to *memorisation* of the training examples if the training set size is not large. Moreover, while training a large classifier, all the parameters may not get involved in the training process as they balance each others effect on the output. Consequently, training error becomes low. However, such free parameters may result in a large variation of the classification efficiency for different test sets [Sus92]. As a consequence, the structure present in the data set is not captured and the generalization ability remains low. In contrast, if the classifier size is small, then it may not be sufficient enough to capture the input-output relationship.

3. Training methodology: The training algorithm, the presentation of the input data during training, the stopping criterion, all affect the performance of the classifier. Use of improper stopping criteria may cause overtraining which may lead to memorisation.

In order to study the generalization capability, we must be able to quantify it. That is, it should be possible to evaluate a classifier, and decide whether its generalization is "good" or not. However, the notion of "good" or "reasonable" themselves are not well-defined. It varies from person to person and is problem dependent. For instance, when the desired output is obtained on most occasions, it is considered as "good" generalization in certain cases. On the other hand, in certain other types of problems, generalization is considered to be "good" if the classifier yields the desired output for a very rare situation which never occurred before. Various methods of measuring generalization are used in practice [Liu95], [MCHK94]. We can classify them into two categories: measure of model fit and measure of *performance*. The first one measures how close the actual classifier function is to the desired one based on the training result. The second one focuses on the difference between the actual classification rate and the desired classification rate after the final class label is assigned to each test pattern with inevitable loss of information. Therefore, the first one measures how good the approximation is, based on the training and when there is no loss of information. The second one stresses on how good the approximation is after the information is lost due to crisp labelling on the test patterns.

Let  $n$  be the number of patterns in the training set  $X = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$ ,  $C$  be the number of output classes and  $y$  be the desired output corresponding to the input  $\mathbf{x}$ . Then, some measures based on the model-fit criterion are

1. Kullback-Leibler measure: This measure of generalization quantifies the difference between the actual classification function and the classifier function obtained by training [Hay94]. The Kullback-Leibler measure ( $e_{kl}$ ) is given mathematically by the following equation:

$$e_{kl} = - \int p(\mathbf{x}, \mathbf{y}) \log \left[ \frac{f(\mathbf{y}, \mathbf{x})}{p(\mathbf{y}, \mathbf{x})} \right] d\mathbf{x}d\mathbf{y} = - \int p(\mathbf{x}, \mathbf{y}) \log \left[ \frac{f(\mathbf{y}|\mathbf{x})f(\mathbf{x})}{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})} \right] d\mathbf{x}d\mathbf{y} \quad (2.4)$$

where  $p(\mathbf{y}|\mathbf{x})$  is the conditional probability distribution of the sample  $\mathbf{x}$  given the output  $\mathbf{y}$ ,  $p(\mathbf{x})$  is the probability distribution of the input  $\mathbf{x}$ ,  $f(\mathbf{y}|\mathbf{x})$  is the probability distribution approximated by the classifier after training and the integral is over

the whole input-output space. Since  $p(\mathbf{x}) = f(\mathbf{x})$  for a given input distribution, we can redefine  $e_{kl}$  as

$$e_{kl} = - \int p(\mathbf{x}, \mathbf{y}) \log[f(\mathbf{y}|\mathbf{x})] d\mathbf{x} d\mathbf{y} \quad (2.5)$$

The Kullback-Leibler measure is difficult to calculate as it requires prior knowledge about the actual classification function  $p$  being realized.

2. **Cross-Validation measure:** Cross-validation measure estimates generalization error by making use of the training data [Liu95]. In this method, generalization error  $e_{kl}$  given by equation (2.5), is estimated as follows:

$$e_{cv} = -\frac{1}{n} \sum_{i=1}^n \log(f(\mathbf{y}_i|\mathbf{x}_i)) \quad (2.6)$$

3. **Mean square error measure:** One of the the most commonly used measure of generalization for the pattern classification task is how large the Euclidean distance is between the actual output of the classifiers (after the training is over) and the desired one. Like the previous two cases, this measure depends on the training set only. The measure is

$$e_{mse} = \frac{1}{nC} \sum_{i=1}^n \left[ (y_{i1} - o_{i1})^2 + (y_{i2} - o_{i2})^2 + \dots + (y_{iC} - o_{iC})^2 \right] \quad (2.7)$$

where  $\mathbf{o}_i$  is the actual output when the training pattern  $\mathbf{x}_i$  is applied to the classifier.

4. **An information criterion (AIC):** This measure [Aka74] is also known as *Akaike's information criterion*. Although memorisation trend in a classifier is related to the size or the number of the free parameters ( $k_f$ ) of the classifier, the above three measures do not consider the classifiers size. AIC is a measure which considers mean square error measure as well as the size of the classifier. It is formulated as

$$e_{AIC} = n \log \left( \frac{e_{mse}}{n} \right) + 2k_f$$

A popular measure based on the performance criterion is *error rate measure*. The easiest way to assess the error rate is to choose a misclassification count on the test set.

$$e_{er} = \frac{1}{n_v C} \sum_{c=1}^C (\text{no. of misclassifications for class } c)$$



where  $n_v$  is the number of test samples. This measure is extensively used because it is simple and easy to implement. It can be viewed as a variation of the cross-validation measure. If we have some idea about the *a priori* probability  $P_c$  for the  $c$ th class, the above measure can be modified as

$$e_{er} = \frac{1}{n_v C} \sum_{c=1}^C P_c (\text{no. of misclassifications for class } c)$$

Note that in all the above cases the less the measure is, the better is the generalization capability. Other than the above measures, there exist many other measures, e.g., *leave-one-out* measure, *entropic* measure, BIC, etc. [Rip96].

Although a relatively small fraction of the overall work done on the pattern classifiers is on the theoretical analysis of generalization, these studies are marked by a variety of approaches. Some of the issues like how much training data is needed, i.e., sample complexity, and how much time is needed for a particular level of generalization, i.e., computational complexity, are formalised and investigated within the field of computational *learning* theory. One popular approach in computational learning theory is probably approximately correct (PAC) learning theory approach [Hau92] [BEHW89] [Nat91]. Most of the theoretical studies assume noiseless synthetic input data, where the raw data represent features. The input-output relationships are assumed to be random. Application of the theoretical results are still limited because the results inferred based on these assumptions are far from the real life situations. Despite these limitations, the theoretical results indeed give us some idea about the extent of the influence of the classifier size, classifier architecture and the training set size on generalization. Due to the limitations of the theoretical studies, there are several heuristics adopted to enhance the generalization capability of the classifiers. These techniques can be broadly classified into two parts:

1. **Problem-independent techniques:** These methods deal with the functioning of the classifiers, methods of presentation of the data, etc. They include pruning of the extra parameters used in the classifiers, generating more training data by introducing noise, accelerating the training algorithms (so that large data set can be used for training), stopping training after some point (so that overtraining cannot take place).
2. **Problem-dependent techniques:** These methods include special design of the classifiers after taking care of the problem-specific knowledge. For example, if we know that the clusters formed in the input space is of shell type, then we can cluster

the input by using fuzzy shell clustering algorithm [FK96]. Classification can be achieved subsequently by labelling the clusters. We may also incorporate domain specific constraint to limit the classification function realizable by the classifier. In other words, we can bias the classifier to have less variance in the performance of the classifier [Bis95].

### 2.2.7 Conclusion

From the review presented in this chapter, it is quite obvious that the stages of pattern classification are involved. As Bezdek rightly pointed out in [Bez81] that if we could choose "optimal" features, clustering and classification would be trivial. But we often attempt to discover the optimal features by clustering the feature vectors. Also, if we could design an "optimal" classifier, then the features selected would be immaterial. The current state of research in pattern classification can be characterized as follows:

1. Basic concepts of pattern classification are being used in many real life applications.
2. Theoretical foundation have gained substantial base.
3. Most of the current research methods use numerical representation. This trend is growing because of the availability of the computing power.
4. Efforts are being made to combine the symbolic and numerical approach for problems in which such information can be obtained from physical systems.
5. The role of uncertainties in a given problem is being examined to avoid possible loss of information. The uncertainties captured in the initial stages are exploited for problem solving till the final stage of solution.

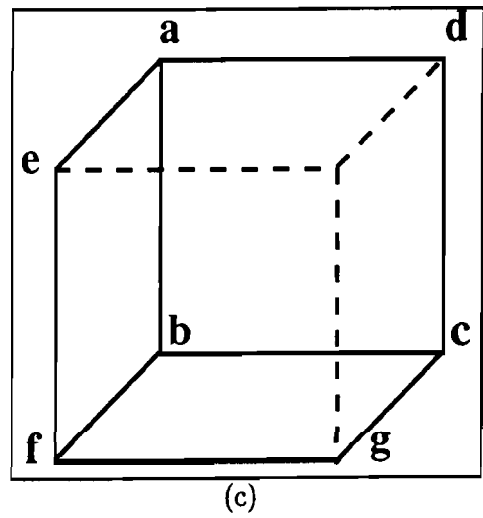
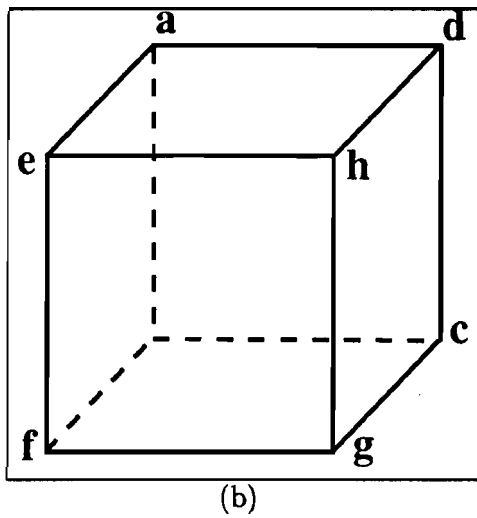
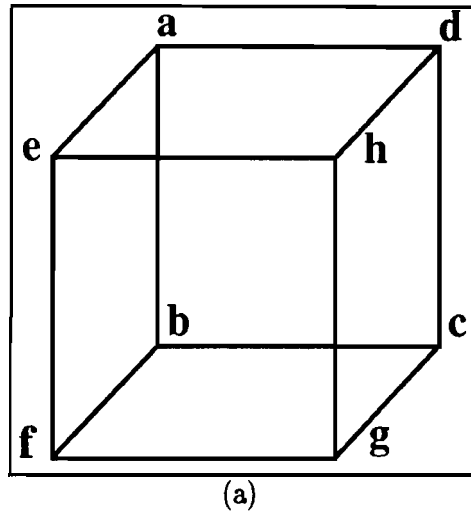
However, the pattern classification ability of the existing machines is still far away from the human classification ability for the following reasons:

1. Humans perceive everything as a pattern, whereas for machines everything is data. Even in a routine data consisting of integer numbers (like telephone numbers, bank account numbers, car numbers), humans tend to perceive a pattern [Yeg98].
2. Functionally also humans and machines differ in the sense that humans understand patterns, whereas machines can be said to recognize patterns in data. In other words, humans can get the whole object in the data even though there is no clear identification of subpatterns in the data. For example, consider the name of a

person written in a handwritten cursive script. Eventhough the individual patterns for each letter may not be evident, the name is understood due to the visual hints provided in the written script [Yeg98].

3. Human beings are capable of making mental patterns in their biological neural network from an input data given in the form of numbers, text, picture, sounds, etc., using their sensory mechanisms of vision, sound, touch, smell and taste. These mental patterns are formed even when the data is noisy or deformed due to variations such as translation, rotation and scaling. The patterns are also formed from a temporal sequence of data as in the case of speech and pictures. Another major characteristic of a human being is the ability to learn continuously from examples. These aspects are not at all well understood in order to implement them efficiently in an algorithmic fashion in a machine [Yeg98].
4. Human beings have the capability to gather information from both data and rule, simultaneously. Still most of the machine classification techniques are based on either data or rules.
5. For classification, human beings generalize most of the common objects and **memo-**rise uncommon objects. Moreover, in human classification there is a smooth transition from memorisation to generalization and vice versa. These abilities are totally absent in the techniques adopted by machines.
6. In pattern classification on machines, we face the problem of inductive bias or the a *priori* bias of the designer. The problem of inductive bias is that the resulting representation and search strategies provide us a medium for encoding an already interpreted world. They do not offer us any mechanism for questioning our interpretation, generating new viewpoints or changing perspectives when they are unproductive [LS95].
7. In pattern classification several uncertainties exist, We still do not know how to model these uncertainties. For instance, Fig. 2.8(a) can be classified to any one of the two classes as shown in Fig. 2.8(b) and Fig. 2.8(c). The classes are not fuzzy. It appears that rough uncertainty is also not present. Moreover, there are no probabilistic and resolution uncertainties involved. Still, the exact classification result may not be known!

The above issues will continue to motivate researchers to explore methods to match the performance of human classification.



**Fig. 2.8:** The psychological uncertainty which we do not know how to model. (a) A cube, and (b) and (c) are its two different interpretations. Apparently no fuzzy, rough and probabilistic uncertainties are involved in the interpretation. Still it is uncertain which interpretation we should follow.

## 2.3 Modular Classifiers

### 2.3.1 Background of Modular Classifiers

In the last section we have discussed the role of uncertainties in pattern classification. In many problems, it is difficult to deal with uncertainties in a monolithic classifier. Therefore, it is useful to divide the classification task among several small subclassifiers, and then combine their individual solutions to obtain the final classification result. In this section, we review various modular **approaches**, which are based on the divide and conquer technique.

Human brain consists of about  $10^{11}$  neurons and  $10^{15}$  connections. Due to its highly organized architecture, the brain manages to execute a myriad of functions and yet maintains a compact size. Execution of mental functions are allocated to different parts of the brain. Split brain patients in which the connection between the two hemispheres is completely severed can live an almost normal life, which shows that the hemispheres indeed function to a large extent independently. It has been found that in each hemisphere, part of the brain has a regular structure in layers, streams and many microscopic levels. Modules containing little more than a hundred cells, also known as minicolumns, have been proposed as the basic functional modular unit of the cerebral cortex.

A functional advantage of the anatomical separation of different areas of the brain might be minimization of mutual interference between simultaneous processing and execution of different tasks. For example, we have no problem in driving a car while listening to the radio. Studies with multiple tasks can easily be performed in parallel, while the simultaneous execution of similar tasks (e.g., presentation of two auditory or two visual messages) causes more interference. Some tasks are processed in distinct streams of modules and do not interfere with each other. Other tasks require simultaneous access to a single module, and are, therefore, much more likely to interfere mutually. This evidence suggests that modular approach is not merely advantageous, but essential.

In many complex pattern classification tasks (e.g., script recognition [CK95b], speech recognition [SY96], etc.), where the number of classes is large and the similarity amongst the classes is high, a monolithic classifier, also known as all-class-one-classifier, either **may** not converge or may take large amount of time to converge during training. But the all-class-one-classifier needs lesser storage and leads to better generalization if they converge. It is also possible to develop a classifier based on the concept of one-class-one-classifier [Kun93] architecture, where a separate classifier is trained for each class.

This kind of local approach offers the following advantages: (a) fast learning, (b) requires a few training examples, and hence, it can operate in real time. This approach requires a large number of subclassifiers, and also the discriminatory capability of the one-class-one-classifier is poor [SY96]. Therefore we need something in between these two extremes, where advantages of both all-class-one-classifier and one-class-one-classifier can be enjoyed. Motivated by the biological evidence, pattern recognition in real life problems can be approached using classifiers that are in between one-class-one-classifier and all-class-one-classifier. In this approach, modularity is viewed as a manifestation of the principle of *divide and conquer*. In [Hay94], [JJ93], modular classifier is defined as follows: "A classifier is called a modular classifier if the computation performed by the classifier can be decomposed into two or more modules that operate on distinct inputs without communicating with each other. The outputs of the modules are mediated by an integrating unit that is not permitted to feed information back to the modules". Thus the principle of modular classifiers can be thought as *Some Class One Classifier*. Modular classifiers have the advantages of both all-class-one-classifier and one-class-one-classifier approaches, like quick convergence, parallel training, better generalization, etc. The use of modular classifier systems was discussed as far back as the mid 1980's by Barto and Hinton. Jacob in [JJ93] presented a taxonomy for a class of modular hierarchical connectionist models. Modular approaches find applications in handwritten character recognition, texture recognition and speech recognition [SY98e].

### 2.3.2 Advantages of Modular Classifiers

The main advantages of the modular approach are as follows ??:

1. The modules can be constructed using different techniques. For example, some modules can be based on ANN, where the input-output relationship can be explained only through input-output patterns. Some other modules can be based on fuzzy rule base systems, where expert's knowledge is easy to obtain in form of rules.
2. Generally modularity results in an architecture of lesser complexity, and hence, is easier and faster to train. The generalization capability may also be enhanced due to the reduction in complexity [CK95a].
3. Training of the modules can be done in parallel. Therefore, it takes less time to train a modular classifier.
4. Different feature sets can be used to train different modules. This flexibility allows

us to use the appropriate set of features for each module so that within-class distance decreases and between-class distance increases. Therefore, the training time decreases and generalization ability increases.

5. Each module can be trained on different input data set. It decreases the training time and increases the generalization capability.
6. If the modular classifier is built carefully, then it can capture discontinuous **input-output** functions [JJ93]. In contrast, the monolithic counterpart does not have this capability.
7. In a modular approach existing modules can be retrained easily. If a new pattern is added, only the related modules need to be retrained. In the modular approach, new modules can also be appended easily.

### 2.3.3 Issues in Modular Approach

In order to construct a modular classifier systematically, we need to consider mainly three points. Firstly, the given classification task has to be decomposed into subtasks. Secondly, an appropriate classification task has to be assigned for each module. Finally, intermodule communication has to be evolved. The following are some of the issues in constructing a modular classifier:

1. **Depending on the decomposition criterion:** The classes can be grouped based on the closeness of the class prototypes in the feature space. A clustering algorithm can be employed for this purpose. Another way is to employ domain-specific knowledge to partition the classification task.
2. **Choice of classifiers for each module:** Various types of classifiers, e.g., feedforward neural networks with backpropagation learning algorithm, radial-basis function neural networks, probabilistic neural networks, fuzzy rule base system, etc., can be used in each module.
3. **Interpretation of the outputs of each module:** The output of each module can be interpreted as an *a posteriori* probability, belief or fuzzy membership values.
4. **Choice of the preprocessor or postprocessor:** It depends on which principle we are adopting to make a module active. In other words, the task may be so distributed that only one module is active or all the modules are active. The first method needs a preprocessor to decide which module should be active, and the

## Categorization of Modular Classifiers

---

- (a) Depending on Task
  - Functional
  - Categorical
- (b) Depending on Decomposition Criterion
  - Problem decomposition
  - Class decomposition
- (c) Depending on Topology
  - Preprocessor-based
  - Postprocessor-based
  - Hierarchical
- (d) Depending on Fusion Criterion
  - Maximum output approach
  - Weighted output approach
  - Dempster-Shepherd approach
  - Fuzzy integral approach

**Fig. 2.9:** Types of modular classifiers.

later method needs a postprocessor so that the results of all the modules can be fused.

### 2.3.4 Types of Modular Classifiers

It is possible to group modular classifiers based on various criteria. They are as follows (Fig. 2.9):

1. **Depending on the task:** One way to decompose a classifier is to create modules that serve very different functions, not different versions of the same function. The top-down structure of a large software projects is an example, where each procedure has its own function. This is called *functional* modularisation [DY97]. Another way is to decompose the classifier such that the modules perform different versions of



the same job. It is called *categorical* modularisation. This can be thought of as a set of experts giving their individual opinions on the same subject.

## 2. Selection of grouping criterion:

- (a) **Problem decomposition:** The designer decomposes the modules based on his knowledge about the classification problem (Fig. 2.10). Sufficient prior knowledge is essential when this is carried out before the learning takes place [TMBC92]. Another variation is to perform the decomposition **auto-**atically when the learning takes place [JJNH91].
- (b) **Class decomposition:** The original classification problem is divided into several sets of subproblems according to the inherent relations among the training data [AMMR93] (Fig. 2.11). It can be done before learning or during learning [LI98]. If it is done before learning, domain specific knowledge is needed. Compared to the problem decomposition, the class decomposition approach needs more computation. But, the later one becomes attractive when there is no prior knowledge about the problem.

3 **Depending on the topology:** Architecturally, modular classifiers can also be subdivided as follows: 1) Preprocessor-based, 2) hierarchical-based and 3) postprocessor-based. In Fig. 2.12, these three variations along with a monolithic classifier are shown. The Fig. 2.12(a) depicts a monolithic classifier. Here only one module is present. In Fig. 2.12(b), the selector or preprocessor analyzes the input, and decides which module should be used to classify the input. As a result, only one module will be active for each input pattern. This type of topology needs the preprocessor to be highly accurate. Fig. 2.12(c) employs a set of modules arranged in a hierarchical fashion. Although the input goes to all the modules directly, the top one becomes active first. If it can classify the input, then the classification of the input is over. Otherwise, it triggers the module just below it. In this way, the control flows from the top to the bottom. It is as if a set of experts are present and we are asking them the same question sequentially, and the whole session is over as soon as someone is able to provide an answer. Note that in some cases the answer given by a particular expert may be wrong also, and there is no scope for the experts, who are hierarchically below him, to rectify the answer. Hence, the drawback of this kind of approach is that if the higher level modules fails to trigger

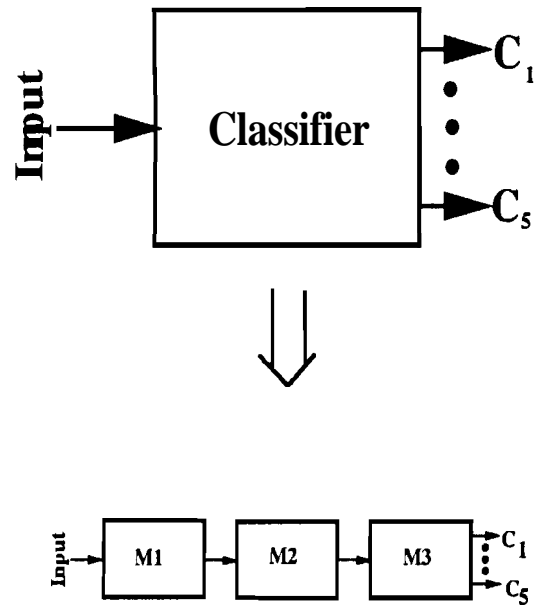


Fig. 2.10: Construction of modular classifiers based on the concept of problem decomposition.

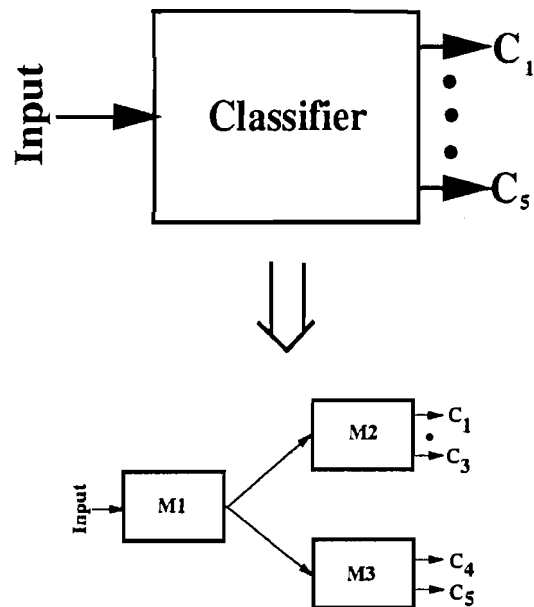
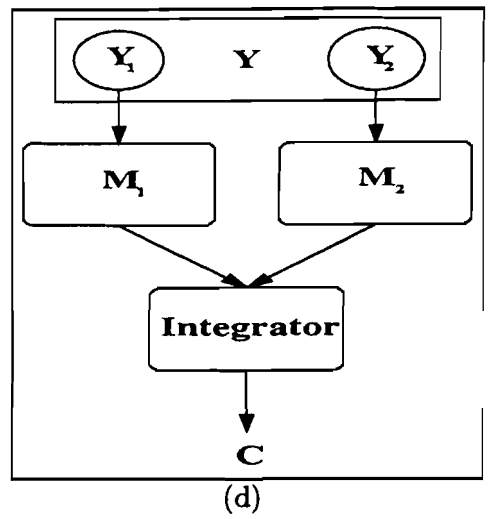
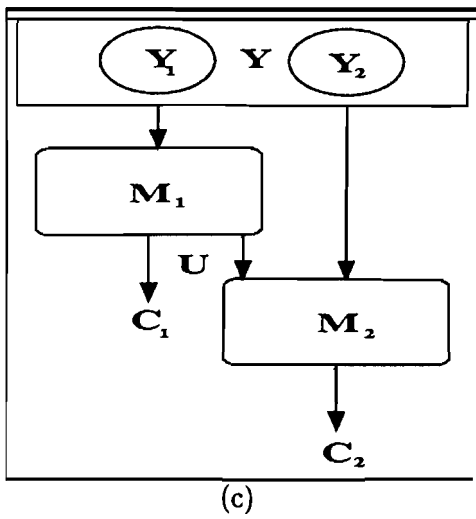
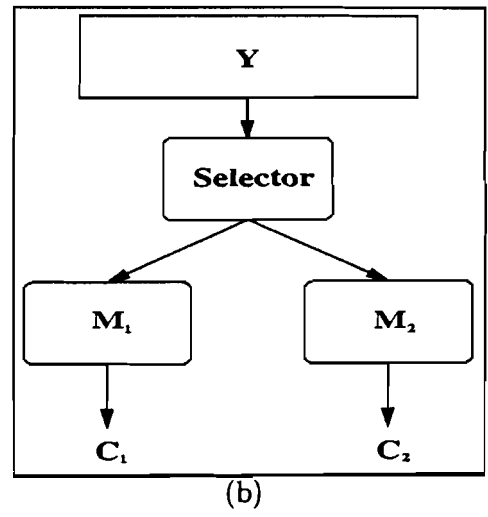
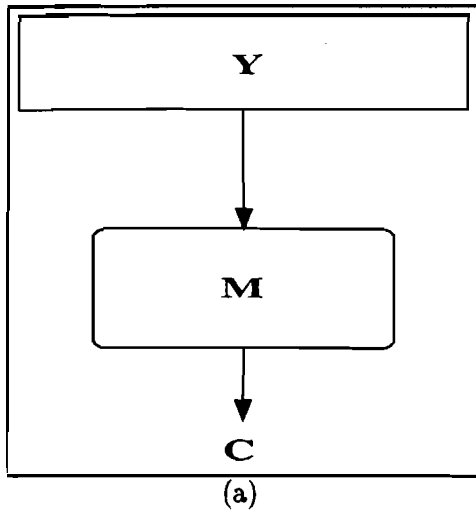


Fig. 2.11: Construction of modular classifiers based on the concept of class decomposition.

or mistakenly triggers any lower level module, then the whole classification problem becomes erroneous. Hence, the accuracy of this model largely depends on the classification accuracy of the higher level modules. Since it is difficult to train each module such that it fires correctly for all the test examples, the efficiency of the whole classifier is not usually high. In Fig. 2.12(d), a postprocessor or integrator combines the results of all modules. When some test input is used, all the modules become active in parallel, and the output result of all of them are fused by the integrator. Note that, in the hierarchical and postprocessor-based modular classifier, the feature set for each module can easily be made different from others. It is also possible to house both selectors and integrators in the same system [RRM<sup>+</sup>96].

**4 Depending on the fusion method:** For the task of categorization, the following fusion methods exist:

- (a) **Maximum output approach:** The class label of the input can be decided based on the winner-take-all policy. It means that the class label of the input pattern is assigned to  $j$ , where  $o_j = \max_{k=1,2,\dots,C} o_k$ , and  $o_k$  is the output corresponding to the  $k$ th output class. Although this is the simplest method, this kind of assignment may not be justified, as all the subclassifiers are independently trained on different sets of data.
- (b) **Gating or weighted output approach:** Compared to the above approach, a better approach is to declare the  $j$ th class winner, if the  $j$ th class corresponds to  $\max_{k=1,2,\dots,C} \{g_k o_k\}$ , where  $g_k$  is the importance associated with the class  $C_k$ . One possible choice for  $g_k$  is the *a posteriori* probability of the class  $C_k$ . The drawback of this method is that the probability constraint  $\sum_{k=1,2,\dots,C} g_k = 1$  cannot discriminate between lack of evidence and ignorance [KO96].
- (c) **Dempster-Shafer theoretic approach:** Dempster-Shafer's theory [Sha76] replaces the additivity requirement of probability measure theory with either a superadditivity or subadditivity requirements. Therefore, Dempster-Shafer's theory can distinguish between lack of evidence and negative evidence. In Dempster-Shafer's theory, each information source, i.e., module, generates a belief function over the power set of the hypotheses (i.e., output classes), which are then combined using Dempster-Shafer's rule [Sha76]. The calculation can have exponential complexity with large number of output classes.



**Fig. 2.12:** Four types of modular classifiers: (a) monolithic, (b) preprocessor-based, (c) hierarchical and (d) postprocessor-based.  $Y$ ,  $C$  and  $M$  denote the input feature vector, the output classes and the subclassifiers, respectively.

- (d) **Fuzzy integral theoretic approach:** In the fuzzy integral approach, the outputs of the modules are processed further so that the interactions among the outputs are also exploited for the final classification result. For an input pattern  $\mathbf{x}$ , each module (say  $s$ th) generates a partial evidence  $h_k(\{\xi_s\})$  to support the  $k$ th class, where  $\xi_s$  denotes the output of the  $s$ th module. The term  $g_k$  may be replaced by a more specific term  $g_k(\{\xi_s\})$ , where  $g_k(\{\xi_s\})$  denotes the importance of  $\xi_s$  in characterizing the class  $C_k$ . With the help of  $g_k(\{\xi_s\})$ ,  $s = 1, 2, \dots, S$ , the fuzzy integral  $\mathcal{F}_k$  for the class  $C_k$  combines all the partial evidence, i.e.,  $h_k(\xi_s) \forall s = 1, 2, \dots, S$ , in a nonlinear fashion. The final class label corresponding to the input is  $j$ , if  $\mathcal{F}_j = \max_{k=1,2,\dots,M} \{\mathcal{F}_k\}$ .

Like Dempster-Shafer approach, fuzzy integral-based approach also can distinguish between equal evidence and ignorance. In the fuzzy integral, the frame of discernment contains the information sources related to a particular hypothesis (i.e., an output class) under consideration, whereas in the Dempster-Shafer theory, the frame of discernment contains all possible hypotheses (i.e., all possible output classes). Thus, the fuzzy integral approach has a means to assess the importance of all groups of information sources towards supporting a particular hypothesis as well as the degree to which each information source supports the hypothesis. In contrast, the Dempster-Shafer theory does not have this advantage [KGT<sup>+</sup>94]. In addition, the fuzzy integral is computationally more efficient than a strict Dempster-Shafer approach. In the Dempster-Shafer theory, each information source generates a belief function over the power set of the hypotheses, which are then combined using Dempster-Shafer's rule. The calculation can have exponential complexity with the number of hypotheses, i.e., with the number of output classes ( $C$ ). In the fuzzy integral, the measure needs to be calculated only for  $S$  subsets, where  $S$  is the number of modules involved with each hypothesis [KGT<sup>+</sup>94]. Fuzzy integral is beneficial because in many modular classifiers,  $C \gg S$ .

For the task of functional modularisation the following methods exist:

- (a) **Majority voting:** The simplest linear combination method is majority voting. That is, the output of the most number of modules will be the output of ensemble. If there is a tie, then the output of the module (among those in the tie) with the lowest error rate on a test set will be selected as the ensemble output. Another method is to keep the number of modules odd so that the

conflict cannot arise.

- (b) **Weighted averaging:** The weights are fixed in proportion to how each module performs on a test set. When the weights summed up to one, the weights can be viewed as *a priori* probability of the module to classify an input accurately. When this additivity constraint is relaxed, Dempster-Shafer approach and fuzzy integral-based approach become suitable tools to use. The advantages and disadvantages of Dempster-Shafer and fuzzy integral-based approaches are similar to that of categorical modularisation.

## 2.4 Opening Bid Problem in Contract Bridge as a Pattern Classification Problem

The opening bid problem is an exercise of high level perception. It involves classifying the pattern in a hand to a single output corresponding to the bid for the hand. A classifier does precisely this. Given a set of input hands and the corresponding opening bids, the classifiers try to capture the implicit relation between the two. Once the classifier has been trained to generalize, then it can respond meaningfully to a new hand. In the opening bid problem we can immediately spot the presence of probabilistic, fuzzy and rough uncertainties. The card pattern, what one gets after shuffling is purely *probabilistic*; on the other hand, whether the player will classify a particular input hand as an 1D or 1H is basically *fuzzy*. Here the classes corresponding to 1D and 1H are overlapping. The relation between the input pattern and the corresponding output bid is not unique even among the expert players (see Table 2.2). Some player may consider the hand pattern “97-5-AKQ8754-AK2” as 1D or some may consider it as 2C. It is because the playing strategy of the players for the remaining part of the play, vulnerability, etc., are different and are difficult to model. Hence the same input hand may belong to different classes, although the classes are not overlapping. This situation creates *rough* uncertainty. It is to be noted that in the card problem *resolution* uncertainty is absent. It may be present, if one is asked to classify the input hand based on the image of the input hand taken by a camera. It is because there may be some ambiguity in identifying, for example, a card as a Heart or a Club due to the poor resolution of the image. In this thesis we are not considering resolution uncertainty.

For the opening bid problem, we have chosen numerical representation of the object data. The raw input acts as a feature vector which is later refined. The structure present

**Table 2.2:** Variations in players' bids. Sets of 80 hands were bid by human players. It can be observed that for every set, the players differed on a significant number of hands, suggesting that more than one correct bid may exist.

S r	No. of    who bid	No. of hands for which	
		In numbers	In %
1	2	14	17.50
2	3	17	21.25
3	2	26	32.50
4	2	24	30.00
5	2	14	17.50
6	2	13	16.25
7	2	17	21.25
8	2	29	36.25

in the feature space can be interpreted in terms of direct classification and classification through clustering. For the opening bid problem, we need to have some classifier with fast learning and quick classification capability in the presence of fuzzy, rough and probabilistic uncertainties. ANNs seem to be a possible candidate for this purpose. Conventional ANNs are not suitable to deal with fuzzy and rough uncertainties efficiently. Hence, in this thesis an attempt is made to develop hybrid learning models, where the **neuro**-computing paradigm is integrated with fuzzy and rough paradigms. Following this line, modular classification approach is adopted to deal with the uncertainties in the opening bid problem. The performance of the resultant model is evaluated in terms of error rate measure.

The next chapter explores the possibility of capturing the implicit relationship in bidding a Bridge hand using an artificial neural network. We study issues like the role of uncertainties in the opening bid problem, input representation, possible architectures for the network.

## Chapter 3

# PRELIMINARY STUDIES ON BIDDING PROBLEM USING ARTIFICIAL NEURAL NETWORKS

### 3.1 Introduction

The objective of this chapter is to study different issues like the role of uncertainties, input representation, possible classifier architectures for the opening bid problem. Since it is not easy to find a recognizable structure in a hand pattern, we opted for numerical representations, rather than symbolic representations, to describe the classification process. Initially we attempted to construct a deterministic classification model for the opening bid system. It is because the deterministic model can be simple, and the representation problem for this type of model is the least. As a deterministic model, a crisp rule base approach is considered. To construct the rule base, we need to extract the rules of the following type from the expert bidders:

If the input hand pattern is A J 9 3, K 8 4, K 7 4 3, A 9, then the output bid is 1S

When an input hand is used for testing, the hand is matched against the *if* part of each rule. The class label is indicated by the output class of the rule that fires.

Extracting the rules from the experts are difficult because players normally use these rules only as a guideline, and often they make bids for which they cannot articulate their reasoning in terms of the given rules. For instance, for a hand containing 4 Spades and 4 Diamonds, a rule may suggest opening 1S, or possibly 1D. But for the two hands given below, which are only slightly different, a player may choose different bids as

A J 9 3, K 8 4, K 7 4 3, A 9 – bid: 1S

A J 9 3, K 8, K 7 4 3, A 9 4 – bid: 1C



This change comes because the player uses subtle reasoning process, and he is also concerned about his next bid. There are other patterns too in the hand for such a reasoning. For example, “K84” is a "support" for a possible bid of 2H by partner, while “K8” is not. “A94” in Clubs, on the other hand, is an *openable* suit if the hand has no five carder. One could possibly list all such possibilities as rules, but the number of rules will be too many (it is approximately the number of possible hands, i.e.,  $6.35 \times 10^9$ ). Constructing, such a large rule base is an impossible task. If the size of the rule base is decreased to a moderate one, then the rule base cannot cover many hand patterns and situations. Consequently, when a hand pattern outside the rule base is encountered, the rule base approach fails to indicate the output. In other words, the system is not generalizable; it works just like a look-up table.

The above drawback of the deterministic model motivated us to exploit uncertainties such that the classification system becomes robust and generalizable. The rule base works as long as the input comes only from the points of the input space at which the input-output relationship is defined. Let us call these points *reference points*. The model can be made more powerful, if we can assign some certainty factors (to belong to the output classes) on the neighborhood points of the reference points. Thus the input-output relation becomes defined for all the points in the input space. In this work, the neighborhood points are viewed as *similar* to the reference point, and hence, the certainty factors associated with the neighborhood points are expected to be close to that of the reference point. Thus fuzzy uncertainty is introduced in terms of *similarity* to make the problem generalizable.

In order to incorporate the fuzzy uncertainty in the opening bid problem, we have used artificial neural networks (ANNs). The reasons behind choosing ANNs over other possible classifiers are various benefits like incremental learning, robustness, universal approximation capability. From section 2.2.4, we know that there are the following two possible schemes to use ANNs as classifiers: Direct classification and classification through clustering. We first experiment ANNs with direct classification techniques. For this study, we explore the possibility of training a multilayer feedforward neural network (FFNN) with backpropagation (BP) training algorithm [RHW86]. It tries to capture the implicit reasoning involved from several examples of input (pattern)-output (bid) pairs of data. In order to perform this study the following issues need to be considered:

1. Collection of data.
2. Representation of the hand.

3. Interpretation of the output bid.
4. Architecture of the network.
5. Training of the network.

The first issue deals with the collection of input data that have to be used for training as well as for testing. To collect variety of inputs, the collection of data should be random. Moreover, care should be taken to partition the data for training and testing so that both have similar probability distribution. Second issue is how to represent the input data on a machine. Although we have decided numerical representation for the input hands, there exist several possible numerical representations. The exact choice will be dictated by the classification performance with the representation. The third issue is the interpretation of the output results. It is quite possible that during testing the network produces a bid which is different from a player's bid. But, then the player should also decide whether the network bid is reasonable for the given hand. The fourth and fifth issues are regarding the type of the network and the training methodology that have to be used. These issues are discussed in this chapter.

Initially, we attempted to train a monolithic ANN for all the classes. But the network did not converge. One possible reason may be that the network was unable to handle, resolve and exploit the associated uncertainties globally. This problem may be reduced if the classification task is partitioned. Partitioning should be such that each subproblem is solved in a module by exploiting the local uncertainties and the results of all the modules are combined by exploiting the global uncertainties. To verify it empirically, we break the monolithic classifier into several modules using some domain specific knowledge, and test the classification performance of each model. The experiments conducted in this chapter advocate the use of a modular network instead of a monolithic network.

The organization of the chapter is as follows: Section 3.2 discusses the representation of the problem. Section 3.3 demonstrates the performance of the feedforward neural networks with different architectures.

## **3.2 Representation of Opening Bid Problem on Machines**

### **3.2.1 Data Generation and Collection**

The hands used for training the network were generated by a program which simulates shuffling of the cards. The distribution of the hand patterns generated by the program

matches the distribution given in Table 3.1. A representative set of 19 hands are given in Table 3.2. Note that the hands which contain suits of maximum length 5 constitute about 80% of all the hands. On the other hand, for the bids of 2H and 2S, the input may require the following features: A six card suit, with no singleton or void in the hand, about 8 to 10 high card points, with most of the high cards in the bid suit. To successfully learn these bids, it is necessary to have a large number of these samples in the training set. It would mean a correspondingly large training set, and hence, a large training period. We have used a generating program to produce the hands according to a given set of constraints, for example, the length of the Heart suit should be at least 6 and the number of points should be at least 6. In this way, we can produce more hands for which we want the system to learn the patterns. The expert's bids were collected from the experts in IIT open Bridge Tournament, 1994.

### 3.2.2 Representation of Input Patterns

The inputs can be represented as a fifty-two dimensional raw data as shown in Fig. 3.1(a). Each component of this vector is either 0 or 1. The value 0 and 1 indicate the **presence/absence** of the card in the hand. Since each player has thirteen cards, the number of 1's in the raw data is equal to thirteen. For example, the first hand in Table 3.2 and the corresponding input pattern vector are given by

**Hand:** K753-KJ8-K87-K76

**Input Pattern:** 01000001010100101001000000010000110000000100000110000

The input can also be represented in the form of feature patterns as shown in Fig. 3.1(b). These patterns are based on the evaluation of the strength of the hands by a bidding system. In this representation there are 16 components of the raw data vector, which can take values between -1 and +4. Thirteen components are used to represent the cards, while three are used as markers (-1) between suits. In this representation an attempt was made to feed some feature information in the form of relative weights given to various cards. For example, on the cards Ace, King, Queen, Jack, 10, 9, 8, ..., 2 we have assigned the following weights: +4, +3, +2, +1, +0.9, +0.8, +0.7, +0.6, ..., +0.1. These weights were close to the points given to the cards in most bidding systems.

Our initial experiments showed that the first representation is preferable. During training we found that the network converged with the first representation, whereas the

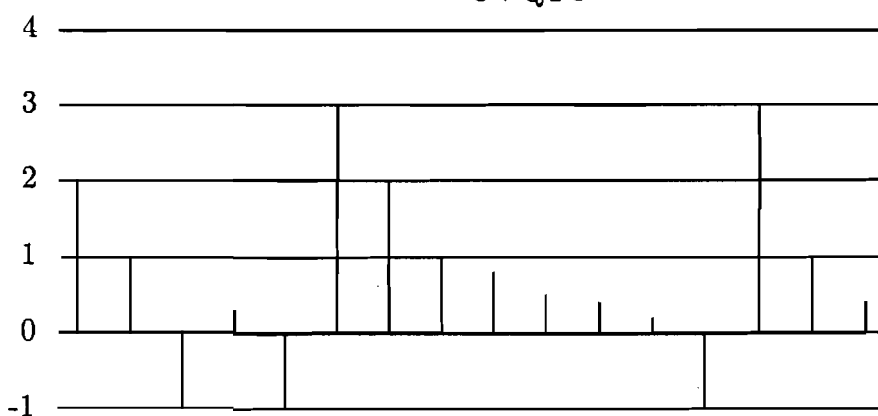
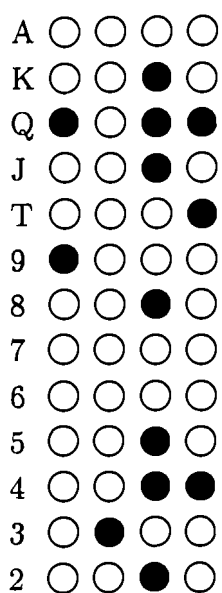
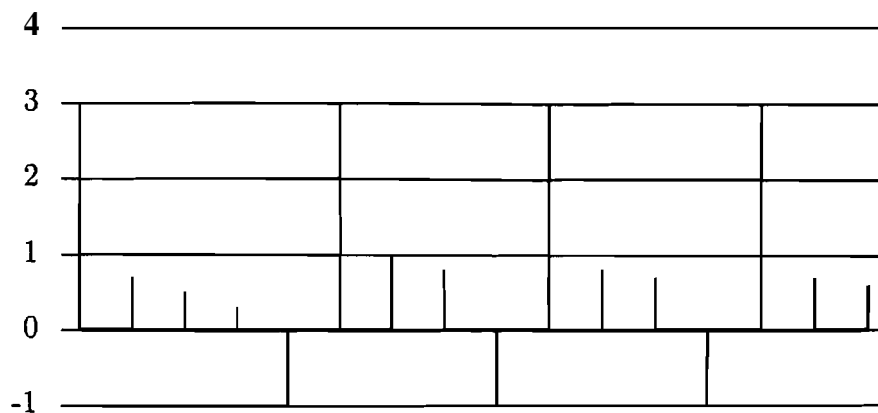
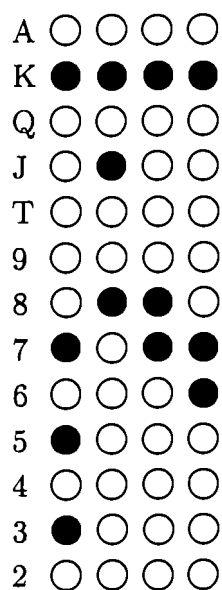
**Table 3.1:** Distribution of hand patterns. Numbers under total sum up the values for all possible ways of choosing suits for the given pattern or shape.

Values listed under *specific* are from a med suits having specified length.

No.	Pattern	Specific (in %)	Total (in %)	No.	Pattern	Specific (in %)	Total (in %)
1	4-4-3-2						
		1.796	21.5512	24	8-2-2-1	0.016	0.1924
2	4-3-3-3	2.634	10.5361	25	8-3-1-1	0.010	0.1176
3	4-4-4-1	0.748	2.9932	26	8-3-2-0	0.005	0.1085
				27	8-4-1-0	0.002	0.0452
4	5-3-3-2	1.293	15.5165	28	8-5-0-0	0.0003	0.0031
5	5-4-3-1	0.539	12.9307				
6	5-4-2-2	0.882	10.5797	29	9-2-1-1	0.001	0.0178
7	5-5-2-1	0.264	3.1739	30	9-3-1-0	0.0004	0.0100
8	5-4-4-0	0.104	1.2433	31	9-2-2-0	0.0007	0.0082
9	5-5-3-0	0.075	0.8952	32	9-4-0-0	0.00008	0.0010
10	6-3-2-2	0.470	5.6429	33	10-2-1-0	0.00004	0.0011
11	6-4-2-1	0.196	4.7021	34	10-1-1-1	0.0004	0.0001
12	6-3-3-1	3.448	0.287	35	10-3-0-0	0.00001	0.00015
13	6-4-3-0	0.055	1.3262				
14	6-5-1-1	0.059	0.7053	36	11-1-1-0	0.000002	0.00002
15	6-5-2-0	0.027	0.6511	37	11-2-0-0	0.000001	0.00001
16	6-6-1-0	0.006	0.0723				
				38	12-1-0-0	0.0000003	0.000003
17	7-3-2-1	0.078	1.8808				
18	7-2-2-2	0.128	0.5129	39	13-0-0-0	0.000000002	0.0000000009
19	7-4-1-1	0.033	0.3918				
20	7-4-2-0	0.015	0.3617				
21	7-3-3-0	0.022	0.2652				
22	7-5-1-0	0.005	0.1065				
23	7-6-0-0	0.0005	0.0056				

**Table 3.2:** Sample hands generated by the shuffling problem. Bids are made by the authors, to illustrate the preparation of the training set. In this training set some less frequent hands are present which are generated specially to ease learning. "T" implies the card number 10.

No.	Hands (S)-(H)-(D)-(C)	points	Desired bid (by authors)
1	K753-KJ8-K87-K76	13	1C
2	Q9-3-KQJ8542-QT4	10	3D
3	863-KQJT954-K9-5	9	3H
4	T853-KT83-A9-AQ7	13	1S
5	J62-AJ5-K652-942	9	P
6	7-AQT976-874-K85	9	2H
7	Q87-KQJT63-92-95	8	2H
8	A98-K2-AQJ62-AT6	18	1N
9	QT75-A73-J3-AKJ8	15	1C
10	T75-K52-AK86-QJ7	13	1D
11	Q94-QJ832-A95-AK	16	1H
12	AKT7642-Q53-8-T5	9	3S
13	J986-72-AQ543-J8	8	P
14	A7-K94-KT65-AK74	17	1N
15	A92-4-AK82-AJ832	16	1C
16	QT-76-AQJ752-AJ3	14	1D
17	A-KQJ9873-J98-T6	11	1H
18	AK842-AKT93-6-82	14	1S
19	AQJT-QT4-AJ-QT92	16	1N



(a) Representation 1

(b) Representation 2

**Fig. 3.1:** Illustration of input layer patterns for two hands. In representation 1 the input is in raw form, while in representation 2 some features (high card points) have been extracted. The networks perform better with the raw information.

feature based representation failed to converge in some cases. This is interesting because the information in the second case is in an interpreted or an abstracted form. It appears that abstraction from raw data, if not done properly, may not be useful for generalising the network. Experiments described in this chapter therefore use the first representation.

### 3.3 Studies on Network Architecture and Training

This section describes the development of the network architecture by trial and error procedure. The training algorithm was the BP algorithm. The input layer has 52 nodes, one for each card. Each node has a value 1, if the card is present, or 0, if the card is not present in the given hand. In the following, we describe our trial experiments for evolving a suitable **ANN** architecture for the bidding problem.

**Experiment with monolithic networks:** An approach using 13 output nodes to capture all the bids from Pass to 3S was explored. Three nodes were assigned to the three levels of bids, five were assigned to the suits and one node was kept for "Pass" bid. Thus, for each input, except for a "Pass" hand, two output nodes are expected to be active, one for the level of the bid, and the other for the suit. The training set consists of 1000 hands. Different network architectures were examined. Two of them are (a) 30 and 20 nodes in the first and second hidden layers, and (b) 30 and 6 nodes in the first and second hidden layers. It was found that the network did not converge. This was probably because there were a large number of bids (some 2 level bids and all 3 level bids) for which very few training patterns were available. For the subsequent experiments, we decided to use a simpler format for output nodes with one output node for each bid. Therefore, we used fifteen output nodes to study the network behaviour. This time also the network failed to converge.

**Experiment with 1-Level networks:** To resolve the convergence problem, we reduced the number of output nodes to seven, by restricting the bids to 1 level only, including "Pass" (P) and the "Unknown" (U) category bids. When the input is not from the first level bids, then the class label "Unknown" is assigned as the desired class label. The resulting network consists of 52 input nodes, 7 output nodes and one hidden layer. The number of nodes in the hidden layer was varied to study its effect on the performance. A training set of size 1000 is chosen. Table 3.3 gives the total sum of squared error for 40, 45, 50, 55 and 60 hidden nodes. We observed that the network converged for various number of hidden nodes. The network with 50 hidden nodes gave the highest accuracy of

**Table 3.3:** Mean square error of the 1-Level network after training. Here 52 input nodes and 7 output nodes are used. The number of iterations is 5000, and the number of hands is 1000.

No. of hidden nodes	Mean square error
40	26.14
45	28.54
50	20.81
55	24.11
60	23.81

69% correct bids on the test set, when compared with the bids made by an expert player. The test data consisted of 500 randomly generated hands. Some test results are given in Table 3.4 and 3.5. It should be noted that while evaluating the performance, if the output of the network was also acceptable by the expert player as a possible bid, then it was taken as a correct output.

**Experiment with 2-NT networks:** We consider a network to include 2 level bids. Here, the number of output nodes is 12, one each for P, 1C, 1D, 1H, 1S, 1N, 2C, 2D, 2H, 2S, 2N and U. Initially we attempted to train the network with a training set conforming to the theoretical distribution of hand patterns. But the network could not be trained. The network was unable to learn the patterns for 2C, 2D, 2H, 2S, 2N bids since they are very rare. Obtaining suitable samples of such hands require large amount of training set. Instead we decided to selectively insert the patterns, which are rare, into the training set. Nearly 600 hands from the 2 level bids were added along with nearly 1000 hands from the 1 level bids. As a result we obtained 1600 hands to train the network. This network was trained using five different architectures having 40, 45, 50, 55, and 60 hidden nodes. The mean square error of the network for this training set is shown in Table 3.6. The network with 50 hidden nodes gave the best performance. Results produced by the network are given in Table 3.5 and 3.7. The network has bid correctly for about 72% of the test **hands**, which were not part of the training set. Initially we planned to give only positive samples of hands for the bids which we wanted the system to make. But we found that for the system to perform well, we also had to give a large number of hands for which we did not want the system to make a bid. Hence, we introduced all those hands under the bid



**Table 3.4:** Bids made by the 1-Level network. Strong imbalanced hands are labelled "Unknown" for training purpose. However sometimes (e.g., the sample no. 2) the system did better by opening 1C. Also, in the sample no. 3, the network's bid seems to be better! The discrepancy in the last example is also typical of human players.

No.	Hands (S)-(H)-(D)-(C)	Points	Expert's bids	1-Level network
1	KJ6-Q62-K94-A864	13	1C	1C
2	-4-AT942-AKJ8974	12	U	1C
3	J8-KQT643-KQ92-T	11	P	1H
4	K64-AQ874-AT953	13	1C	1C
5	AKQT9-J765432-9-	10	U	P
6	AT63-KQJT6-AT73	14	1D	1D
7	AQT87-K4-AKT4-Q7	18	1S	1S
8	AKJ9542-QJ852-6	11	U	U
9	62-AT3-J942-AKT7	12	1C	1C
10	Q653-AKJ8-AJ-974	15	1H	1S

"Unknown".

Experiment with modular networks: In the bidding problem we have observed that a) large networks are difficult to train, b) a 1-Level network performs well, and c) a 2-NT network does not learn well because of lack of data for 2-Level bids. But, if specific data are added for the 2-Level bids, then it may perform well. From these results, it can be clearly observed that smaller networks are easier to train, and consequently they also perform better. Looking at the task environment, one can see that all the bids made at higher levels are specialized. In addition, they deal with hands that are less frequent. Hands with four card and five card suits are most common (80%) and the bidding systems are designed to use the cheaper (low level) bids for these hands. To design a complete ANN system would require sufficient training samples. It appears reasonable to consider the high level specialized bids as exceptions, and train different networks to deal with them. Thus one would have a modular structure of the network, each module catering to a specialized situation. Following this line, we employed one module for each level of output bids. Since output bids upto third level are present, we used total three modules. The first module is for the first level bids. It can classify Pass, 1C, 1D, 1H, 1S and 1N. Similarly, the second module is for 2C, 2S, 2H, 2D and 2N. The third module is supposed to classify 3C, 3D, 3H and 3S. Note that none of the module has the output class "Unknown". For each module we used FFNN with one hidden layer. All the modules have fifty hidden nodes. Let the training sets for the first, second and third modules be called "TrainingSet1", "TrainingSet2" and "TrainingSet3", respectively. The size of "TrainingSet1", "TrainingSet2" and "TrainingSet3" are 1200, 700 and 400, respectively. These training sets will be used again in the subsequent chapters for the experiments. For all the modules the convergence was achieved during training. Three test sets "TestSet1", "TestSet2" and "TestSet3" are formed to test the performance of the first, second and third modules, respectively. These test sets will be used again in the subsequent chapters to compare the performance of other networks. When the test data sets are presented to the first, second and third modules, the classification results are shown in Table 3.8, 3.9, and 3.10, respectively. The overall classification result of the third module is quite high compared to the other modules. It is because if the hand is strong, then Bridge players have less problem in giving the higher level bids.

**Table 3.5:** The bids made by the 1-Level and 2-NT networks for some hands are shown. Bids made by two experts are also included. Bids marked “\*\*” are incorrect. " T implies the card number 10.

No.	Hands (S)-(H)-(D)-(C)	Points	Expert's bids	1-Level network	2-NT network
1	AT5-J983-K5-AJT5	13	1C, 1H	1C	1H
2	AQ6-A752-AT2-KQ3	19	1C, 1H	U	1C
3	A95-AQ9-543-AQ93	16	1C, 1N	1N	1C
4	K85-K94-KQJ94-96	12	P, 1D	1C**	1D
5	T-AK6-J8642-QJ63	11	P, 1D	P	1D
6	KJ3-T-QT9643-AQ9	12	P, 1D	1D	1D
7	AQ84-AJ984-J92-9	12	P, 1H	1S	P
8	AQ8-AQJ6-63-AJ94	18	1H, 1N	1C	1C
9	974-A732-AK6-QJ6	14	1C, 1H	1C	1H
10	8753-4-AT5-AKT98	11	P, 1C	P	P
11	98-AK6-K854-AK43	17	1C, 1N	1C	1C
12	QJ72-K743-K9-A54	13	1S, 1H	1S	1S
13	Q72-A98-Q64-KJ92	12	P, 1C	P	P
14	JT4-J8-AKQ6-AQ64	17	1C, 1N	1N	1C
15	AKQ97-52-T3-9853	9	P, 2S	U	2S
16	K843-A64-A52-AT3	12	P, 1S	1S	1S
17	K832-AKJ3-QT5-K6	16	1N, 1H	1S	1H
18	-Q98762-KQ753-Q3	9	P, 1H	P	P
19	8-QJ5-A74-KJ9875	11	P, 1C	P	P
20	K8-AT94-JT7-AT65	12	P, 1H	1H	P

**Table 3.5:** (Continuation)

No.	Hands (S)-(H)-(D)-(C)	Points	Expert's bids	1-Level network	2-NT network
21	97-5-AKQ8754-AK2	16	2C, 1D	1D	1D
22	96542-AQJ2-AKQ7	16	1S, 1D	1S	1C
23	AJ75-4KJT6-AKT8	16	1D, 1C	1C	1D
24	A6-QJ86-J94-KQJ9	14	1H, 1C	1C	1C
25	T8-AT5-K976-A874	11	1N, P	1C	1C
26	KQ-QJ7642-QJ92-T	11	P, 1H	1H	1H
27	K87-J8-AKQ8-AKQ2	22	1C, 2N	U	2N
28	KT84-KQT7-K63-T2	11	P, 1S	P	P
29	AJ-Q64-J42-AKJ85	15	1N, 1C	1C	1C
30	A632-3-96-KQJ863	10	P, 1C	1C	P
31	965-AT8762-J7-A9	9	1H, P	P	P
32	Q7-A74-A2-KQJ972	16	1D, 1C	1C	1C
33	3-T6-KT74AQJT74	10	P, 1C	P	P
34	AKJ3-982-J85-KQ5	14	1S, 1C	1S	1C
35	92-KQ6-A93-AQJ73	16	1N, 1C	1C	1C
36	94-KQ3-AQT2-J632	12	P, 1D	1D	1D
37	-A874-AJT52-AQT5	15	1C, 1D	1D	1D
38	2-JT86-AQT9864-Q	9	P, 3D	U	P
39	K95-94-A852-AT83	11	P, 1D	P	P
40	Q-A743-AJ5-A8732	15	1H, 1C	1C	1H

**Table 3.5: (Continuation)**

No.	Hands (S)-(H)-(D)-(C)	Points	Expert's bids	1-Level network	2-NT network
41	-A642-AKT83-J943	12	1H, 1D	1H	P
42	AJ76-AQ73-75-J87	12	1S, 1H	1S	1S
43	AQJ73-KT85-42-72	10	1S, P	P	1S
44	KQ98754-83-Q7-A2	11	1S, 2S	1S	2S
45	QJ6-Q-KT864-A854	12	1D, P	1D	1D
46	743-Q65-AKT7-KQ9	14	1C, 1D	1N	P
47	J4-AJ654-J53-KJ9	11	P, 1H	P	P
48	K8653-3-7-AKJ753	11	3C, 1C	1C	1S
49	AT5-AK-Q9532-KQ3	18	1N, 1D	U	1N
50	A3-AT2-KT64-AKQJ	21	2C, 1C	U	1N**
51	75-A3-AQT632-AK5	17	1N, 1D	1D	1D
52	K53-QJ3-QJ8753-3	9	3D, P	U	2D
53	Q-KQ65-AKQ74-Q43	18	1N, 1D	1H	1D
54	A7-K8643-K863-A8	14	1S, 1H	1H	1H
55	K-AJ8-AK965-QT87	17	1N, 1D	U	1N
56	A943-Q432-Q52-A2	12	1C, P	P	1S
57	K87-AT9-J87-A972	12	1C, P	P	1C
58	JT86-AK43-76-AQ7	14	1S, 1H	1S	1S
59	982-Q652-AKQT-K5	14	1H, 1D	1D	1H
60	KQJT85-A96-743-8	10	3S, 2S	P	2S
61	AT982-QT2-A3-J65	11	1S, P	1S	1S
62	AJ32-AQ93-5-AQJ2	18	2C, 1C	U	1N**
63	76-AKQJ653-7-KQ9	15	1H, 4H	1H	1H
64	5-AJT8753-K4-K93	11	3H, 1H	1H	1H

**Table 3.6:** Mean square error of the 2-NT network after training. 52 input nodes and 12 output nodes are used. The number of iterations and the number of hands are 5000 and 1600, respectively.

No. of hidden nodes	Mean square error
40	23.15
45	22.68
50	15.21
55	15.17
60	17.19

**Table 3.7:** Bids made by the 2-NT network. Many experts would open the sample no. 4 with 2H, because the key feature - long solid suit is present. In the sample no. 8 the system has in fact done better by opening 1D. In the sample no. 9 it possibly had to choose between "Unknown" and 1C, since it does not know the 3N bid, which is very specialized.

No.	Hands (S)-(H)-(D)-(C)	Points	Expert's bids	2-NT network
1	KJT4-QT9762-A-T3	10	P	1H
2	A8-QT2-KQJ97532-	12	1D	1D
3	7-QJ98-QJ8752-T3	6	P	P
4	Q-KQJ852-73-K983	11	P	2H
5	K94-A5-AJT743-74	12	1D	1D
6	AK8742-J4-K852-9	11	1S	1S
7	AT9872-5-K54-643	7	2S	2S
8	A-K82-QJ9642-Q97	12	P	1D
9	J6-JT-8-AKQ98654	11	3N	1C
10	84-KJT9542-Q8-A3	10	2H	2H

### 3.4 Summary

The aim of the work reported in this chapter is to explore the possibility of capturing the reasoning process used in bidding an opening bid in Bridge game using an ANN. The network captures the implicit mapping in bidding a Bridge hand adopting a standard convention (bidding system) which acts as a guide or a weak constraint on the mapping function. We used an FFNN with BP algorithm as a classifier, whose input is a hand and the output is the corresponding opening bid. The input is represented as a series of 52 **one/zero** where presence or absence of a card is denoted by 1 or 0. While experimenting, it turned out that the training of the whole network is time consuming. Moreover, in many cases the network do not converge at all. Possible reason may be that the network is not able to tackle the uncertainties. We adopted a modular structure to deal with the uncertainties. Three modules are used to deal with the first three levels of opening bids.

The studies reported in this chapter demonstrate that a neural network can be trained to capture the implicit reasoning used for bidding a hand in the Bridge game. The present study clearly brings out several interesting research issues. The first issue is the representation of the input data. In situations like card games, representation in raw **form** appears preferable, as any feature representation is likely to be subjective and may result in loss of information. In contrast, in problems dealing with image and speech data, it is essential to represent the data in a manner that reflects the visual and auditory sensory processing, respectively. Errors in the feature representations are usually responsible for poor generalization performance in the pattern recognition tasks involving image and speech. The second major issue is training a network with patterns occurring with widely different probabilities. This is a difficult issue in many practical problems. For example, in optical character recognition different characters occur with widely different probabilities. Similarly, in speech recognition different speech sounds occur with different probabilities.

This chapter demonstrates that modular networks can be used for the opening bid problem. The classification efficiency may further be improved if the feature vectors corresponding to each module are modified depending on the output classes present in the module. The discussion in chapter 4 is along this line. Chapter 5 and 6 will concentrate on how to design each module. In chapter 7 we will combine the results of all these modules.

**Table 3.8:** Classification performance of FFNNs with BP algorithm for first level bids.

Pass	1C	1D	1S	1H	1N	Overall
85.12%	66.83%	63.13%	71.82%	77.16%	64.52%	71.43%

**Table 3.9:** Classification performance of FFNNs with BP algorithm for second level bids.

2C	2D	2S	2H	2N	Overall
61.04%	75.71%	77.56%	72.33%	74.34%	72.19%

**Table 3.10:** Classification performance of FFNNs with BP algorithm for third level bids.

3C	3D	3S	3H	Overall
75.17%	77.28%	83.54%	84.97%	80.24%



## Chapter 4

# IMPORTANCE OF INPUT FEATURES IN CLASSIFICATION: ROUGH-FUZZY SET THEORETIC APPROACH

### 4.1 Introduction

In the last chapter a modular network architecture is advocated for complex pattern classification problems like opening bid problem. This chapter attempts to fine tune the input features specifically for each module so that the class discriminatory capability of the input patterns are enhanced. In order to accomplish it, the importance of each feature is quantified, and the input representation is biased accordingly. Therefore, this chapter deals with feature analysis. In actual Bridge game, players impose different weightages on each card. These weightages depend on the player, level of the game and many other factors, e.g., experience of the player, characteristics of the player, vulnerability, etc. This fact also justifies the quantisation of the importance of each feature for each module in the opening bid problem.

In the modular structure, each module exploits the uncertainties locally, and specialises to classify only a group of output classes. Since the class discriminatory property of all the input features are not same for different sets of output classes, the representation of each pattern for a particular module should be fine tuned based on the **output** classes present in the module. One way to accomplish it is to put different importance on each feature. The input representation for each module can be biased so that the feature with higher importance gets more weightage. Determination of the importance [SB97] [WAM97] is generally based on a criterion function and a search strategy. The search strategy chooses a set of importance among **many** possible sets of importance, while the criterion function decides whether a set of importance is superior to another set. The search techniques, that exist in literature, can be broadly classified into *filter* approach and *wrapper* approach [SB97] [WAM97]. The filter approach depends on a crite-

tion function which is classifier independent. In contrast, the wrapper approach uses the classifier accuracy to judge whether a particular set of importance is superior to another set. The wrapper approach can be used only with the classifiers of low computational cost like K-nearest neighbours (KNN) algorithm, decision tree, etc. In addition, the wrapper approach may cause overfitting as the learning algorithm is fitted by the change of input features [JK95]. We prefer the filter approach as this technique is generally applicable, and can be used with complex classifiers like feedforward neural networks with BP algorithm.

Most of the classifiers classify a test input based on the fact that the more similar the test input is to the set of training patterns, the higher is the possibility that this test input belongs to the same class. Therefore, the similarity between the inputs is a crucial one. The input representation, leading to a reinforcement of similarities between the inputs from the same class and deterioration of the similarities between the inputs from different classes, may lead to enhancement of the classification performance. Following this line, in pattern recognition literatures [PC86], [TG74], an input feature  $s$  is considered to be important if the compactness and interclass distance of all the classes along the  $s$ th axis is high. In this chapter we attempt to exploit this criterion to measure the importance of each feature. The compactness of the classes are affected when the classes are overlapping and the patterns with the same  $s$ th feature have different class labels. In card games it indeed happens because the output bids are fuzzy and the output bids are not unique for the same card. It implies that the compactness of the classes can be estimated if we can quantify the roughness and fuzziness associated with the  $s$ th feature. In this spirit, a rough-fuzzy set [DP92] theoretic measure rough-fuzzy entropy is proposed as a criterion function. To measure the rough-fuzzy entropy, it is essential to know the fuzzy membership values of the training data in the output classes. *Possibilistic* K-means algorithm is proposed to accomplish it.

As a search technique, an iterative method is adopted here. The iterative procedure starts assuming equal importance for all the features. At the first iteration, the value of rough-fuzzy entropy is calculated, and using it the set of importance is updated. In the next iteration the input features are weighted by this set of importance. The rough-fuzzy entropy is further calculated for this modified feature vector, and the importance are updated accordingly. The iterative method goes on until the criterion function attains a local minimum. Subsequently, the resultant set of weights is used to bias the input representation of each hand so that more important features get more weightages, and eventually result in a better classification.

The organization of the chapter is as follows: Section 4.2 discusses fuzzy K-nearest neighbors algorithm. Section 4.3 embodies the proposed method. Section 4.4 illustrates the efficacy of the proposed method through some experiments. The basics of rough sets and rough-fuzzy sets can be found in Appendix-B and Appendix-C.

## 4.2 Background of Fuzzy K-Nearest Neighbors Algorithm

Fuzzy K-nearest neighbors algorithm (FKNN) classifies an input pattern  $x$  by assigning it a fuzzy membership value. The membership of  $x$  depends on a) the vector distance between  $x$  and the K-nearest ( $K$  is a positive integer) input training patterns, and b) the memberships of those neighboring training patterns in the possible classes. Let  $X = \{x_1, x_2, \dots, x_n\}$  be a set of input training patterns for whom the corresponding membership assignments are already known. Let  $\mu_c(x_i)$  be the membership of the  $i$ th training pattern in the  $c$ th class and  $1 \leq K \leq n$ . The initial memberships on each training pattern can be assigned in the following two ways [KH85]:

1. Crisp membership: Each training pattern can have complete membership in their known class and nonmembership in all other classes.
2. Constrained fuzzy membership: The K-nearest neighbors of each training pattern are found, and the membership in each class is assigned according to the following equation:

$$\mu_c(x_i) = \begin{cases} 0.51 + n_j/K & \text{if } j = i \\ 0.49 & \text{otherwise} \end{cases} \quad (4.1)$$

The value  $n_j$  is the number of the neighbours found which belong to the  $j$ th class. This initialization technique fuzzifies the memberships of the labelled samples that are in the overlapping class regions. Moreover, the samples that are well away from the overlapping area, are assigned with complete membership in the known class. Consequently, an unknown sample lying in the overlapping region will be influenced to a lesser extent by the labelled samples that are also in the overlapping area.

The algorithm to find the membership of a test pattern  $x$  in the  $c$ th class, i.e.,  $\mu_c(x)$ , is shown in Fig. 4.1 [KH85]. In Equation (4.2) of Fig. 4.1,  $q$  determines how strongly the distance is weighted when calculating each neighbour's contribution to the membership value. Generally, the value of  $q$  is taken as 2.

```

Set  $i=1$ .
DO UNTIL (K-nearest neighbors of  $\mathbf{x}$  are found)
    Determine the distance between  $\mathbf{x}$  and  $\mathbf{x}_i$ .
    IF ( $i \leq K$ )
        Include  $\mathbf{x}_i$  in the set of K-nearest neighbors.
    ELSE IF ( $\mathbf{x}_i$  is closer to  $\mathbf{x}$  than any previous nearest
neighbor)
        Delete the farthest of the K-nearest neighbors.
        Include  $\mathbf{x}_i$  in the set of K-nearest neighbors.
    END IF
    Set  $i = i + 1$ .
END DO UNTIL
Set  $c = 1$ .
DO UNTIL ( $\mathbf{x}$  is assigned membership in all classes)
    Determine  $\mu_c(\mathbf{x})$  using

$$\mu_c(\mathbf{x}) = \frac{\sum_{k=1}^K \mu_c(\mathbf{x}_k) \left(1 / \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}\right)}{\sum_{k=1}^K \left(1 / \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}\right)} \quad (4.2)$$

    Set  $c = c + 1$ .
END DO UNTIL

```

Fig. 4.1: Fuzzy K-nearest neighbors algorithm. The input consists of a set of labelled patterns and a test pattern. The output is the class membership value of the test pattern.

**Lemma 4.1:** *In a C-class classification problem, the class membership assignments on the test patterns by the FKNN are constrained fuzzy.*

**Proof.** The membership of the test patterns are constrained fuzzy because the initial class membership values for the training patterns are constrained fuzzy. It can be formally shown by the following steps:

$$\begin{aligned} \sum_{c=1}^C \mu_c(\mathbf{x}) &= \sum_{c=1}^C \frac{\sum_{k=1}^K \mu_c(\mathbf{x}_k) \left(1 / \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}\right)}{\sum_{k=1}^K \left(1 / \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}\right)} \\ &= \frac{\sum_{k=1}^K \left(1 / \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}\right) \sum_{c=1}^C \mu_c(\mathbf{x}_k)}{\sum_{k=1}^K \left(1 / \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}\right)} \end{aligned}$$

Since  $\sum_{c=1}^C \mu_c(\mathbf{x}_k) = 1$ ,

$$\begin{aligned} \sum_{c=1}^C \mu_c(\mathbf{x}) &= \frac{\sum_{k=1}^K \left(1 / \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}\right)}{\sum_{k=1}^K \left(1 / \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}\right)} \\ &= 1 \end{aligned} \tag{4.3}$$

■

Consequently, the membership values assigned on each test pattern cannot distinguish between equal evidence and ignorance.

## 4.3 Proposed Method

### 4.3.1 Criterion Function

In an N dimensional input pattern  $\mathbf{x} \in \mathbf{X}$ , the sth feature is considered to be important if the compactness and the interclass distance of all the classes along the sth ( $1 \leq s \leq N$ ) axis is high. A measure of compactness and interclass distance can be a criterion to measure the importance of the sth feature. This measurement becomes complicated because two patterns  $\mathbf{x}_u$  and  $\mathbf{x}_v$  may be identical based on their sth feature; but they may belong to two different classes. That is, the relationship between the sth feature and the class labels may be a one-to-many mapping. This lack of discriminatory power of the feature is due to the fact that we are not considering other features like the player's past

experience, vulnerability, etc., into our account. In other words, we do not have sufficient amount of information about the problem. To determine the importance of the  $s$ th feature with such incomplete knowledge, the concept of rough set is helpful. In the terminology of rough set, two input patterns  $\mathbf{x}_u$  and  $\mathbf{x}_v$ , are called indiscernible or indistinguishable with respect to the  $s$ th feature when the  $s$ th component of these two patterns have the same value. Mathematically, it can be stated as

$$\mathbf{x}_u R^s \mathbf{x}_v, \text{ iff } x_{us} = x_{vs} \quad (4.4)$$

where  $R^s$  is a binary relation over  $X \times X$ . Obviously,  $R^s$  is an equivalence relation. Therefore,  $R^s$  partitions  $X$  into a set of equivalence classes, namely  $\{X_1^s, X_2^s, \dots, X_H^s\}$ . The  $s$ th feature alone is sufficient to classify all the input patterns to the  $c$ th class iff  $X/R^s$ , i.e.,  $\{X_1^s, X_2^s, \dots, X_H^s\}$ , approximates the boundary of  $C_c$  accurately. In this case  $BND_R(C_c) = \emptyset$  or  $\overline{R}^s(C_c) = \underline{R}^s(C_c)$ . It implies each  $X_i^s$ ,  $1 \leq i \leq H$ , either belongs to the positive region of  $C_c$  or negative region of  $C_c$ . If this condition holds, the output class  $C_c$  achieves a high degree of compactness and large interclass distance along the  $s$ th axis. Therefore, the extent to which  $X/R^s$  approximates the output class  $C_c$  can be a measure of importance of the  $s$ th feature to classify the patterns to that class.

In the opening bid problem the output classes are overlapping, and hence,  $C_c$  is a fuzzy set. This brings the concept of a rough-fuzzy set. In the current context, a rough-fuzzy set  $\langle \overline{R}^s(C_c), \underline{R}^s(C_c) \rangle$  is defined as follows: Lower approximates  $\underline{R}^s(C_c)$  and upper approximates  $\overline{R}^s(C_c)$  of  $C_c$  are fuzzy sets of  $X/R^s$ , with membership functions defined by [DP92]

$$\mu_{\underline{R}^s(C_c)}(X_i) = \inf \{ \mu_{C_c}(\mathbf{x}) \mid \mathbf{x} \in X_i \} \quad (4.5-a)$$

$$\mu_{\overline{R}^s(C_c)}(X_i) = \sup \{ \mu_{C_c}(\mathbf{x}) \mid \mathbf{x} \in X_i \} \quad (4.5-b)$$

Here,  $\mu_{\underline{R}^s(C_c)}(X_i)$  and  $\mu_{\overline{R}^s(C_c)}(X_i)$  are the membership values of  $X_i$  in  $\underline{R}^s(C_c)$  and  $\overline{R}^s(C_c)$ , respectively. Since the number of training patterns is finite for all practical purposes, we can substitute inf by min and sup by max in (4.5-a) and (4.5-b), respectively.

To measure the importance of the  $s$ th feature to classify all the input patterns into the  $c$ th class, we define rough-fuzzy entropy for the  $s$ th feature and the  $c$ th class as

$$\mathcal{H}_c^s = - \frac{1}{H \ln 2} \sum_{i=1}^H \left[ \mu_i \ln \mu_i + (1 - \mu_i) \ln(1 - \mu_i) \right] \quad (4.6)$$

where  $\mu_i$  represents either  $\mu_{\underline{R}^s(C_c)}(X_i)$  or  $\mu_{\overline{R}^s(C_c)}(X_i)$  throughout the equation. From (4.6), it can be noticed that  $\mathcal{H}_c^s$  increases monotonically in  $[0, 0.5]$  and decreases monotonically

in  $[0.5, 1]$ . It reaches the maximum value when  $\mu_i = 0.5 \forall i$ , and minimum value when  $\mu_i = 0$  or  $1 \forall i$  [PB95]. The lower the value of  $\mathcal{H}_c^s$  is, the greater is the number of  $X_i$ 's having  $\mu_i \approx 1$  or  $\mu_i \approx 0$ , i.e., less is the difficulty in deciding whether  $X_i$  can be considered a member of  $C_c$  or not. In particular, when  $\mu_i \approx 1$ , greater is the tendency of  $X_i$  to form a compact class  $C_c$  along the sth axis, resulting in less internal scatter along the sth axis. Moreover, when  $\mu_i \approx 0$ ,  $X_i$  is far away from the cth class, and hence, the interclass distance increases along the sth axis. On the other hand, when  $\mu_i \approx 0.5$ ,  $X_i$  lies in between  $C_c$  and the other classes along the sth axis. Hence, both compactness and interclass distance along the sth axis decrease. The reliability of a feature  $s$ , in characterizing the class  $C_c$ , increases as the corresponding  $\mathcal{H}_c^s$  value decreases. Therefore,  $\mathcal{H}_c^s$  quantifies the importance of the sth input feature for the class  $C_c$ . We introduce total rough fuzzy entropy for the sth feature to quantify the importance of the sth input feature for all the classes. It is defined as

$$\mathcal{H}^s = \sum_{c=1}^c P_c \mathcal{H}_c^s \quad (4.7)$$

Here  $P_c$  is the weightage that has to be given to the cth class. One possible choice for  $P_c$  is the *a priori* probability of the cth class. Note that  $\mathcal{H}^s$  lies in  $[0, 1]$ . Evidently, the more the value of  $\mathcal{H}^s$  is, the less is the importance of the sth feature.

The value of  $\mathcal{H}_c^s$  depends on the choice of  $\mu_i$ . When we take  $\mu_i = \mu_{\underline{R}^s(C_c)}(X_i)$ , we are basically pessimistic as (4.5-a) involves min operator. Similarly, when we assume  $\mu_i = \mu_{\overline{R}^s(C_c)}(X_i)$ , we become optimistic as (4.5-b) involves *max* operator. In (4.7) these two choices result in the two extreme bounds for  $\mathcal{H}^s$ . It indeed depends on the application which one we should take as the importance [PP92], because (4.5-a) and (4.5-b) are equivalent to asking the question "to what extent  $\{\mathbf{x} | \mathbf{x} \in X_i\}$  as a whole belongs to  $C_c$ ?". For example, in a quiz team if  $\mu_i$  is the ability of the  $i$ th member, the ability of the team as a whole is  $\max_i(\mu_i)$ , because if one member succeeds the whole team succeeds [PP92]. On the other hand, suppose a group of acrobats are standing in such a manner that all of them fall if any one of them falls. If  $\mu_i$  is the stability of the  $i$ th member, the stability of the team as a whole is  $\min_i(\mu_i)$ . However, our concern also may be to obtain a measure in between these two extreme cases. Hence, we need some kind of aggregation operator in between min and max to generalize the definition of  $\mathcal{H}^s$ . It can be conveniently done by Yager's ordered weighted average (OWA) operator [Yag93].

The term min in (4.5-a) measures the degree to which all the  $X_i$ 's are classified to  $C_c$ . Similarly, the term *max* in (4.5-b) measures the degree to which at least one  $X_i$  is

classified to  $C_c$ . It is natural to consider other t-norm and t-conorm operators [KY95] in place of min and max, respectively. Using OWA operator "softening" is done by changing all to most and at least one to some. A mapping  $W: [0, 1]^u \rightarrow [0, 1]$  is called an OWA operator if there exists a weighting vector  $\mathbf{w} = [\omega_1, \omega_2, \dots, \omega_u]'$  associated with  $W$ . The characteristics of the weighting vector  $\mathbf{w}$  are

1.  $\omega_i \in [0, 1]$ ,
2.  $\sum_{i=1}^u \omega_i = 1$ , and
3.  $\mathcal{W}(a_1, a_2, \dots, a_u) = \omega_1 b_1 + \omega_2 b_2 + \dots + \omega_u b_u$ , where  $b_i$  is the  $i$ th largest element in the collection  $a_1, a_2, \dots, a_u$ .

In [Yag93] Yager illustrated how different assignments of the weights allow implementation of different quantifiers. For example,  $\omega_1 = 1$  and  $\omega_i = 0, \forall i \neq 1$ , provides the *max* operator. On the other hand,  $\omega_u = 1$  and  $\omega_i = 0, \forall i \neq u$  gives the min operator. Moreover,  $\omega_i = \frac{1}{u} \forall i$  yields the average. It shows that the more the weights are near the bottom, the more AND-like the aggregation is, and the more the weights are near the top, the more OR-like the aggregation is.

There are two special types of OWA operators [Yag93] [Cho95], which are useful for extending the concept of rough-fuzzy set. They are called S-OWA-AND and S-OWA-OR operators. The S-OWA-AND operator is defined by

$$\mathcal{W}_\alpha(a_1, a_2, \dots, a_u) = \frac{1-\alpha}{u} \sum_i a_i + \alpha \min_i \{a_i\} \quad (4.8)$$

The parameter  $\alpha$  lies in the unit interval. The closer  $\alpha$  is to one, the more AND-like the aggregation becomes. In (4.5-a) we can obtain the effect of S-OWA-AND operators by replacing  $\min\{\mu_{C_c}(x) \mid x \in X_i\}$  with

$$\frac{1-\alpha}{|X_i|} \sum_{\mathbf{x} \in X_i} \mu_{C_c}(\mathbf{x}) + \alpha \min \left\{ \mu_{C_c}(\mathbf{x}) \mid \mathbf{x} \in X_i \right\} \quad (4.9)$$

On the other hand, the S-OWA-OR operator is for an OR like aggregation. This operator is defined by

$$\mathcal{W}_\beta(a_1, a_2, \dots, a_u) = \frac{1-\beta}{u} \sum_i a_i + \beta \max_i \{a_i\} \quad (4.10)$$

Here again the parameter  $\beta$  lies in the unit interval and the closer  $\beta$  is to 1, the more like a pure OR the operation is. In (4.5-b) we can obtain the effect of S-OWA-AND operators



by replacing  $\max\{\mu_{C_c}(\mathbf{x}) \mid \mathbf{x} \in X_i\}$  with

$$\frac{1-\beta}{|X_i|} \sum_{\mathbf{x} \in X_i} \mu_{C_c}(\mathbf{x}) + \beta \max\{\mu_{C_c}(\mathbf{x}) \mid \mathbf{x} \in X_i\} \quad (4.11)$$

Thus the OWA operators generalize the definitions given in (4.5-a) and (4.5-b). Consequently, the definition of the criterion function given in Equation (4.7) is also generalized.

### 4.3.2 Possibilistic K-Nearest Neighbors Algorithm

To calculate (4.7), we need to determine the membership values  $\mu_c(\mathbf{x}) \forall c \forall \mathbf{x}$ . Since the membership assignment should be possibilistic to extract the maximum advantage of fuzzy sets, we modify the FKNN algorithm to *possibilistic K-nearest neighbors* (PKNN) algorithm. Other than Equation (4.2), the steps of the PKNN algorithm are exactly similar to that of the FKNN algorithm. In the PKNN algorithm Equation (4.2) is modified as

$$\mu_c(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^K \frac{\mu_c(\mathbf{x}_k)}{1 + \kappa \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}} \quad (4.12)$$

where  $\kappa$  is a parameter that decides the bandwidth of the membership, i.e., the point at which  $\mu_c(\mathbf{x})$  attains the value 0.5. One way to assign the value of  $\kappa$  is by making it equal to the average distance between any two training patterns. The initial memberships can be assigned in the following three ways: a) *Crisp initialization*: As done in the case of FKNN. b) *Constrained fuzzy initialization*: As done in the case of FKNN. c) *Possibilistic initialization*: The initialisation can be possibilistic if certain domain specific knowledge is present.

**Lemma 4.2:** *For a C-class classification problem, the membership assignments of the test patterns in the PKNN are possibilistic even if the initial class memberships for the training patterns are crisp or constrained fuzzy.*

*Proof.*

$$\begin{aligned} \sum_{c=1}^C \mu_{C_c}(x) &= \sum_{c=1}^C \frac{1}{K} \sum_{k=1}^K \frac{\mu_c(\mathbf{x}_k)}{1 + \kappa \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}} \\ &= \frac{1}{K} \sum_{k=1}^K \frac{1}{1 + \kappa \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}} \sum_{c=1}^C \mu_c(\mathbf{x}_k) \\ &= \frac{1}{K} \sum_{k=1}^K \frac{1}{1 + \kappa \|\mathbf{x} - \mathbf{x}_k\|^{2/(q-1)}} \end{aligned} \quad (4.13)$$

Since  $\sum_{c=1}^C \mu_{C_c}(x)$  needs not to be equal to a constant, the resultant classification procedure is *possibilistic* [KY95] [PB95]. ■

In the opening bid problem, the membership values of all the training patterns can be obtained in crisp form. The sole purpose of using the PKNN algorithm is to **fuzzify** the crisp membership values of the training patterns in a possibilistic manner. All training patterns are used to construct the PKNN algorithm. Now a training pattern is considered as a test pattern for the PKNN. The PKNN algorithm is used for only one iteration. The closest neighbor of any test pattern will be the pattern itself. The output of the PKNN algorithm will give the possibilistic class membership values of the test pattern. The possibilistic membership value corresponding to each training pattern can be found similarly. This technique is similar to obtaining a blurred image from a noisy image (i.e., image smoothing).

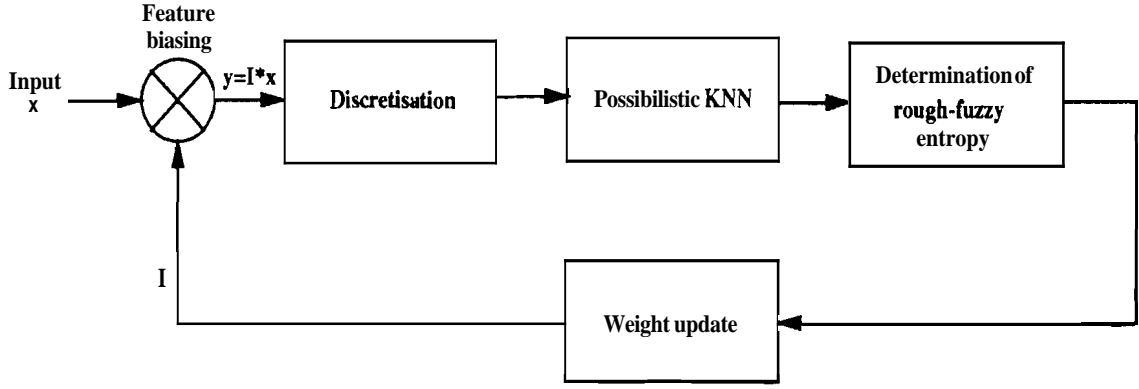
Experimentally it is observed that the performance of the PKNN is comparable to that of the FKNN. However, the PKNN has an edge over the FKNN as the membership values generated by the first one is possibilistic, whereas the membership values generated by the later one is constrained fuzzy.

### 4.3.3 Optimization Technique and Weight Update

Depending on the importance of a feature, the feature is biased by assigning some **weightage** on it. If the  $j$ th feature is very important, then the importance of the feature  $I_j$  is assumed to be close to 1. On the other hand, if the feature is redundant, then the **value** of  $I_j$  is taken as 0. The importance is found using an iterative procedure (see Fig. 4.2). The iterative process starts with the original input  $\mathbf{x} = [x_1, x_2, \dots, x_N]$ . The importance of each input feature is initially assigned as 1, i.e., at starting  $\mathbf{I} = [1, 1, \dots, 1]$ . In the first iteration, the input features are weighted by  $\mathbf{I}$  such that the modified feature vector becomes  $\mathbf{y} = \mathbf{I} * \mathbf{x}$ . Here the operation  $*$  signifies a component wise multiplication between the two vectors, i.e.,  $\mathbf{I} * \mathbf{x} = [I_1 x_1, I_2 x_2, \dots, I_N x_N]$ . The PKNN algorithm is applied on the modified feature vector (which is discrete) to determine the fuzzy class membership values. The change of importance  $\Delta \mathbf{I}^1 = [\Delta I_1^1, \Delta I_2^1, \dots, \Delta I_N^1]$  for the modified feature is determined using

$$\Delta I_j^1 = (1 - \mathcal{H}^s) \forall s \quad (4.14)$$

Now the value of importance is updated as  $\mathbf{I} = \mathbf{I} * \Delta \mathbf{I}^1$ . In the next iteration, the input features are weighted again using  $\mathbf{y} = \mathbf{I} * \mathbf{x}$ . Since this modified input set is not dis-



**Fig. 4.2:** Flow diagram to show how the feature weightages are determined iteratively using rough-fuzzy method. The training patterns are the inputs to the flow diagram. The importance  $I$  is the output of the of the flow diagram.  $I * x$  implies  $[I_1x_1, I_2x_2, \dots, I_Nx_N]$ .

crete, it cannot be used directly to calculate the rough-fuzzy entropy. It is discretized by rounding each feature value to the closest integer value. The fuzzy membership values corresponding to the modified feature vectors are reevaluated from the PKNN algorithm. These membership values are used to calculate the rough-fuzzy entropy. Again the change of importance ( $\Delta I^2$ ) is determined, and the importance is updated as  $I = I * \Delta I^2$ . The iterative process is continued until a local minimum is reached, i.e., there is no significant change in the value of

$$\mathcal{H} = \sum_3 \mathcal{H}^s \quad (4.15)$$

Finally the modified feature vector is  $y = I * x$ , where  $I$  is the importance after the final iteration. The complete algorithm is shown in Fig. 4.3. Application of this method will result in more impact on the distance measure by the important features. For instance, the distance between the modified feature vectors  $y_i$  and  $y_j$  is  $d(y_i, y_j) = \sqrt{(y_{i1} - y_{j1})^2 + (y_{i2} - y_{j2})^2 + \dots + (y_{iN} - y_{jN})^2} = \sqrt{I_1^2(x_{i1} - x_{j1})^2 + I_2^2(x_{i2} - x_{j2})^2 + \dots + I_N^2(x_{iN} - x_{jN})^2}$ . If  $I_u \gg I_v \forall v \neq u$  and  $u, v \in \{1, 2, \dots, N\}$ , then the value of  $d(y_i, y_j)$  will be dictated by the  $u$ th feature only.

#### 4.4 Results and Discussion

Before using the proposed method on the opening bid problem, we will show the effectiveness of the proposed method on some artificially generated two-dimensional data sets.

```

Initialize  $I$  as an  $N$ -dimensional vector  $[1,1,\dots,1]$ .
Set  $i = 1$ .
Assign the maximum possible value of  $\mathcal{H}$ , i.e.,  $N$  to  $\mathcal{H}$ .
DO
     $\mathcal{H}^{\text{old}} = \mathcal{H}$ 
    Find modified feature vectors using  $\mathbf{y} = \mathbf{x} * \mathbf{I}$ .
    Discretize the modifies feature vector.
    Use PKNN to determine the fuzzy memberships of the
    training patterns to all the classes.
    FOR feature  $s = 1$  to  $N$ 
        FOR class  $c = 1$  to  $C$ 
            Compute  $\mathcal{H}_c^s$  from (4.6) .
            Calculate the a priori probability  $P_c$ .
        END FOR
        Compute  $\mathcal{H}^s$  from (4.7)
        Determine  $\Delta I^i$  using (4.14) .
    END FOR
    Compute  $I = \mathbf{I} * \Delta \mathbf{I}^i$ .
    Set  $i = i + 1$ .
    Compute  $3l$  from (4.15) .
END DO UNTIL  $\text{abs}(\mathcal{H}^{\text{old}} - \mathcal{H})$  is larger than certain prespecified
constant
 $I$  stores the importance of all the features.

```

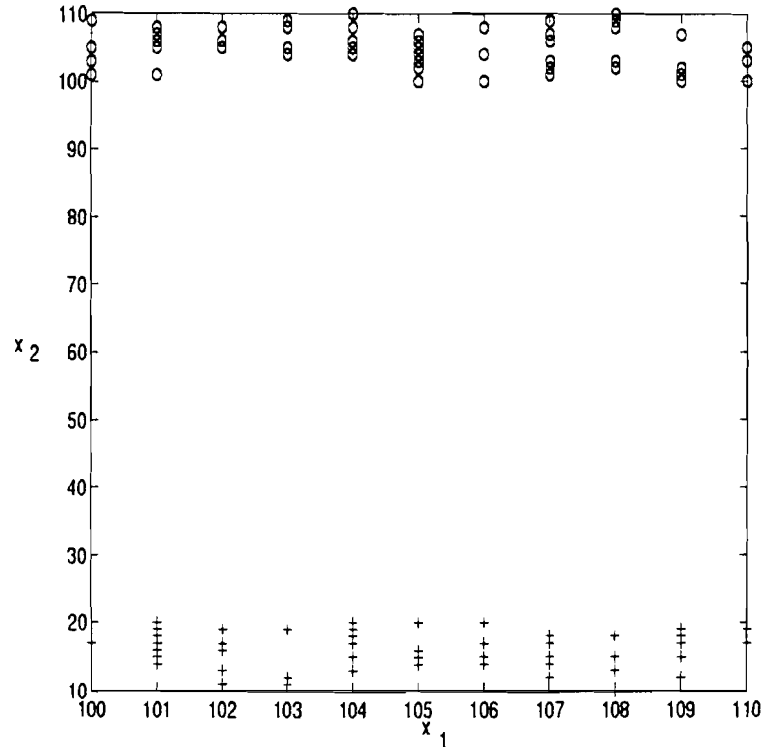
**Fig. 4.3:** The proposed algorithm to determine the importance of input features using the rough-fuzzy entropy. The input to the algorithm is the set of training patterns and the output of the algorithm is the importance of the features. The term  $\text{abs}(\mathbf{x})$  represents the absolute value of  $\mathbf{x}$ .

**Table 4.1:** Importance of features against the number of iterations for the data set shown in Fig. 4.4.

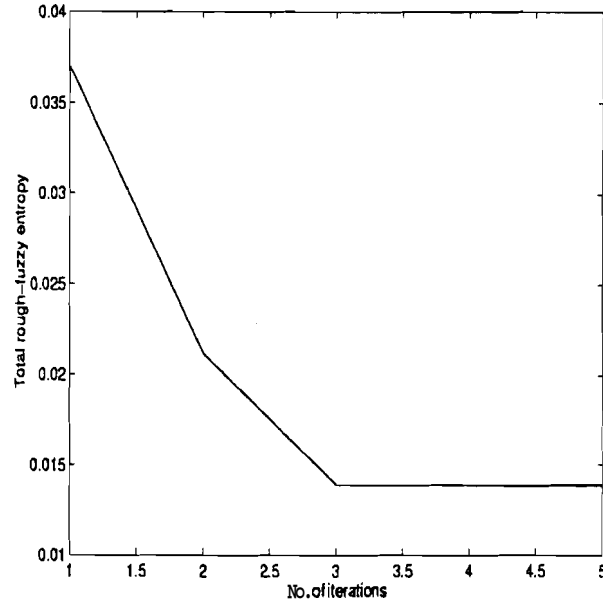
No. of iterations	Importance of $x_1$	Importance of $x_2$
1	1.0000	1.0000
2	0.0522	1.0000
3	0.0001	1.0000
4	0.0000	1.0000
5	0.0000	1.0000

For the first experiment we use the 2-Class data set shown in Fig. 4.4. The data is discrete. The number of inputs is 120, and the dimension of each input is two. In the data set, some patterns from the class  $C_1$  and  $C_2$  are present with the same  $x_1$  value. Hence the classes are indistinguishable based on  $x_1$ . It creates rough uncertainty. On the other hand, the feature  $x_2$  is sufficient for classification. We used PKNN to determine the possibilistic membership values of the inputs. Since the overlaps between the classes are minimum, the generated memberships are almost crisp. The parameter  $P_c$  of the PKNN is made equal to the fraction of the number of training patterns in the  $c$ th class to the total number of patterns. The importance of each feature is found by using  $\alpha = 0.5$ . Fig. 4.5 shows the change of total rough-fuzzy entropy against the number of iterations. The iterative process converged within 5 iterations. The importance of both the features after each iteration are normalized. Otherwise after a certain number of iterations, the importance will be so small that **roundoff** error will take place in digital computers. The change of importance (normalized) is shown in Table 4.1. The final importance for  $x_1$  and  $x_2$  are 0 and 1, respectively. The final importance for  $x_1$  (or  $x_2$ ) is obtained by multiplying all the entries present in the first (second) column of Table 4.1. The value of final importance indicates that the feature  $x_1$  is redundant and the feature  $x_2$  is essential. It tallies with our visual observation also.

The next experiment is on the data set of a 2-class problem. The number of inputs is 140, and the dimension of each input is 2 (see Fig. 4.6). The features are all discrete. In this problem the output classes are indistinguishable with respect to the feature  $x_1$ . The classes are quite separable with respect to the feature  $x_2$ . Unlike the data set shown in Fig. 4.4, here the output classes are overlapping to some extent. The possibilistic membership values are determined using the PKNN. The importance of each feature is



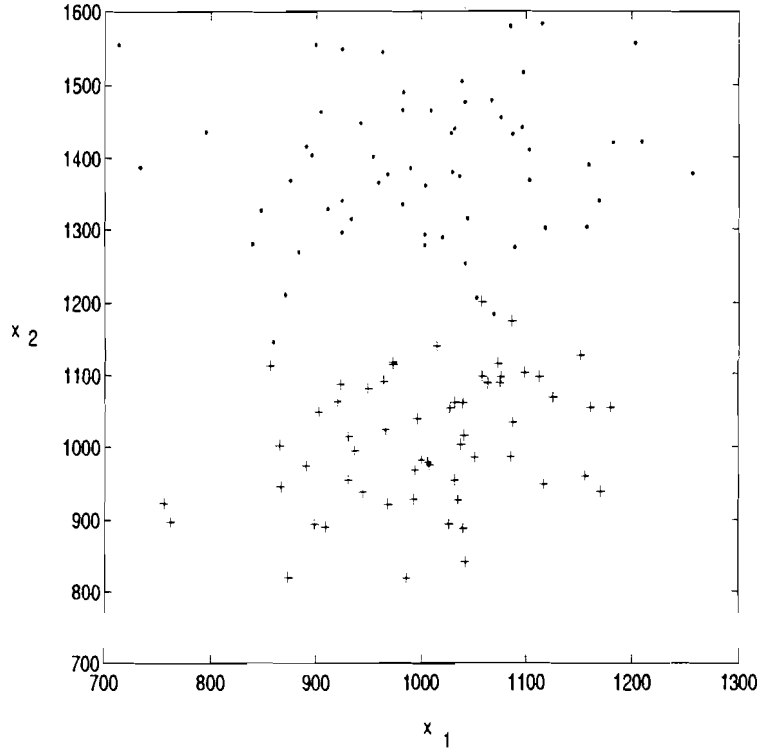
**Fig. 4.4:** Artificially generated input data set for the first experiment. The horizontal and vertical axes represent the feature  $x_1$  and  $x_2$ , respectively. The points with the symbols '+' and '.' represent the patterns from the classes  $C_1$  and  $C_2$ , respectively. The classes are indistinguishable with respect to the feature  $x_1$ . But the classes are distinguishable with respect to the feature  $x_2$ .



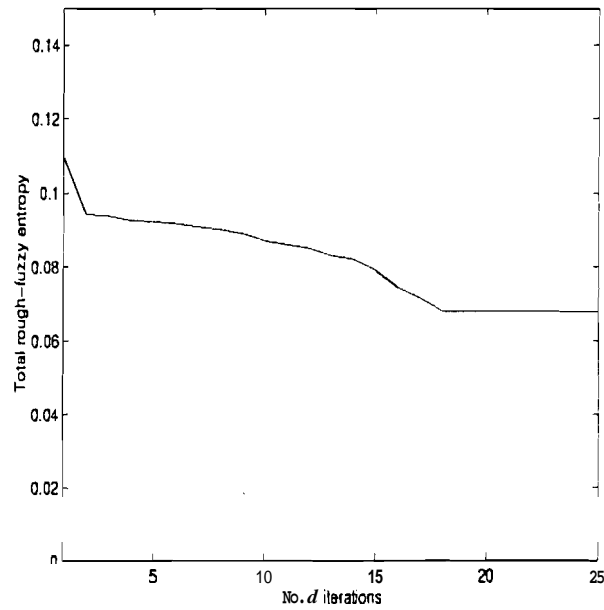
**Fig. 4.5:** Change of total rough-fuzzy entropy with the number of iterations for the data set shown in Fig. 4.4.

found by using  $\alpha = 0.5$ . Fig. 4.7 shows the change of total rough-fuzzy entropy against the number of iterations. The iterative process converges within 25 iterations. The change of importance after each iteration is shown in Table. 4.2. The final importance of the features  $x_1$  and  $x_2$  are 0.0511 and 1.0000, respectively. It implies that the feature  $x_1$  is not an important feature.

Next we apply the proposed method on the opening bid problem. We initially collected a set of patterns with class labels corresponding to first level bids. The crisp membership value corresponding to each pattern is known. The possibilistic membership of each input pattern is determined by using the PKNN algorithm. In the PKNN,  $P_c$  is taken as the fraction of the number of training patterns in the  $c$ th class to the total number of patterns. The importance of each feature is found by using  $\alpha = 0.7$ . Instead of  $\alpha = 0.5$ , we have kept  $\alpha = 0.7$  so that the calculation becomes slightly optimistic. The iterative process was continued for 25 iterations. The value of rough-fuzzy entropy against the number of iterations is shown in Fig. 4.8. The ranking of the features, as given by the proposed method, in a decreasing order is: UA, OK,  $\clubsuit A$ , UA,  $\diamond A$ ,  $\spadesuit Q$ ,  $\spadesuit K$ ,  $\spadesuit 9$ ,  $\heartsuit K$ ,  $\diamond 7$ ,  $\diamond Q$ ,  $\diamond 9$ ,  $\spadesuit T$ ,  $\spadesuit 2$ ,  $\diamond 5$ ,  $\heartsuit 4$ ,  $\clubsuit 5$ , ..., 03,  $\spadesuit T$ , ...,  $\diamond 8$ ,  $\clubsuit 8$ ,  $\heartsuit 4$ ,  $\spadesuit 2$  (here "T" represents 10). It implies that for the first level bids, Ace, King, Queen, Jack cards are more important, which is also as per the notion of Bridge players. Thereafter, we took "TrainingSet1"



**Fig. 4.6:** Input data set for the second experiment. The horizontal and vertical axes represent the features  $x_1$  and  $x_2$ , respectively. The points with the symbols '+' and '.' represent the patterns from the classes  $C_1$  and  $C_2$ , respectively. The classes are indistinguishable with respect to the feature  $x_1$ . But the classes are distinguishable with respect to the feature  $x_2$ .



**Fig. 4.7:** Change of total rough-fuzzy entropy with the number of iterations for the data set shown in Fig. 4.6.



**Table 4.2:** Importance of features against the number of iterations for the data set shown in Fig. 4.6.

No. of iterations	Importance of $x_1$	Importance of $x_2$
1	1.0000	1.0000
2	0.9769	1.0000
3	0.9453	1.0000
4	0.9118	1.0000
5	0.8777	1.0000
6	0.8433	1.0000
7	0.8056	1.0000
8	0.7673	1.0000
9	0.7289	1.0000
10	0.6859	1.0000
11	0.6429	1.0000
12	0.5997	1.0000
13	0.5522	1.0000
14	0.5014	1.0000
15	0.4530	1.0000
16	0.4015	1.0000
17	0.3503	1.0000
18	0.2959	1.0000
19	0.2429	1.0000
20	0.1842	1.0000
21	0.1282	1.0000
22	0.6781	1.0000
23	0.9832	1.0000
24	0.9961	1.0000
25	0.9901	1.0000

Table 4.3: Classification performance of FFNNs with BP algorithm for the first level bids.

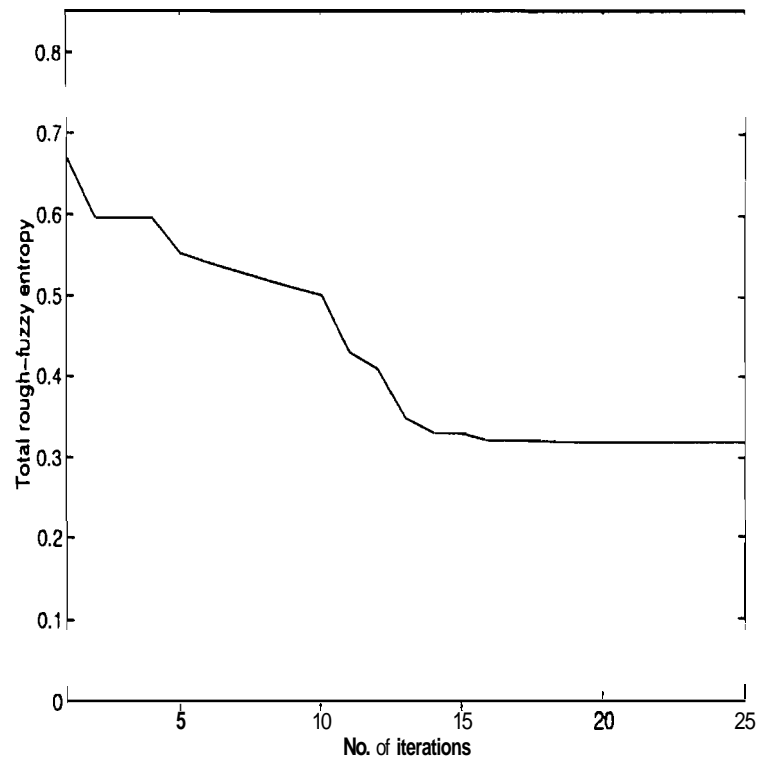
Input	Pass	1C	1D	1S	1H	1N	Overall
Raw	85.12%	66.83%	63.13%	71.82%	77.16%	64.52%	71.43%
Modified	85.82%	69.94%	72.13%	70.01%	78.25%	69.92%	74.34%

to train to train a three layered FFNN by backpropagation algorithm. This is the same training set that we used in chapter 3 to train the FFNNs for first level bids. Each component of a pattern of this set is multiplied by the corresponding importance. The number of input nodes, hidden nodes and output nodes are 52, 50 and 6, respectively. We used the set "TestSet1" for testing. While testing also we use the modified representation for the test patterns. The class corresponding to a test pattern is chosen as the label corresponding to the output node with the highest output. With the original input representation, the network takes 2000 iterations to converge, whereas with the modified representation the same network takes 1650 iterations to converge, which is significantly better. Overall classification performance on the same test set with the original (raw) and modified representation are given in the first and second rows of Table 4.3.

Similarly the proposed method is used for the bids of second and third levels. The training sets "TrainingSet2" and "TrainingSet2" are again considered to train the modules for the second and third level bids, respectively. The importance are used to weight the patterns of the training sets. These weighted patterns are used to train FFNNs with BP algorithm. The FFNN for the second level bids has 52 input nodes, 50 hidden nodes and 5 output nodes. The FFNN for the third level bids has 52 input nodes, 50 hidden nodes and 4 output nodes. The classification results for the second and third level bids on "TestSet2" and "TestSet2" are given in the second row of Table 4.4 and 4.5. The first row of Table 4.4 and 4.5 show the classification results when the original input (raw) is fed to the network. The comparative results show a significant improvement of the classification results.

## 4.5 Summary

This chapter proposes a filter-type feature weighting method. Since the class discriminatory property of all the cards are not same to classify an input hand, the representation



**Fig. 4.8:** Change of total rough-fuzzy entropy for the data set with first level bids.

**Table 4.4:** Classification performance of FFNNs with BP algorithm for the second level bids.

Input	2C	2D	2S	2H	2N	Overall
Raw	61.04%	75.71%	77.56%	72.33%	74.34%	72.19%
Modified	67.45%	76.88%	78.19%	75.43%	74.98%	74.59%

**Table 4.5:** Classification performance of FFNNs with BP algorithm for the third level bids.

Input	3C	3D	3S	3H	Overall
Raw	75.17%	77.28%	83.54%	84.97%	80.24%
Modified	85.14%	79.43%	82.46%	85.11%	83.03%

of each input pattern should be biased based on the importance of each card. This necessitates a way to measure the importance of each card, i.e., each feature, individually. A filter type approach, which does not depend on the classifier being used, is employed. As a criterion function, rough-fuzzy entropy is used. The criterion function is optimized iteratively. To determine the fuzzy membership values of the training patterns, other than the PKNN algorithm we could have used other fuzzy classifiers that do not need any *a priori* information about the structure of the data.

For different values of  $\alpha$  and  $\beta$  we get some interesting results. For example, if  $\alpha = 0$  or  $\beta = 0$  in (4.9) and (4.11), then  $\mu_{\underline{R}^*(C_c)}(X_i) = \mu_{\overline{R}^*(C_c)}(X_i) = \frac{1}{|X_i|} \sum_{\mathbf{x} \in X_i} \mu_{C_c}(\mathbf{x}) \forall \mathbf{x} \in X_i$ . From Equation (C.2) in Appendix-C, it can be observed that  $\frac{1}{|X_i|} \sum_{\mathbf{x} \in X_i} \mu_{C_c}(\mathbf{x})$  is equal to the rough-fuzzy membership function of  $\mathbf{x}$  for the output class  $C_c$ . Therefore, the rough-fuzzy entropy in Equation (4.6) is equal to the rough-fuzzy entropy proposed in [SY] [SY98d]. In the absence of roughness, each input will be labelled always with unique class label. In this case, if there is no repetition of any input, then the number of equivalence classes will be equal to the number of input data and  $\mu_{\underline{R}^*(C_c)}(X_i) = \mu_{\overline{R}^*(C_c)}(X_i) = \mu_{C_c}(\mathbf{x}_i)$ . Thus, the rough-fuzzy entropy for the  $c$ th class and the  $s$ th feature becomes

$$\mathcal{H}_c^s = -\frac{1}{|X| \ln 2} \sum_{i=1}^{|X|} \left[ \mu_{C_c}(\mathbf{x}_i) \ln(\mu_{C_c}(\mathbf{x}_i)) + (1 - \mu_{C_c}(\mathbf{x}_i)) \ln(1 - \mu_{C_c}(\mathbf{x}_i)) \right] \quad (4.16)$$

It is explicitly the fuzzy entropy proposed for feature selection in [PC86] [Pal92]. If no fuzziness but roughness is present, then  $\mu_{\underline{R}^*(C_c)}(X_i)$  is actually the rough membership function for any pattern  $\mathbf{x} \in X_i$  (see Equation (B.1) in Appendix B). Then the proposed rough-fuzzy entropy can be compared with the definition of rough entropy given in [PWZ88] and [SY].

The advantages of the proposed method are

1. It exploits roughness and fuzziness simultaneously.
2. It is moderately fast.
3. If we seek to find the importance of the features in terms of intervals, then we have to run the algorithm twice with  $\mu_i = \mu_{\underline{R}^*(C_c)}(X_i)$  and  $\mu_i = \mu_{\overline{R}^*(C_c)}(X_i)$ . The importance of the  $s$ th feature is an interval  $[u, v]$ , where  $u$  and  $v$  are the importance with  $\mu_{\underline{R}^*(C_c)}(X_i)$  and  $\mu_{\overline{R}^*(C_c)}(X_i)$ , respectively. Appropriate point in the interval can be chosen based on the given problem. Instead of an interval, by taking a specific value as an importance of the feature, we lose some information. In this

chapter we have adopted a specific value as an importance as processing of the interval may be complicated in the next stages.

4. For feature selection, a threshold value can be chosen for the importance. All features with importance lesser than this **threshold** value will be ignored.
5. It does not depend on the type of the classifier used in the feature analysis stage. It does not need significant domain dependent knowledge.

The drawback of the method is that the resultant importance may not be globally optimum.

For the  $s$ th module, the derivation of the modified feature vectors using the proposed method is a mapping from an  $N$  dimensional discrete space to an  $N$  dimensional continuous space. For the  $s$ th module, we need to find the relation from the continuous  $N$  dimensional space to  $n_s$  dimensional continuous space (assuming that the  $s$ th module has  $n_s$  output classes). In the next two chapters we propose two alternative schemes to capture this relation.

## Chapter 5

# DESIGN OF CLASSIFIER MODULES THROUGH DIRECT CLASSIFICATION

### 5.1 Introduction

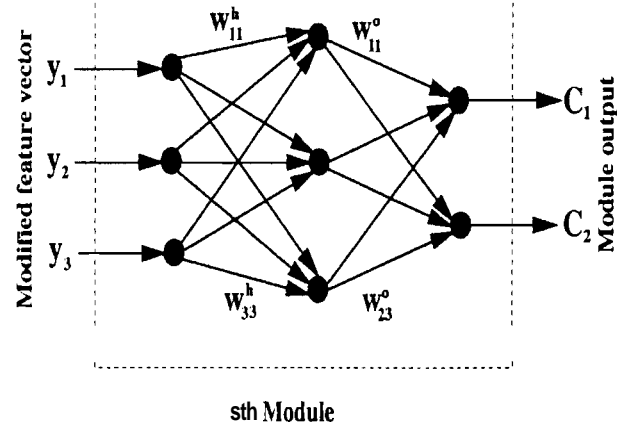
In this chapter an attempt has been made to build a module using direct classification techniques. The aim is to capture the relationship between the modified feature vectors and the fuzzy output classes of the module. For each module, this chapter proposes FFNNs with the BP learning algorithm that minimizes certain fuzzy objective functions from possibilistic classification viewpoint. After training, if a new input pattern is presented to the network, it yields the outputs as class membership values corresponding to the input pattern. The classification performance of the FFNNs can be improved further if the networks are configured optimally. To configure the FFNNs, the BP algorithm with the fuzzy objective functions is embedded into a stochastic search operation.

When the output classes are fuzzy, an input pattern may not necessarily belong to a single class; rather it may belong to more than one class with different degrees of belongingness. The conventional BP learning algorithm is not tailored to this type of fuzzy classification problem. Section 5.2 makes an FFNN powerful by proposing a method to embed fuzzy classification properties into the conventional BP learning algorithm. In section 5.3 we use *evolutionary programming* (EP), a multipoint, controlled, stochastic search and optimization technique, for finding the optimal configuration of the FFNN. Training and configuring the FRNN involves local as well as global search in the parameter space. EP is good for global search, whereas it is slow for local search. Although gradient-based search techniques like BP algorithms are quite fast for local search, they may get stuck in local minima while exploring a search space globally. We combine the BP and EP, and exploit the advantages of both. Efficiency of this hybrid method is further enhanced by incorporating the concepts of adaptive structural mutation.

## 5.2 Feedforward Neural Network Classifiers: Backpropagation Learning Algorithm with Fuzzy Objective Functions

A major drawback of the conventional BP algorithm is that it assigns each input pattern exactly to one of the output classes, assuming well-defined class boundaries. In real life situations boundaries between the classes may be overlapping. This section proposes a method of incorporating fuzzy classification properties into the conventional BP learning algorithm. In the opening bid problem, the inputs are modified feature vectors (crisp) and the output classes are fuzzy. An input pattern may not necessarily belong to a single class; rather it may belong to more than one class with different degrees of **belongingness**. Unlike the conventional BP, here the number of target classes corresponding to each input training pattern may be more than one. The aim of the proposed learning algorithm during training is to minimize an error term, henceforth termed as fuzzy mean square error. The fuzzy mean square error is the overall weighted sum of the square error between the actual network output and all possible target outputs, where the weight signifies the level of belongingness of the input pattern into the corresponding target class. If a modified feature vector is presented to the network after training, it yields the output as class membership values corresponding to the input pattern. We also propose another learning algorithm that tries to minimize an alternative error term, called fuzzy cross entropy, which is a fuzzy counterpart of crisp cross entropy [Hay94]. Although the learning algorithms for the fuzzy mean square error and fuzzy cross entropy differ, the basic philosophy of introducing the concept of fuzzy classification into the crisp error measure is same.

The sum of one's belief that a particular bid is from the class 1C or 1D is not necessarily equal to one. Hence the proposed learning algorithm is derived in such a manner that the sum total of the membership values for a particular pattern to all the classes need not necessarily be equal to one. It implies that the membership assignment is not constrained fuzzy [PB95]; on the other hand, it is possibilistic [PB95]. In the case of constrained fuzzy membership assignment, we show that the learning algorithm, given by Pal and Mitra [PM92], is equivalent to the proposed algorithm. In addition, when the classification is crisp, the proposed learning algorithm reduces to the conventional BP algorithm. Thus, the possibilistic approach of the proposed algorithm leads it to encompass both constrained fuzzy classification and crisp classification.



**Fig. 5.1:** A typical fully connected feedforward neural network with two input nodes, three hidden nodes and two output nodes.

### 5.2.1 Architecture of Feedforward Neural Networks

Let the training set in a C-class problem consists of vector pairs  $\{(y_1, z_1), (y_2, z_2), \dots, (y_n, z_n)\}$ , where  $y_u \in \mathbb{R}^N$  refers to the  $u$ th modified feature vector and  $z_u \in \{t_c | c = 1, 2, \dots, C; t_c \in \mathbb{R}^C\}$  refers to the target output of the network corresponding to this input. Specifically, if  $y_u$  is from the  $k$ th class, then  $z_u = t_k$ , where  $t_{kk} = 1$  and  $t_{ck} = 0 \forall c, c \neq k$ .

The network used here is a multilayer feedforward network, which can have several hidden layers. Without loss of generality, the number of the hidden layers can be assumed to be one with  $H$  hidden nodes. When a modified feature vector  $y_u = [y_{u1}, y_{u2}, \dots, y_{uN}]$  is applied at the input layer of the network, the input units distribute the values to the hidden layer units. The output of the  $j$ th hidden unit is  $o_{uj}^h = f_j^h(\text{net}_{uj}^h) = \frac{1}{1 + \exp(-\text{net}_{uj}^h)}$ , where  $\text{net}_{uj}^h = \sum_{i=1}^N w_{ji}^h y_{ui} + \theta_j^h$ .  $w_{ji}^h$  is the weight of the link from the  $i$ th input node to the  $j$ th hidden node. Here,  $\theta_j^h$  and  $f_j^h$  are the bias term and transfer function of the  $j$ th hidden node. Similarly, the output of the  $k$ th output node is  $o_{uk}^o = f_k^o(\text{net}_{uk}^o) = \frac{1}{1 + \exp(-\text{net}_{uk}^o)}$ , where  $\text{net}_{uk}^o = \sum_{j=1}^H w_{kj}^o f_j^o(\text{net}_{uj}^o) + \theta_k^o$ . The superscripts  $h$  and  $o$  refer to the quantities in the hidden and output layers, respectively (see Fig. 5.1).

### 5.2.2 Training of Feedforward Neural Networks

The adaptive parameters of FFNNs consist of all weights and bias terms. The sole purpose of the training phase is to determine the optimum setting of the weights and bias terms



so as to minimize the difference between the network output and the target output. This difference is referred to as training error of the network. The error measure can be fuzzy mean square error, which is a fuzzy counterpart of the mean square error [Hay94] used in the conventional BP algorithm.

In the conventional BP algorithm, the mean square error for the  $u$ th input pattern is defined as  $\mathcal{E}_u = \frac{1}{2} \sum_{k=1}^C (t_{uk} - o_{uk}^o)^2$ . The use of  $\mathcal{E}_u$  as an error term is justified when each input pattern belongs to only one class. But, in fuzzy classification the input pattern may belong to more than one class with different degrees of belongingness. It implies that the target value of an input pattern may be more than one. In other words, each input pattern can have all possible target values with different membership values (certain membership values may be zero also). Through training, the network attempts to reach those target values weighted by different membership values. In other words, the problem of training can also be conceptually viewed as a fuzzy constraint satisfaction problem. Here the constraint is that each input pattern should belong to a particular class, and the associated membership value signifies to what extent this constraint should be satisfied. In the training phase, the proposed network adapts the parameters so that these constraints are resolved optimally. For the  $u$ th input pattern the constraints can be mathematically expressed as the *fuzzy mean square error*. It is defined as

$$\mathcal{E}_u^f = \frac{1}{2} \sum_{k=1}^C \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o)^2 \quad (5.1)$$

Here the index of  $\mu$ , i.e.,  $q \in [0, \infty)$  controls the amount of fuzziness present in the classification. Different values of  $q$  signifies to what extent the constraints should be satisfied. When  $q$  is equal to zero, each input pattern tries to attain all the target outputs with equal importance, and ultimately the network learns the mean of all the class centers. When  $q$  is greater than one, the constraints associated with the high membership values get more importance to be resolved. When  $q$  tends to be infinity, only the input pattern that belongs to a class completely, i.e., with membership one, is learned. That means, at  $q \approx \infty$  and  $0 \leq \mu_c(\mathbf{y}_u) \leq 1 \forall c$ ,  $\mathcal{E}_u^f$  is equivalent to the conventional mean square error  $\mathcal{E}_u$ . Specifically, the larger the value of  $q$  is in  $[0, \infty)$ , the less fuzzier are the membership assignments. As a consequence,  $\mathcal{E}_u^f$  decreases strictly towards zero as  $q$  increases in  $[1, \infty)$  for  $0 < \mu_c(\mathbf{y}_u) < 1 \forall c$ .

**Lemma 5.1:**  $\mathcal{E}_u^f$  is a monotonically decreasing function for  $0 < \mu_c(\mathbf{y}_u) < 1 \forall c$  and  $q \in [1, \infty)$ .

**Proof.** Differentiating  $\mathcal{E}_u^f$  with respect to  $q$  we get

$$\frac{\partial \mathcal{E}_u^f}{\partial q} = \frac{1}{2} \sum_{k=1}^C \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) \ln(\mu_c^q(\mathbf{y}_u)) (t_{ck} - o_{pk}^o)^2 \quad (5.2)$$

$$= \frac{1}{2} \sum_{k=1}^C \sum_{c=1}^C \left[ \mu_c(\mathbf{y}_u) \ln(\mu_c(\mathbf{y}_u)) \right] \left[ \mu_c^{q-1}(\mathbf{y}_u) (t_{ck} - o_{pk}^o)^2 \right] \quad (5.3)$$

With the usual convention that  $x \ln(x) = 0$  if  $x = 0$ , we have  $[\mu_c(\mathbf{y}_u) \ln(\mu_c(\mathbf{y}_u))] \leq 0$  and  $[\mu_c^{q-1}(\mathbf{y}_u) (t_{ck} - o_{pk}^o)^2] \geq 0 \forall c, k$ . Both the inequalities being strict whenever  $0 < \mu_c(\mathbf{y}_u) < 1$ . Hence, when  $0 < \mu_c(\mathbf{y}_u) < 1$ ,  $\mathcal{E}_u^f$  strictly decreases [Bez81] on every finite interval of the form  $[1, b]$  with  $1 < b$ . ■

On the other hand, when  $q$  is less than 1, the constraints associated with the high membership values get less importance to be resolved. Thus,  $q$  controls the extent of the membership sharing between the fuzzy classes. This can be good; on the other hand, one must choose  $q$  to actually implement it. In our work  $q$  is assumed to be one. The role of  $q$  is quite similar to the index of fuzziness in the concentration and dilation operators (found in fuzzy hedge) [KF93], and the index of fuzziness in fuzzy K-means clustering algorithm [Bez81].

Next, we derive the learning laws for the network following the same method as followed in the conventional BP algorithm [Hay94]. Here, we assume that the weight updating  $\Delta w$  takes place after the presentation of each input pattern. Assuming the use of same learning-rate parameter  $\eta$  for all the weight changes made in the network, the change of weights for  $w_{kj}$  and  $w_{ji}$  are calculated in accordance to the gradient-descent rules:  $\Delta w_{kj}^o = -\eta \frac{\partial \mathcal{E}_u^f}{\partial w_{kj}^o}$  and  $\Delta w_{ji}^h = -\eta \frac{\partial \mathcal{E}_u^f}{\partial w_{ji}^h}$ .

Lemma 5.2:  $\Delta w_{kj}^o = \eta \delta_{uk}^o o_{uj}^h$  and  $\Delta w_{ji}^h = \eta \delta_{uj}^h y_{ui}$  where  $\delta_{uk}^o = \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uk}^o (1 - o_{uk}^o)$  and  $\delta_{uj}^h = f_j^h(\text{net}_{uj}^h) \sum_{k=1}^C \delta_{uk}^o w_{kj}^o$ .

**Proof.** The expression for  $\frac{\partial \mathcal{E}_u^f}{\partial w_{kj}^o}$  can be derived as

$$\frac{\partial \mathcal{E}_u^f}{\partial w_{kj}^o} = - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) \frac{\partial f_k^o(\text{net}_{uk}^o)}{\partial (\text{net}_{uk}^o)} \frac{\partial (\text{net}_{uk}^o)}{\partial w_{kj}^o} \quad (5.4)$$

$$= - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) o_{uk}^o (1 - o_{uk}^o) o_{uj}^h \quad (5.5)$$

$$= - \left[ \mu_k^q(\mathbf{y}_u) (t_{kk} - o_{uk}^o) + \sum_{c=1, c \neq k}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) \right] o_{uk}^o (1 - o_{uk}^o) o_{uj}^h \quad (5.6)$$

Since  $t_{kk} = 1$  and  $t_{ck} = 0 \forall c \neq k$ ,

$$\frac{\partial \mathcal{E}_u^f}{\partial w_{kj}^o} = - \left[ \mu_k^q(\mathbf{y}_u)(1 - o_{uk}^o) - \sum_{c=1, c \neq k}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uk}^o (1 - o_{uk}^o) o_{uj}^h \quad (5.7)$$

$$= - \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uk}^o (1 - o_{uk}^o) o_{uj}^h \quad (5.8)$$

Therefore,

$$\Delta w_{kj}^o = \eta \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uk}^o (1 - o_{uk}^o) o_{uj}^h \quad (5.9)$$

$$= \eta \delta_{uk}^o o_{uj}^h \quad (5.10)$$

where  $\delta_{uk}^o = \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uk}^o (1 - o_{uk}^o)$ .

Again, the expression for  $\frac{\partial \mathcal{E}_u^f}{\partial w_{ji}^h}$  can be found as follows:

$$\frac{\partial \mathcal{E}_u^f}{\partial w_{ji}^h} = - \sum_{k=1}^C \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) \frac{\partial f_k^o(\text{net}_{uk}^o)}{\partial (\text{net}_{uk}^o)} \frac{\partial (\text{net}_{uk}^o)}{\partial o_{uj}^o} \frac{\partial o_{uj}^o}{\partial (\text{net}_{uj}^h)} \frac{\partial (\text{net}_{uj}^h)}{\partial w_{ji}^h} \quad (5.11)$$

$$= - \sum_{k=1}^C \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) o_{uk}^o (1 - o_{uk}^o) w_{kj}^o f_j^h(\text{net}_{uj}^h) y_{ui} \quad (5.12)$$

$$= - f_j^h(\text{net}_{uj}^h) y_{ui} \sum_{k=1}^C \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) o_{uk}^o (1 - o_{uk}^o) w_{kj}^o \quad (5.13)$$

Following the steps involved in deriving Equation (5.8) from Equation (5.5), we can write

$$\mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \equiv \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) \quad (5.14)$$

Hence,

$$\frac{\partial \mathcal{E}_u^f}{\partial w_{ji}^h} = - f_j^h(\text{net}_{uj}^h) y_{ui} \sum_{k=1}^C \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uk}^o (1 - o_{uk}^o) w_{kj}^o$$

Finally we can state

$$\Delta w_{ji}^h = \eta f_j^h(\text{net}_{uj}^h) y_{ui} \sum_{k=1}^C \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uk}^o (1 - o_{uk}^o) w_{kj}^o \quad (5.15)$$

$$= \eta f_j^h(\text{net}_{uj}^h) y_{ui} \sum_{k=1}^C \delta_{uk}^o w_{kj}^o \quad (5.16)$$

$$= \eta \delta_{uj}^h y_{ui} \quad (5.17)$$

where  $\delta_{uj}^h = f_j^h(\text{net}_{uj}^h) \sum_{k=1}^C \delta_{uk}^o w_{kj}^o$ .

Therefore,

$$\Delta w_{kj}^o = \eta \delta_{uk}^o o_{uj}^h \quad (5.18-a)$$

$$\Delta w_{ji}^h = \eta \delta_{uj}^h y_{ui} \quad (5.18-b)$$

Now, we generalize the other error measure, i.e., cross entropy. For the  $u$ th input pattern, the cross entropy is defined as

$$\mathcal{H}_u = \sum_{k=1}^C \left( t_{uk} \ln \left( \frac{t_{uk}}{o_{uk}^o} \right) + (1 - t_{uk}) \ln \left( \frac{1 - t_{uk}}{1 - o_{uk}^o} \right) \right) \quad (5.19)$$

Since  $t_{uk}$  is either zero or one, we can rewrite the above definition as

$$\mathcal{H}_u = - \sum_{k=1}^C (t_{uk} \ln(o_{uk}^o) + (1 - t_{uk}) \ln(1 - o_{uk}^o)) \quad (5.20)$$

Following the same logic, as we used to justify the use of the fuzzy mean square error in the place of mean square error, we can generalize  $\mathcal{H}_u$  to its fuzzy counterpart, called fuzzy cross entropy. The fuzzy cross entropy is defined as

$$\mathcal{H}_u^f = - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) \left[ \sum_{k=1}^C (t_{ck} \ln(o_{uk}^o) + (1 - t_{ck}) \ln(1 - o_{uk}^o)) \right] \quad (5.21)$$

It is trivial to show that for  $0 \leq \mu_c(\mathbf{y}_u) \leq 1$ ,  $\mathcal{H}_u^f$  reduces to  $\mathcal{H}_u$  at  $q \approx \infty$ . Here,  $q$  controls the amount of fuzziness in a similar way as it does in Equation (5.1). Consequently, like Lemma 5.1,  $\mathcal{H}_u^f$  decreases strictly to zero as  $q$  increases in  $[1, \infty)$  for  $0 < \mu_c(\mathbf{y}_u) < 1 \forall c$ .

**Lemma 5.3:**  $\mathcal{H}_u^f$  is a monotonically decreasing function for  $q \in [1, \infty)$  and  $0 < \mu_c(\mathbf{y}_u) < 1 \forall c$ .

**Proof.** For  $q > 1$ ,

$$\frac{\partial \mathcal{H}_u^f}{\partial q} = \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) \ln(\mu_c^q(\mathbf{y}_u)) \left[ \sum_{k=1}^C (-t_{ck} \ln(o_{uk}^o) - (1 - t_{ck}) \ln(1 - o_{uk}^o)) \right] \quad (5.22)$$

With the usual convention that  $x \ln(x) = 0$  if  $x = 0$ , we have  $\mu_c(\mathbf{y}_u) \ln(\mu_c(\mathbf{y}_u)) \leq 0$  and  $(-t_{ck} \ln(o_{uk}^o) - (1 - t_{ck}) \ln(1 - o_{uk}^o)) \geq 0 \forall c, k$ . It implies,  $\mathcal{H}_u^f$  decreases monotonically in  $q \in [1, \infty)$ . First inequality becomes strict when  $0 < \mu_c(\mathbf{y}_u) < 1$ . For the fuzzy cross entropy to be a well-defined criterion, we must have the additional constraint  $0 < o_{uk}^o < 1$  on the outputs of the neural networks. Therefore,  $\mathcal{H}_u^f$  can be zero only when  $t_{ck}$  is equal to zero and one, simultaneously; which is impossible. Therefore, when  $0 < \mu_c(\mathbf{y}_u) < 1$ ,

$\mathcal{H}_u^f$  decreases monotonically in a strict manner on every finite interval of the form  $[1, b]$  with  $1 < b$ . ■

**Lemma 5.4:**

$$\Delta w_{kj}^o = \eta \frac{\partial \mathcal{H}_u^f}{\partial w_{kj}^o} = \eta \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uj}^h \quad (5.23-a)$$

$$\Delta w_{ji}^h = \eta \frac{\partial \mathcal{H}_u^f}{\partial w_{ji}^h} = \eta f'(\text{net}_{uj}^h) y_{ui} \sum_{k=1}^C \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] w_{kj}^o \quad (5.23-b)$$

**Proof.** To find  $\frac{\partial \mathcal{H}_u^f}{\partial w_{kj}^o}$ , we differentiate Equation (5.21) with respect to  $w_{kj}^o$ .

$$\frac{\partial \mathcal{H}_u^f}{\partial w_{kj}^o} = - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) \left( \frac{t_{ck}}{o_{uk}^o} - \frac{1 - t_{ck}}{1 - o_{uk}^o} \right) f'_k(\text{net}_{uk}^o) o_{uj}^h \quad (5.24)$$

$$= - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) \frac{(t_{ck} - o_{uk}^o)}{o_{uk}^o (1 - o_{uk}^o)} o_{uk}^o (1 - o_{uk}^o) o_{uj}^h \quad (5.25)$$

$$= - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) o_{uj}^h \quad (5.26)$$

Using the identity (5.14),

$$\frac{\partial \mathcal{H}_u^f}{\partial w_{kj}^o} = - \left[ Y - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] o_{uj}^h \quad (5.27)$$

The value of  $\frac{\partial \mathcal{H}_u^f}{\partial w_{ji}^h}$  can be calculated as

$$\frac{\partial \mathcal{H}_u^f}{\partial w_{ji}^h} = - \sum_{k=1}^C \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) \left( \frac{t_{ck}}{o_{uk}^o} - \frac{1 - t_{ck}}{1 - o_{uk}^o} \right) f'_k(\text{net}_{uk}^o) w_{kj}^o f'_j(\text{net}_{uj}^h) y_{ui} \quad (3.28)$$

$$= - \sum_{k=1}^C \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) \frac{(t_{ck} - o_{uk}^o)}{o_{uk}^o (1 - o_{uk}^o)} o_{uk}^o (1 - o_{uk}^o) w_{kj}^o f'_j(\text{net}_{uj}^h) y_{ui} \quad (5.29)$$

$$= - \sum_{c=1}^C \sum_{k=1}^C \mu_c^q(\mathbf{y}_u) (t_{ck} - o_{uk}^o) w_{kj}^o f'_j(\text{net}_{uj}^h) y_{ui} \quad (5.30)$$

Applying the identity (5.14),

$$\frac{\partial \mathcal{H}_u^f}{\partial w_{kj}^o} = - f'_j(\text{net}_{uj}^h) y_{ui} \sum_{k=1}^C \left[ \mu_k^q(\mathbf{y}_u) - \sum_{c=1}^C \mu_c^q(\mathbf{y}_u) o_{uk}^o \right] w_{kj}^o \quad (5.31)$$

■

Therefore, by introducing the fuzzy concepts in the usual BP error measures, we can obtain a large class of learning equations. Although the exact formulation of the learning

equations for the fuzzy mean square error and fuzzy cross entropy differ, the underlying concept of introduction of fuzziness into the usual error measures is same.

To make the learning faster, the learning rate can be increased or decreased dynamically as the learning algorithm progresses. In addition, momentum term can be used for faster learning.

Now, we illustrate the following particular cases of the proposed learning algorithms.

1. **Crisp classification:** In the case of crisp classification only one component of  $\mu_c^q(\mathbf{y}_u)$   $\forall c = 1, \dots, C$  is one and the remaining components are zero. Thus, the expression for  $\mathcal{E}_u^f$  reduces to the following expression:

$$\mathcal{E}_u^f = -\frac{1}{2} \sum_{k=1}^C \sum_{c=1}^C (t_{cu} - o_{uk})^2 \quad (5.32)$$

which is the mean square error term found in the conventional BP algorithm. Consequently, in a crisp case the learning equations based on the mean square error and fuzzy mean square error become identical. It can be verified easily by making the membership assignments in Equation (5.18-a) and (5.18-b) crisp.

Similarly, in the case of crisp classification, the fuzzy cross entropy reduces to the conventional cross entropy, and consequently, the learning equations for the cross entropy and fuzzy cross entropy become same.

2. **Constrained fuzzy classification:** When  $\sum_c \mu_c(\mathbf{y}_u) = 1 \forall u$  and  $q = 1$ , the learning equations (5.18-a) and (5.18-b) achieve simpler forms as follows:

$$\Delta w_{kj}^o = -\eta \delta_{uk}^o o_{uj}^h \quad (5.33-a)$$

$$\Delta w_{ji}^h = -\eta \delta_{uj}^h y_{ui} \quad (5.33-b)$$

where  $\delta_{uk}^o = [\mu_k(\mathbf{y}_u) - o_{uk}^o] o_{uk}^o (1 - o_{uk}^o)$  and  $\delta_{uj}^h = f_j^h(\text{net}_{uj}^h) \sum_{k=1}^C \delta_{uk}^o w_{kj}^o$ . This particular version of the proposed algorithm is available as the learning algorithm proposed by Pal *et al.* in [PM92]. Note that we are not considering Pal *et al.*'s algorithm with fuzzy linguistic inputs; rather we are considering it with crisp inputs.

For  $\sum_c \mu_c(\mathbf{y}_u) = 1 \forall u$  and  $q = 1$ , the learning equations (5.23-a) and (5.23-b) can be simplified to

$$\Delta w_{kj}^o = \eta [\mu_k(\mathbf{y}_u) - o_{uk}^o] o_{uj}^h \quad (5.34-a)$$

$$\Delta w_{ji}^h = \eta f_j^h(\text{net}_{uj}^h) y_{ui} \sum_{k=1}^C [\mu_k(\mathbf{y}_u) - o_{uk}^o] w_{kj}^o \quad (5.34-b)$$

This particular case of the learning algorithm is derivable from a variant of Pal *et al.*'s cross entropy [PM92], i.e.,  $\mathcal{H}_u^{\text{Pal}} = (\mu_k(\mathbf{y}_u) \ln(o_{uk}^o) + (1 - \mu_k(\mathbf{y}_u)) \ln(1 - o_{uk}^o))$ . This result is quite obvious as the definition of the fuzzy cross entropy reduces to Pal *et al.*'s cross entropy when  $q = 1$  and  $\sum_c \mu_c(\mathbf{y}_u) = 1 \forall u$ . This claim can be proved from the following Lemma.

**Lemma 5.5:**  $\mathcal{H}_u^f = \mathcal{H}_u^{\text{Pal}}$  when  $\sum_c \mu_c(\mathbf{y}_u) = 1 \forall u$  and  $q = 1$

**Proof.** When  $q = 1$  and  $\sum_{c=1}^C \mu_c(\mathbf{y}_u) = 1$ ,

$$\mathcal{H}_u^f = - \sum_{c=1}^C \mu_c(\mathbf{y}_u) \left[ \sum_{k=1}^C (t_{ck} \ln(o_{pk}^o) + (1 - t_{ck}) \ln(1 - o_{pk}^o)) \right] \quad (5.35)$$

$$= - \sum_{k=1}^C \left[ \sum_{c=1}^C \mu_c(\mathbf{y}_u) (t_{ck} \ln(o_{pk}^o) + (1 - t_{ck}) \ln(1 - o_{pk}^o)) \right] \quad (5.36)$$

$$= - \sum_{k=1}^C \left[ \mu_k(\mathbf{y}_u) (t_{kk} \ln(o_{pk}^o) + (1 - t_{kk}) \ln(1 - o_{pk}^o)) + \sum_{c=1, c \neq k}^C \mu_c(\mathbf{y}_u) (t_{ck} \ln(o_{pk}^o) + (1 - t_{ck}) \ln(1 - o_{pk}^o)) \right] \quad (5.37)$$

Since  $t_{kk} = 1$  and  $t_{ck} = 0 \forall c \neq k$ ,

$$\mathcal{H}_u^f = - \sum_{k=1}^C \left[ \mu_k(\mathbf{y}_u) \ln(o_{pk}^o) + \sum_{c=1, c \neq k}^C \mu_c(\mathbf{y}_u) \ln(1 - o_{pk}^o) \right] \quad (5.38)$$

In the case of constrained fuzzy approach,  $\mu_k(\mathbf{x}_u) + \sum_{c=1, c \neq k}^C \mu_c(\mathbf{y}_u) = 1$ , and hence,

$$\mathcal{H}_u^f = - \sum_{k=1}^C \left[ \mu_k(\mathbf{y}_u) \ln(o_{pk}^o) + (1 - \mu_k(\mathbf{y}_u)) \ln(1 - o_{pk}^o) \right] = \mathcal{H}_u^{\text{Pal}} \quad (5.39)$$

■

Thus, being possibilistic in nature, the proposed algorithm encapsulates various BP algorithms based on crisp as well as constrained fuzzy classification.

### 5.2.3 Testing of Feedforward Neural Networks

The network learns the fuzzy boundaries between the classes after training. In this stage, a separate set of test patterns is given as the inputs to the network. Generated outputs are the class membership values corresponding to the test inputs.

Note that, the network with the proposed learning algorithm is a universal approximator [HSW89].

#### 5.2.4 Results and Discussion

We employed the BP learning algorithms with the fuzzy mean square error (or fuzzy cross entropy) to train FFNNs for first, second and third level bids. The inputs are modified feature vectors. The number of input nodes for all the FFNNs are 52. The number of hidden nodes for all the FFNNs are chosen as 50. We have chosen the number of hidden nodes as 50 because we have observed in chapter 3 that the performance of the networks is good with 50 hidden nodes. The value of  $q$  is chosen as 1. The learning-rate is adaptively changed in the following way: If the error decreases during training, then the learning-rate is increased by a predefined amount. In contrast, if the error increases, then the learning-rate is decreased and the new weights and errors are discarded. As a result, the error always decreases or stays as it is. The momentum is kept 0.5 throughout the process. We adopted the strategy of picking the output node with the highest activation value as the output class corresponding to an input.

For the first level bids, the FFNN has 6 output nodes. We used the same training and test sets as we used in chapter 3 and 4 (i.e., “TrainingSet1” and “TestSet1”). While using the fuzzy mean square error, the convergence was achieved within 1570 iterations (Fig. 5.2Top). Using the fuzzy cross entropy, the network took 1400 iterations to converge (Fig. 5.2Bottom). The error values shown in Fig. 5.2Top and Bottom are the average of the error values with five different network initializations. From these figures, it appears that the convergence property of FFNNs with fuzzy cross entropy is slightly better than that of fuzzy mean square error. In chapter 4, we found that FFNNs with crisp BP converge within 1650 iterations. Therefore, the BP algorithm with fuzzy objective functions offers slight improvement in the convergence property for the first level bids. In the first row of Table 5.1, we have rewritten the classification performance of the conventional BP algorithm with crisp mean square error (from Table 4.3). Classification efficiency of the network with fuzzy objective functions is depicted in the second and third rows of Table 5.1. In this table, we can observe the better classification performance of the BP algorithm with fuzzy objective functions compared to the conventional BP algorithm. This improvement takes place because the proposed method takes care of the fuzziness involved in the classification from the possibilistic angle. The proposed algorithms can find the fuzzy decision boundary more accurately as some input patterns (especially, at the borders or away from the classes) may not satisfy the condition  $\sum_c \mu_c(\mathbf{x}_p) = 1$ . In Table 5.1, we can observe that the BP algorithm with the fuzzy mean square error is showing marginally better results compared to the fuzzy cross entropy. Therefore, training



**Table 5.1:** Classification performance of FFNNs with the BP algorithm for first level bids. The inputs are modified feature vectors. The symbols Obj. fn., cmse, fmse and fce imply objective function, crisp mean square error, fuzzy mean square error and fuzzy cross entropy, respectively.

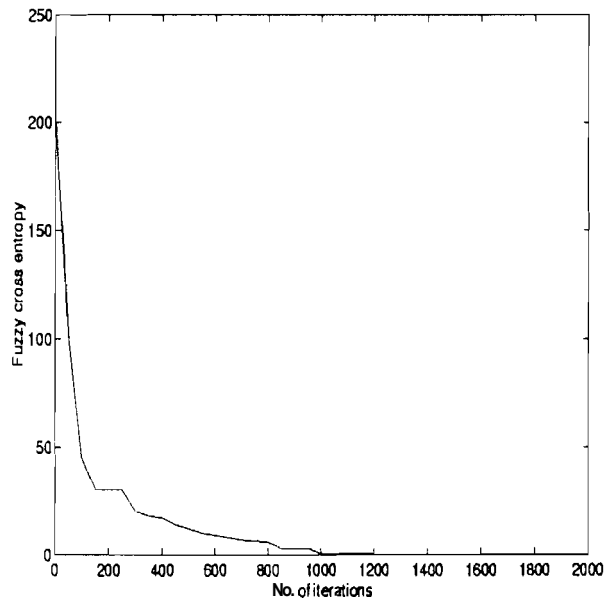
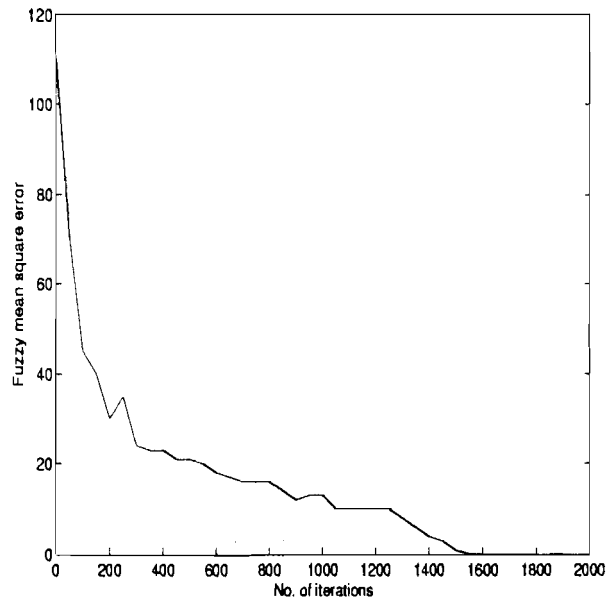
Obj. fn.	Pass	1C	1D	1S	1H	1N	Overall
cmse	<b>85.82%</b>	<b>69.94%</b>	<b>72.13%</b>	<b>70.01%</b>	<b>78.25%</b>	<b>69.92%</b>	<b>74.34%</b>
fmse	<b>77.14%</b>	<b>81.98%</b>	<b>71.03%</b>	<b>92.61%</b>	<b>59.33%</b>	<b>75.02%</b>	<b>76.18%</b>
fce	<b>87.31%</b>	<b>71.74%</b>	<b>82.12%</b>	<b>87.33%</b>	<b>59.27%</b>	<b>65.04%</b>	<b>75.46%</b>

**Table 5.2:** Classification performance of FFNNs with the BP algorithm for second level bids. The inputs are modified feature vectors. The symbols Obj. fn., cmse, fmse and fce imply objective function, crisp mean square error, fuzzy mean square error and fuzzy cross entropy, respectively.

Obj. fn.	2C	2D	2S	2H	2N	Overall
cmse	<b>67.45%</b>	<b>76.88%</b>	<b>78.19%</b>	<b>75.43%</b>	<b>74.98%</b>	<b>74.59%</b>
fmse	<b>71.01%</b>	<b>77.23%</b>	<b>78.57%</b>	<b>81.12%</b>	<b>74.17%</b>	<b>76.42%</b>
fce	<b>80.23%</b>	<b>75.72%</b>	<b>67.43%</b>	<b>80.55%</b>	<b>76.02%</b>	<b>75.99%</b>

an FFNN with the fuzzy cross entropy may be easier compared to the FFNN with the fuzzy mean square error (Fig. 5.2); but the generalization capability of the FFNN with the fuzzy mean square error is better than the FFNN with the fuzzy cross entropy.

In a similar manner, the proposed method trains an FFNN for the second level bids using "TrainingSet2". The inputs to the networks are the modified feature vectors. This network has 5 output nodes. The FFNN for the third level bids was trained by "Training Set1" and tested on "TestSet3". This network has 4 output nodes. The classification performance of these two FFNNs are given in Table 5.2 and 5.3. These tables show the improvements in the classification results of the proposed method compared to the BP algorithm with crisp objective functions. Note that FFNNs with the fuzzy mean square error are consistently performing better than FFNNs with the fuzzy cross entropy.



**Fig. 5.2:** Top: No. of iterations vs. fuzzy mean square error of an FFNN. Bottom: No. of iterations vs. fuzzy cross entropy of an FFNN. In both the cases, the FFNN has fifty-two input nodes, fifty hidden nodes and six output nodes. All training patterns are from first level bids.

**Table 5.3:** Classification performance of FFNNs with the BP algorithm for third level bids. The inputs are modified feature vectors. The symbols Obj. fn., cmse, fmse and fce imply objective function, crisp mean square error, fuzzy mean square error and fuzzy cross entropy, respectively.

Obj. fn.	3C	3D	3S	3H	Overall
cmse	85.14%	79.43%	82.46%	85.11%	83.03%
fmse	90.11%	84.51%	88.23%	86.15%	87.25%
fce	87.12%	87.36%	86.73%	82.66%	85.96%

### 5.3 Configuration of Feedforward Neural Networks Using Evolutionary Programming-Based Hybrid Technique

Use of FFNNs with fuzzy objective functions improves the classification performance of the modules. But the training process may be slow or in some cases it may halt due to the presence of local minima. Even if the network converges, the generalization capability of the trained network may not be high because of the improper choice of the network size. This section proposes a method to configure FFNNs in terms of optimum structure and optimum parameter set so that the resultant network generalizes well. The proposed method uses the BP algorithm with fuzzy objective functions as a local search operation. In addition, it employs evolutionary programming (EP) technique as a global search operation.

In many classification problems, it has been proved that learning in general, as well as choosing an optimal network configuration, are NP complete [Man93]. The selection of an appropriate number of hidden nodes and weights is so difficult because small number of hidden nodes and weights may hamper convergence of the network on a training set; on the other hand, large number of hidden nodes and weights may affect the *generalization* [Hay94] capability of the network. Large size of a network **affects** the generalization capability mainly in two ways. Firstly, the large size of a network may cause *overfitting* [Hay94], i.e., the network simply memorises the training patterns. Secondly, while training a large network, all the weights may not get involved in the training process as they balance each others effect on the output. Consequently, training error becomes **low**. However, such free weights may result in a large variation of the classification efficiency for different test sets [Sus92]. Other than the generalization issue, smaller networks are

better because they are usually faster and cheaper to build. Moreover, the operation of smaller networks is easier to understand where users need to know how the system works. But it is not always true that the smaller a network is, the better is its generalization capability. It is because, sometimes small networks may cause *underfitting* [Hay94] of the data. In addition, there may exist certain networks of optimum size just complex enough to generalize the data but very sensitive to the initial conditions [Ree93]. These two problems, i.e., underfitting and sensitiveness to the initial condition, may result in a low classification rate on the test sets, and thus these networks, although they are small, are not useful.

In order to have an optimal network architecture, we need an objective function and an advanced search and optimization method. The search method should necessarily look for the following:

1. How to determine the optimum number of hidden nodes of the network after avoiding locally optimal solutions.
2. How to determine the optimum set of weights and bias of the network after avoiding locally optimal solutions.
3. How to make the network configuration to be less sensitive to the initial choice of the weights and bias values.
4. How to reduce the configuration time.

In light of these requirements we can formulate an objective function, whose minimization will generate an optimum network configuration that generalizes well. The choice of the objective function should be such that minimization of it should not lead to memorisation of the input patterns. One such objective function in neural networks training can be the fuzzy mean square error or fuzzy cross entropy on a validation set. The validation set consists of a set of input-output pairs which do not occur in the training set.

A potential candidate for the optimization method is gradient descent algorithms [Hay94]. The advantages of the gradient descent algorithm are: (a) It uses the local information in an efficient way resulting in better accuracy, provided it does not get stuck in local optima or saddle points, and (b) it is quite fast to find the local optima or saddle points. The disadvantages of this method are: (a) It may stagnate at certain potentially suboptimal solutions, rendering the network incapable of sufficient performance, (b) it is sensitive to the initial values of the weights and bias, and (c) it cannot be used when the objective function is not differentiable at certain points. Another candidate for

the optimization process is EP [FOW66] [Fog91b] [Fog95], which is a stochastic search and optimization technique. EP optimizes the objective function by using a controlled stochastic search, and it performs the search parallelly from more than one point. In other words, while searching for the global minimum, this technique explores many paths simultaneously. Certain search paths may be less promising at the initial stage, whereas due to the random perturbation of the search parameters these search paths may become highly promising after some time. In the EP-based approach, these less promising solutions are kept along with highly promising solutions, hoping that they would lead new search paths towards the global minima after some time. These new paths enable the search process to avoid locally optimal solutions. Also, by adding or deleting hidden nodes or by small perturbation of the weights and bias terms, the search operation may jump over local minima. Due to these two reasons, EP can avoid a locally minimal solution, whereas the gradient-based approach cannot. In EP-based approach more than one solution is generated initially, and the solutions are repeatedly adapted by adding and deleting the hidden nodes, or by small perturbation of the weights or bias values. Thus the problem of proper initialization of the weights and bias values is also reduced. However, EP can suffer from extremely slow convergence before arriving at the correct solution. It is because EP does not exploit available local information [RF96]. Therefore, a clever approach is to go for a trade off where merits of both the methods, i.e., speed, accuracy, reliability and fast computation can be achieved. EP is good for *exploration* in the search space, whereas the BP is good for *exploitation*. Inspired by biology and especially by the manner in which living beings adapt themselves to their environment, the hybrid method adopted in this section involves two interwoven levels of optimization, namely evolution (EP) and individual learning (BP), which co-operate in a global process of optimization. The evolution of individuals are carried out to minimize certain global objective function. The global objective function is the fuzzy mean square error or fuzzy cross entropy on a validation set. The local search method, i.e., minimization of the fuzzy mean square error or fuzzy cross entropy on a training set, is used to guide the global search method [PIL96].

By fusing gradient descent and evolutionary algorithm, the search method becomes faster than a pure evolutionary approach. However there is a further scope to accelerate the proposed method by accelerating EP. Although EP is based on random search, it is not totally random – rather it is a controlled random search. This control action is provided by certain mutation parameters. The proper choice of mutation parameters has a profound impact on the convergence and performance of the proposed method. In accordance with this requirement, another issue that is addressed in this section is the

dynamic adaptation of the mutation parameters.

### 5.3.1 Evolutionary Programming in Network Configuration

While designing a feedforward network for a particular problem, the aim is to find the optimum number of hidden nodes and a set of optimum parameters. Formally, it can be written as a problem of finding the global maxima of the following function:

$$\mathcal{G}(\mathbf{y}, \theta) : \mathbb{R}^N \rightarrow \mathbb{R} \quad (5.40)$$

where  $\mathbf{y}$  represents a modified feature vector with dimension  $N$ ,  $\theta$  consists of weights and bias terms, and  $\mathcal{G}(\mathbf{y}, \theta)$  signifies how good the network for the particular classification problem is. In the proposed method,  $\mathcal{G}$  is maximised such that EP is able to find the optimum value of  $\theta$  as well as the optimum value for the dimension of  $\theta$ .

The above idea to configure a network is implemented through the following sequence of events (Fig. 5.3). Initially, EP creates a population of networks. EP initializes the population with fully connected networks of randomly (uniform distribution) generated hidden nodes. Thus,  $\nu$  such networks are formed with each network having any number of hidden nodes between one to some prespecified positive integer. The number of hidden nodes in each network is determined randomly from a uniform distribution. These networks are called *parents*. Each parent network is trained for a fixed number of iterations using the BP algorithm with the fuzzy mean square error or fuzzy cross entropy. *Fitness* value (a measure to indicate how good the network is for the given classification task) of each parent network is measured. Each parent is now allowed to create an offspring. Thus,  $\nu$  offspring networks are generated. The method to create the offspring is described in the next section. Each offspring network is trained for a fixed number of iterations using the BP algorithm with the fuzzy mean square error or fuzzy cross entropy. The *fitness* value of each offspring network is measured. Thus  $2\nu$  networks, comprising of parents as well as offsprings, are generated. Next in the competition phase, pairwise comparison of *fitness* values of all the networks (parents as well as offsprings) are conducted. For each solution, the algorithm chooses 10 randomly selected opponents from all the parents and offsprings with uniform probability. In each comparison, if the conditioned network offers as good performance as the randomly selected opponent, it receives a *win* [Fog95]. Based on the wins, networks scoring in the top 50% are designated as parents. All the networks, other than the parents, are discarded. Again, these parents create offsprings, and thus, the whole procedure is continued until the number of generations becomes greater than

some prespecified constant. Finally, the network with the highest fitness is considered as the desired network.

It is to be noted that while embedding the BP algorithm in the EP paradigm, we have employed *Lamarckian principle* [RF96]. In this case the properties learned by the individuals are transferred to the next generation. The life of each individual spans the number of iterations used in the BP algorithm.

### 5.3.2 Implementation Issues

#### 5.3.2.1 Fitness function

Fitness value of a network decides how good it is in the competition phase. Specifically, a network with higher fitness value has higher chance of survival and vice versa. The fitness function of a network is

$$\mathcal{G} = \frac{1}{E} \quad (5.41)$$

where  $E$  is equal to either  $\sum_u \mathcal{E}_u^f$  or  $\sum_u \mathcal{H}_u^f$  on a validation set. It is important to note that although  $\sum_u \mathcal{E}_u^f$  or  $\sum_u \mathcal{H}_u^f$  is differentiable with respect to the connection weights, it is nondifferentiable with respect to the number of hidden nodes. Thus, gradient-based optimization methods cannot be applied here to determine the optimal number of hidden nodes.

An alternative fitness function could be the inverse of *Akaike's information criterion* (AIC) [Aka74] [Fog91a] [BZ95] or *network information criterion* (NIC) [MYA94]. Since the AIC (or NIC) value is supposed to be used only after the network is completely evolved, AIC (or NIC) value calculated from a network which is not evolved completely may not reflect the generalization capability of the network at the current generation.

In order to generate offsprings, the following steps are needed:

#### 5.3.2.2 Replication of parents

In our work, each parent is typically represented combinedly by the number of hidden layers, number of input, output and hidden nodes, set of weight values and set of bias values. Since the number of hidden layers is one, and the number of input and output nodes are fixed, a network is actually represented by the number of hidden nodes, weight values and bias values. In this step, these values are copied from the parent to generate a new offspring.

```

Randomly generate a population of  $\nu$  networks (call them
parents).
FOR each parent
    Train the parent network for a fixed number of iterations
    by the BP algorithm with fuzzy objective functions.
    Find the fitness value of the parent network.
END FOR
WHILE (the number of generations is less than a specific number
or the fitness of the best parent is less than a specific value)
    FOR each parent network
        Create an offspring of the parent network.
        Train the offspring network for a fixed number of
        iterations by the BP algorithm with fuzzy objective
        functions.
        Find the fitness value of the offspring network.
    END FOR
    Competition starts among all parent and offspring networks
    based on the fitness values.
    Survival of the fittest networks (call them parents).
END WHILE
The parent network with the highest fitness is considered as the
desired network.

```

**Fig. 5.3:** Configuration of feedforward neural networks using evolutionary programming.



### 5.3.2.3 Mutation

The aim of creating offsprings is to minimize the global objective function. Basically creation of an offspring is searching one step forward or backward in the search space. But the length of a step size and the corresponding step direction are unknown. The step size cannot be too big as well as too small, because it may result the search process to jump over the global minimum or to take long time to reach the global minimum. It necessitates the use of mutation operators to decide the stepsize and step direction of the search method probabilistically. The nondeterminism associated with the selection of step size and step direction enables the search process to avoid local minima. To search an optimum set of weights and bias terms, we encounter local minima which we call parametric local minima, and to find an optimum number of hidden nodes we come across local minima which we call structural local minima. Parametric local minima and structural local minima are alleviated by parametric mutation and structural mutation, respectively.

In the parametric mutation, each weight  $w$  is perturbed using a Gaussian noise. Hence,  $w = w + \mathcal{N}(0, T)$ . The mutation step size  $\mathcal{N}(0, T)$  is a Gaussian random number with mean 0 and variance  $T$ . The intensity of the parametric mutation should be high when the fitness value of the parent is low and vice versa. It can be accomplished if we consider  $T$  of a particular network as its temperature, and define it as

$$T = \alpha \mathcal{U}(0, 1) \left[ \frac{\text{minimum fitness}}{\text{fitness of the network}} \right] \quad (5.42)$$

where  $\mathcal{U}(0, 1)$  is a uniform random number over the interval  $[0, 1]$  and  $\alpha$  is a constant ( $0 \leq \alpha \leq 1$ ). Obviously, the range of  $T$  lies in between 0 and 1. This temperature in fact determines how close the network is to the solution for the task [ASP94], and the amount of the parametric mutation is controlled depending on that. Like simulated annealing, the temperature is used to anneal the mutation parameters. Initially when the temperature is high, the mutation parameters are annealed quickly like coarse grains, and at low temperature they are annealed slowly like fine grains. Large mutations are needed to escape parametric local minima; but many times large mutations adversely affect the offspring's ability to perform better than its parent [ASP94]. Hence, to lessen the frequency of large parametric mutations, we have multiplied right hand side of Equation (5.42) by  $\alpha \mathcal{U}(0, 1)$ . In Equation (5.42) we need to know the minimum value of the fitness function. The maximum value of  $\mathcal{E}^f$  is  $0.5nC^2$ . Therefore, from Equation (5.41), the minimum fitness

corresponding to  $\mathcal{E}^f$  is

$$\mathcal{G}_{\min} = \frac{1}{0.5C^2n} \quad (5.43)$$

Similarly, the minimum fitness corresponding to  $\mathcal{H}^f$  is  $\mathcal{G}_{\min} = \frac{1}{nC^2 \ln 2}$ .

In the proposed method, the structural mutation is used to obtain an optimum number of hidden nodes after avoiding structural local minima. Using the structural mutation, hidden nodes are added or deleted during the creation of offsprings. Determination of the optimum number of hidden nodes can be considered as a search problem in a structure space where each point represents a particular network. If some performance index like fuzzy mean square error on the validation set is assigned to each network, then the performance levels of all possible networks form a surface in the structure space. Thus, determination of the optimum number of hidden nodes is equivalent to finding the lowest point on this surface. However this search operation becomes complicated as the surface has the following typical characteristics [Yao93] [MTH89]:

1. The surface is very large since the number of possible networks can be very high.
2. The surface is nondifferentiable as the number of hidden nodes and weights are discrete.
3. The surface is multimodal as performance of two networks with different number of hidden nodes and weights may be same.

Due to the structural mutation, sometimes one hidden node is added or deleted during the creation of offsprings. The specific instants of hidden node addition or deletion in a network depend upon the probability of the structural mutation, which further depends on the fitness of the network. The hidden node, which is added to or deleted from the network, is selected randomly (uniformly). The amount of structural mutation depends on the probability of the structural mutation ( $p_m$ ). Large value of  $p_m$  makes EP a purely random search algorithm, while some amount of mutation is needed to prevent the premature convergence of EP to suboptimal solution [SP94]. Therefore, a scheme is adopted here to change  $p_m$  adaptively.

The value of  $p_m$  is increased when the population tends to get stuck in local minima, and it is decreased when the population is scattered in the solution space. Let the average fitness value of the population and the maximum fitness value of the population be denoted by  $\mathcal{G}_{av}$  and  $\mathcal{G}_{\max}$ , respectively.  $(\mathcal{G}_{\max} - \mathcal{G}_{av})$  is likely to be less for a population that has converged to an optimal solution than that for a population scattered in the solution

space [SP94]. It can be expressed as

$$p_m \propto \frac{1}{\mathcal{G}_{\max} - \mathcal{G}_{\text{av}}} \quad (5.44)$$

In order to preserve the good solutions of the population,  $p_m$  of a network with lower fitness must be greater than  $p_m$  of a network with higher fitness. It results in the following relation:

$$p_m \propto (\mathcal{G}_{\max} - \mathcal{G}) \quad (5.45)$$

Accommodating Equation (5.44) and (5.45) simultaneously, we can write

$$p_m = k_m \frac{\mathcal{G}_{\max} - \mathcal{G}}{\mathcal{G}_{\max} - \mathcal{G}_{\text{av}}} \quad (5.46)$$

where  $k_m$  is a proportionality constant. From this relation, it appears that for solutions with subaverage fitness values, i.e.,  $\mathcal{G} < \mathcal{G}_{\text{av}}$ ,  $p_m$  may assume value larger than 1. In order to make  $p_m$  for the subaverage solutions always less than or equal to one, the expression for  $p_m$  is modified as

$$p_m = k_m \frac{\mathcal{G}_{\max} - \mathcal{G}}{\mathcal{G}_{\max} - \mathcal{G}_{\text{av}}} \quad \text{if } \mathcal{G} \geq \mathcal{G}_{\text{av}} \quad (5.47\text{-a})$$

$$= k_m \quad \text{if } \mathcal{G} < \mathcal{G}_{\text{av}} \quad (5.47\text{-b})$$

To keep  $p_m$  in  $[0, 1]$ ,  $k_m$  should be less than or equal to 1. In fact, solutions with fitness values less than or equal to  $\mathcal{G}_{\text{av}}$  should be disrupted completely. Hence, the value of  $k_m$  is taken as 0.5. This assignment always makes  $p_m$  of the best network zero. But the best network should also be allowed to undergo through the structural mutation process. Obviously, the amount of mutation for the best network should be the lowest. This observation modifies the above equations as

$$p_m = k_{m1} \frac{\mathcal{G}_{\max} - \mathcal{G}}{\mathcal{G}_{\max} - \mathcal{G}_{\text{av}}} + k_{m2} \quad \text{if } \mathcal{G} \geq \mathcal{G}_{\text{av}} \quad (5.48\text{-a})$$

$$= k_m \quad \text{if } \mathcal{G} < \mathcal{G}_{\text{av}} \quad (5.48\text{-b})$$

where  $k_{m1}$  and  $k_{m2}$  are two constants. After this modification,  $p_m$  for the above average networks increases linearly from  $k_{m2}$  to  $k_{m1} + k_{m2}$ .

In the above relations, we have considered the spread of the population through  $(\mathcal{G}_{\max} - \mathcal{G})$ ; but we did not consider whether the members of the population are diverse in the population space [LD95]. It may happen that the spread is high, but the diversity is low (Fig. 5.4(a)) and vice versa (Fig. 5.4(b)). We indeed seek both spread and diversity

should be high (Fig. 5.4(c)). Higher spread allows the search to be carried out in a wider space, and higher diversity allows uniform exploration in that space.

To take the diversity factor into account,  $p_m$  should be high when the diversity is low and vice versa. In order to measure the diversity of a population, the concept of probabilistic entropy [Hay94] can be used over the fitness values. To measure the entropy, the interval  $[\mathcal{G}_{\min}, \mathcal{G}_{\max}]$  is divided into  $L$  subintervals  $[\mathcal{G}_{\min} + jD, \mathcal{G}_{\min} + (j+1)D]$ , where  $j = 0, 1, 2, \dots, L-1$  and  $D = (\mathcal{G}_{\max} - \mathcal{G}_{\min})/L$ .  $\tilde{p}_j$  is defined as  $\frac{n_j}{2\nu}$ , where  $n_j$  is the number of members of the population in the  $j$ th interval and  $2\nu$  is the size of the population ( $\nu$  parents +  $\nu$  offsprings). With these introduced notations, the probabilistic entropy is defined as

$$\mathcal{EN} = \frac{1}{\ln(2\nu)} \sum_{i=0}^{L-1} \tilde{p}_i \ln \tilde{p}_i \quad (5.49)$$

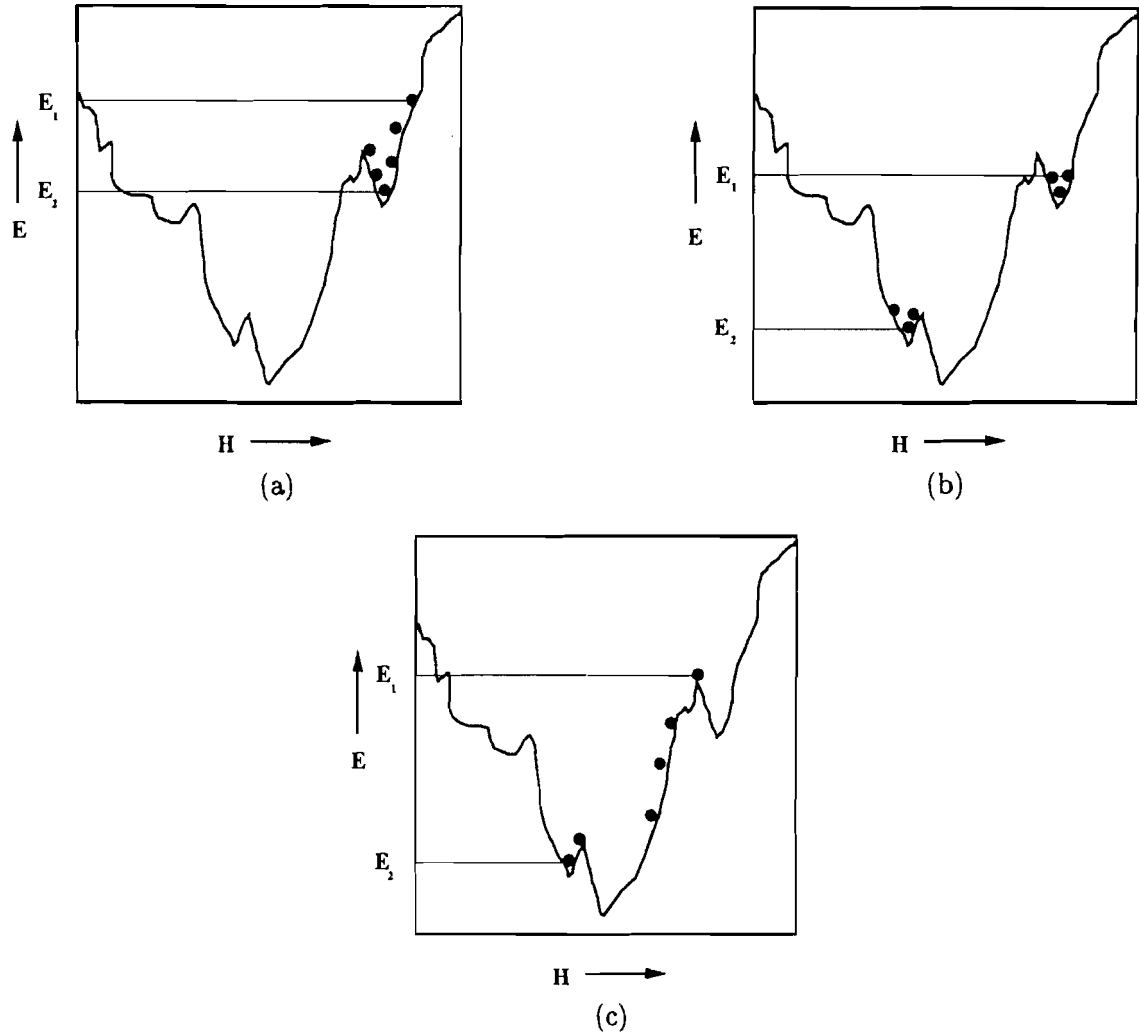
If the members are uniformly distributed, then  $\mathcal{EN}$  attains the maximum value 1. On the other hand, if all the members are grouped around a few values, the entropy value is close to 0. So, we can write,  $p_m \propto (1 - \mathcal{EN}^s)$  where  $s$  is a weighing factor. Combining this result with Equation (5.48-a and b), we can write

$$p_m = k_{m1} \frac{\mathcal{G}_{\max} - \mathcal{G}}{\mathcal{G}_{\max} - \mathcal{G}_{av}} (1 - \mathcal{EN}^s) + k_{m2} \quad \text{if } \mathcal{G} \geq \mathcal{G}_{av} \quad (5.50-a)$$

$$= k_m (1 - \mathcal{EN}^s) \quad \text{if } \mathcal{G} < \mathcal{G}_{av} \quad (5.50-b)$$

### 5.3.3 Results and Discussion

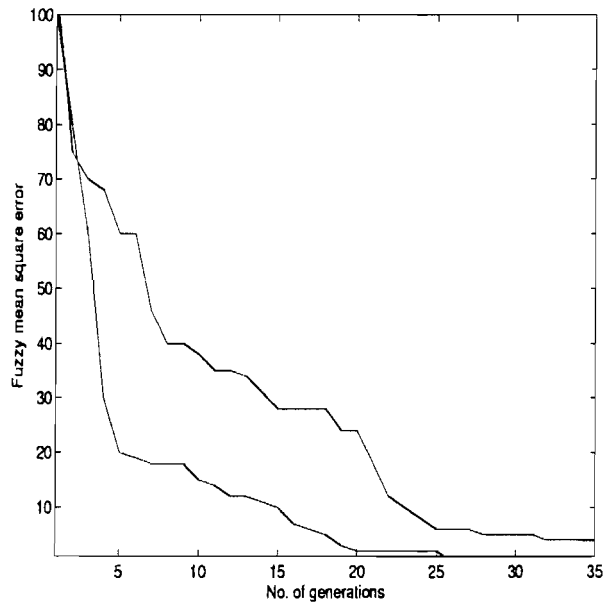
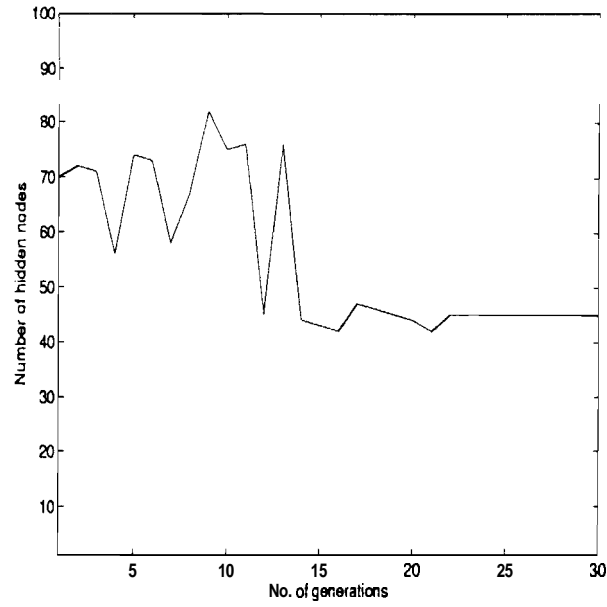
In section 5.2.4, we have already observed the classification performance of the BP algorithm with the proposed fuzzy objective functions. There we chose the number of hidden nodes arbitrarily. In the following experiments we choose the number of hidden nodes dynamically using the EP-based network configuration strategy. To configure the FFNN for first level bids, "TrainingSet1" was used. We first generated randomly 25 FFNNs (each with the number hidden nodes in between 1 and 100). In each generation, each parent network was trained for 100 iterations using the BP algorithm with the fuzzy mean square error. The fitness value of each parent was determined by calculating the fuzzy mean square error of the network for a validation set of size 200. For each parent, an offspring was created. During the structural mutation, between 1 to 3 hidden nodes were added or deleted at a time. The exact number of hidden node addition or deletion was decided randomly. If more number of hidden nodes are added or deleted, the fitness of



**Fig. 5.4:** Each circle represents a solution in the population space.  $E$  and  $H$  represent fitness and the number of hidden nodes in the solution, respectively. Three figures represent three different cases: (a) Spread of the population is low, but diversity is high, (b) spread of the population is high, but diversity is low, and (c) both spread and diversity are high.

the network decreases drastically. It is because the behavioural gap between the parent and the offspring becomes too high. To enhance the structural mutation, we used Equation (5.50-a and b). The value of  $s$  was chosen as 2.  $k_{m1}$  and  $k_{m2}$  were taken as 0.3 and 0.2 such that  $p_m$  varies linearly from 0.2 (for the best set) to 0.5 (for the average set). Fig. 5.5Top illustrates the number of hidden nodes in the best network for first level bids against the number of generations. The resultant network has 45 hidden nodes. This figure demonstrates the self-organization capability of the proposed algorithm, due to which, it is able to find better structure eventhough it starts with inappropriate number of hidden nodes. "Plot 1" in Fig. 5.5Bottom exhibits the fuzzy mean square error of the best networks, while trained by the proposed method. This error value is the average of the error values of the best network in ten runs. "Plot 1" is obtained when Equation (5.50-a and b) are used for adapting the parameters of structural mutation. "Plot 2" represents another curve when Equation (5.50-a and b) are not used for adapting the parameters of the structural mutation. This comparative study clearly demonstrates how effectively Equation (5.50-a and b) enhance the performance of the search process. Using "Plot 1" the convergence was achieved at the 20th generation. Since each generation needs 100 iterations for each parent network, the proposed method requires a long time on sequential computer. This drawback can be reduced if we use parallel machines. Actually this amenability to asynchronous parallel computation has made EP popular [BR94]. Classification efficiency of the FFNN on "TestSet1" is shown in the third row of Table 5.4. The fourth row of Table 5.4 depicts the classification performance of the EP-based method with the fuzzy cross entropy. The first and second rows of this table are reproduced from Table 5.1. Table 5.4 demonstrates the better generalization capability of the network while configured by the proposed method. We can observe that the FFNN trained with the fuzzy mean square error is giving better result (overall) compared to the FFNN with the fuzzy cross entropy.

Similarly the proposed technique was used for the second and third level bids. The number of hidden nodes of the FFNNs for the second and third level bids are 32 and 23, respectively. The classification results are illustrated in Table 5.5 and 5.6. We can observe that in the configured architectures, the network with the fuzzy mean square error shows better performance than that of fuzzy cross entropy. Hence, in the subsequent experiments with FFNNs, we shall use only the fuzzy mean square error.



**Fig. 5.5:** Top: No. of generations vs. fuzzy mean square error of an FFNN. Bottom: No. of generations vs. no. of hidden nodes. "Plot 1" represents a curve when Equation (5.50-a and b) are used for adapting the parameters of the structural mutation. In contrast, "Plot 2" represents another curve when Equation (5.50-a and b) are not used for adapting the parameters of the structural mutation.

Table 5.4: Classification performance of FFNNs with the BP algorithm for first level bids. The inputs are modified feature vectors. The symbols Arc., Obj. fn., F, D, fmse and fce imply architecture, objective function, fixed architecture, configured architecture, fuzzy mean square error and fuzzy cross entropy, respectively.

Arc.	Obj. fn.	Pass	1C	1D	1S	1H	1N	Overall
F	fmse	77.14%	81.98%	71.03%	92.61%	59.33%	75.02%	76.18%
F	fce	87.31%	71.74%	82.12%	87.33%	59.27%	65.04%	75.46%
C	fmse	80.64%	75.14%	79.38%	85.63%	70.44%	73.02%	77.37%
C	fce	72.90%	78.45%	81.23%	78.01%	78.81%	72.23%	77.60%

Table 5.5: Classification performance of FFNNs with the BP algorithm for second level bids. The inputs are modified feature vectors. The symbols Arc., Obj. fn., F, D, fmse and fce imply architecture, objective function, fixed architecture, configured architecture, fuzzy mean square error and fuzzy cross entropy, respectively.

Arc.	Obj. fn.	2C	2D	2S	2H	2N	Overall
F	fmse	71.01%	77.23%	78.57%	81.12%	74.17%	76.42%
F	fce	80.23%	75.72%	67.43%	80.55%	76.02%	75.99%
C	fmse	75.82%	77.57%	80.24%	78.12%	76.68%	77.68%
C	fce	78.13%	75.42%	81.23%	79.12%	74.12%	77.60%

Table 5.6: Classification performance of FFNNs with the BP algorithm for third level bids. The inputs are modified feature vectors. The symbols Arc., Obj. fn., F, D, fmse and fce imply architecture, objective function, fixed architecture, configured architecture, fuzzy mean square error and fuzzy cross entropy, respectively.

Arc.	Obj. In.	3C	3D	3S	3H	Overall
F	fmse	90.11%	84.51%	88.23%	86.15%	87.25%
F	fce	87.12%	87.36%	86.73%	82.66%	85.96%
C	fmse	89.23%	88.52%	93.15%	85.63%	89.13%
C	fce	82.18%	81.14%	93.12%	93.54%	87.49%

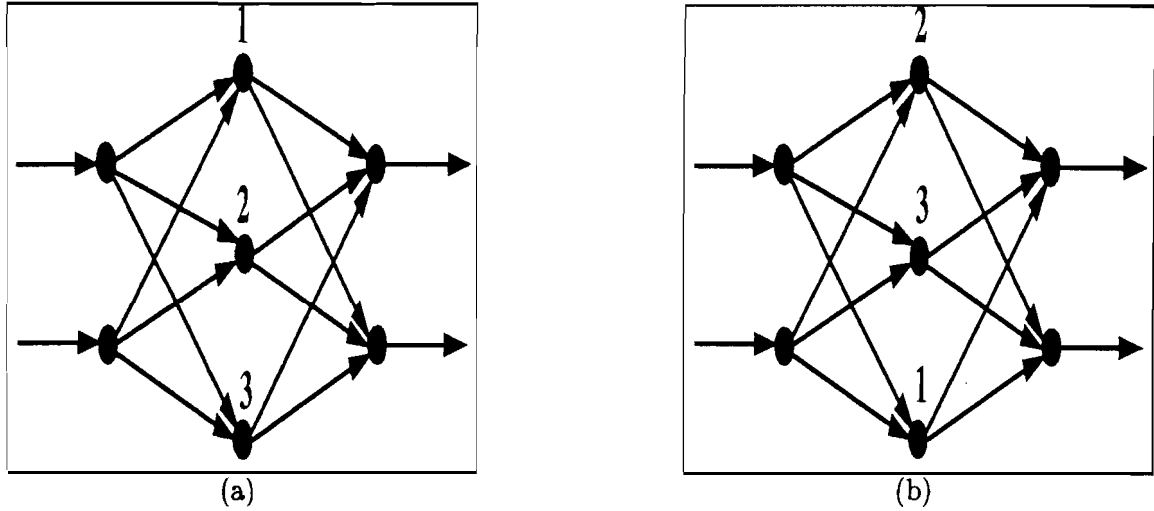


## 5.4 Summary

In this chapter, we applied feedforward neural networks to construct each module by capturing the relationship between the feature vectors and the output classes present in the module. Since the output bids are fuzzy, the network is trained by the BP learning algorithm with fuzzy objective functions. The proposed training algorithm has the possibilistic classification ability, and hence, it can encompass various BP learning algorithms based on crisp and constrained fuzzy classification. To increase the generalization capability of the network, we configure FFNNs using a hybrid search operation consisting of both deterministic and stochastic search operations. As a deterministic search, the proposed BP algorithm with fuzzy objective functions is used. As a stochastic search, EP is employed. The BP algorithm uses local information efficiently, whereas EP exploits global information. The efficiency of the whole search process is further enhanced by dynamic adaptation of the structural mutation. If a modified feature vector is presented to the configured network, the output of the network is produced as class membership values corresponding to the input pattern.

As a global search method, in place of EP, we could have chosen constructive and destructive pruning techniques [Ree93], [SM93]. Constructive pruning techniques [SST93] initially assume a simple network, and add nodes and links as warranted, while destructive techniques [MS89a] start with a large network and prune off superfluous components. The aim of the pruning techniques is to evolve a near optimal neural network architecture. However the problems associated with the pruning techniques are [ASP94]: (a) These methods get stuck in local minima very easily. (b) In these methods, once an architecture is explored and determined to be insufficient, the old one becomes topologically unreachable. Thus, they investigate only restrictive topological subsets of networks rather than the complete class of network architectures [ASP94].

While configuring neural networks, EP is considered to be more powerful optimization tool than simulated annealing [KJV83] and genetic algorithm [Gol89], [Mic92], [Dav91]. In particular, simulated annealing is a sequential search operation, whereas EP is a parallel search algorithm. In fact, we can say that EP is more than a parallel search algorithm. Parallel search starts with a number of different paths (say  $\nu$  where  $\nu > 1$ ) and continues until all the search paths get stuck in blind alleys or any one of them finds the solution. EP also starts with  $\nu$  different paths. But, it tries to generate new paths which are always better than the current paths. Due to this inherent parallelism, in many cases EP-based



**Fig. 5.6:** (a) and (b) are two equivalent networks, which order their hidden nodes differently. The genotype representations of the networks become different, although the networks are equivalent.

search operation becomes more efficient and faster than simulated annealing-based operation [PFF95]. Although both EP and genetic algorithm are parallel search operations, the EP-based optimization approach is more attractive for the network configuration. It is due to the following reasons:

1. EP manipulates networks directly. So it does not need any dual representation. Genetic algorithm needs coding which may not represent the problem itself [ASP94].
2. While creating offsprings, EP avoids recombination between networks. It helps to keep the individuality of the network intact [ASP94].
3. One major problem with genetic algorithm-based approach is permutation problem [Yao93]. The permutation problem stems from the fact that in genetic algorithm two functionally identical networks which label their hidden nodes differently (Fig. 5.4) will have two different genotype representations. Therefore, the probability of producing a highly fit offspring from them by crossover will be very low. EP-based optimization method does not suffer from this problem.
4. Asymptotic convergence property of EP is better than that of genetic algorithm [Fog94a], [FS93].

The EP-based network configuration technique can also be seen from Markov-chain

perspective [Fog95]. Each state of the Markov chain consists of all possible networks with the same fitness value. Since the fitness representation is finite on digital computers, the number of states is finite. The starting state depends on the initialisation. The state with the highest fitness acts as an absorbing state. The probability of jumping from one state to another state is dictated by the probability of mutation.

In the next chapter, we will employ clustering to capture the relationship between the modified feature vectors and the output class labels.

## Chapter 6

# DESIGN OF CLASSIFIER MODULES THROUGH CLUSTERING

### 6.1 Introduction

This chapter proposes a classifier module that uses clustering to capture the relationship between the modified feature vectors and the output classes of a module. Construction of such a classifier can be carried out in two phases. First phase is necessary to perform clustering, and the second phase is needed to establish the relationship between each cluster and the class labels. When a modified feature vector is presented as an input, the classifier detects the belonginness of the input into the clusters. The output class label corresponding to the pattern is determined depending on the relationship between each cluster and the output classes. In the opening bid problem, the clusters generated by the feature vectors are generally overlapping or fuzzy. In addition, the class labels of the patterns from the same cluster may not be similar. This *one-to-many* relationship between the clusters and the output class labels creates rough uncertainty. This chapter proposes a classification technique in presence of fuzzy and rough uncertainties.

It is possible to use the conventional fuzzy *K-means* (FKM) clustering algorithm to cluster the modified feature space. However, to apply the FKM user has to know *a priori* the number of clusters present in the given set of input patterns. Moreover, the solution obtained from the FKM may be locally optimal or too much dependent on the initializations. To reduce some of these limitations, in section 6.2, an evolutionary programming-based fuzzy clustering algorithm is proposed. This algorithm effectively groups a given set of input patterns into an optimum number of clusters. The algorithm determines the number of clusters and the cluster centers in such a way that there is a high chance of avoiding locally optimal solutions. The clustering results of the algorithm do not depend critically on the choice of the initial cluster centers.

After clustering, the next task is to label each cluster with an appropriate class label.

The main assumption of the clustering-based classification is that *similar inputs produce similar outputs*. It means that any two input patterns from the same cluster must be from the same class. Generalization is possible in such classifiers due to this similarity property. In the bidding problem, however, two patterns from the same cluster may belong to different classes, and hence, classification based on mere similarity property is inadequate. This problem arises because the available features are not sufficient to discriminate the classes. It implies that the fuzzy clusters generated by the modified feature vectors have rough uncertainty. To exploit the fuzziness and roughness, section 6.3 proposes *fuzzy-rough neural networks*. For any modified feature vector, the network determines the classification result in terms of *fuzzy-rough membership* values.

## 6.2 Evolutionary Programming-Based **Fuzzy** Clustering

Clustering a set of patterns provides a systematic approach for partitioning the set of patterns into different groups such that patterns with similar features are grouped together, and patterns with different features are placed in different groups [DJ87]. Formally, clustering can be defined as follows: [Bez81]: Given a set  $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$  of feature vectors, find an integer  $K$  ( $2 \leq K < n$ ) and the  $K$  partitions of  $\mathcal{Y}$  which exhibit categorically homogeneous subsets. An important requirement for resolving this issue is a suitable measure of clusters – what clustering criterion should be used? Specifically, what mathematical properties – e.g., distance, angle, curvature – possessed by the members of the data should be used, and in what way, to identify the clusters in  $\mathcal{Y}$ ? In fact, each observation may have infinite number of variations. In addition, the data set may be a mixture of different shapes, sizes and geometries. Therefore, infinite varieties of structures are possible. It is evident that clustering criterion must be problem-specific, and it cannot be universally applicable. Three types of clustering approaches are commonly used [Bez81]. They are (1) hierarchical approach, (2) graph theoretic approach and (3) objective function-based approach. Among them, the objective function approach is well-known. One extensively used objective function type clustering algorithm is *hard K-means algorithm* [TG74] [Bez81]. It involves assigning each pattern exactly to one of the clusters, assuming well-defined boundaries between the clusters. It is used for clustering where clusters are crisp and spherical. In the hard K-means algorithm, clustering is based on minimization of the overall sum of the squared errors between each pattern and the

corresponding cluster center. That is

$$E^c = \sum_{j=1}^n d(\mathbf{y}_j, \mathbf{m}_k)^2 \quad (6.1)$$

Here,  $K$  is the number of clusters and  $\mathbf{m}_k$  is the closest cluster center to the pattern  $\mathbf{y}_j$ . In real life situations, boundaries between groups may be overlapping. In particular, there may be some patterns that completely belong to one cluster, but partially belong to other clusters also. In order to overcome this problem, the idea of fuzzy K-means (FKM) algorithm has been introduced [Bez81]. Incorporation of fuzzy theory in the FKM algorithm makes it a generalized version of the hard K-means algorithm.

In the FKM, clustering is based on minimization of the overall weighted sum of squared error between each pattern and each cluster center, where the weight signifies the level of belongingness of the pattern into the cluster. It can be treated as minimization of the following objective function:

$$E^f(U, \mathbf{m}) = \sum_{j=1}^n \sum_{k=1}^K (\mu_k(\mathbf{y}_j))^q (d_{jk})^2 \quad (6.2)$$

where  $U = [\mu_k(\mathbf{y}_j)]$  is a fuzzy partition of  $\mathcal{Y}$  and  $\mathbf{m} = \{\mathbf{m}_1, \dots, \mathbf{m}_K\}$ , with  $\mathbf{m}_k$  designating the center of the cluster  $F_k$ . In this equation,  $q \in (1, \infty)$  and  $d_{jk}$  is a distance measure between  $\mathbf{y}_j$  and  $\mathbf{m}_k$ . Although the FKM algorithm is extensively used in literature [DJ87] [SS91], it suffers from several drawbacks. Firstly, to apply the algorithm, the user has to know *a priori* the number of clusters present in the given input data set. Secondly, the objective function is not convex, and hence, it may contain local minima. Therefore, while minimizing the objective function, there is a chance of getting stuck in local minima (also in local maxima and saddle points). Finally, the performance of the FKM algorithm depends on the choice of the initial cluster centers.

In this section we propose a clustering algorithm to address the following issues:

1. How to determine the optimum number of clusters.
2. How to avoid local minima solutions.
3. How to make the clustering less dependent on the initial choice of the cluster centers.

Since human ability to cluster data is far superior to any of the clustering algorithms, we examine some of the aspects of human way of clustering to address the above issues. For example, when we see a picture, we try to cluster the elements of the picture into different groups. It is interesting to note that, immediately after observing a picture

we can find how many clusters there are, and it is done without looking at each point within the clusters. It appears that clustering depends on the global view of the observer. After deciding the number of clusters, we try to see which point belongs to which cluster. Hence, we gather global information first, and then we look for local properties. Now the question is, what criterion do we use to gather the global information? Possibly we collect this global information from the isolation and compactness of the clusters in the whole picture. Although the FKM considers the local properties of the picture, it does not take the global view into its account.

We propose a clustering algorithm that tries to mimic the above mentioned features of the human way of clustering. In this algorithm, two objective functions are minimized simultaneously. The global view of the input data set is considered by an objective function called *fuzzy hypervolume* [GG89]. Minimization of this objective function takes place by randomly merging and splitting the clusters. The objective function  $E^f$  (given in Equation (6.2)) is minimized to consider the local property, i.e., to determine which input pattern should belong to which cluster. It turns out that minimization of the global performance index, i.e., the fuzzy hypervolume, gives the optimum number of clusters, whereas minimization of  $E^f$  leads to proper positioning of the cluster centers. In other words, the task of minimizing the fuzzy hypervolume can be considered **as** a major one, while the task of minimizing  $E^f$  can be regarded **as** a minor one. The role played by the fuzzy hypervolume and  $E^f$  is quite similar to the role played by the fuzzy mean square error on a validation set and the fuzzy mean square error on a training set (see configuration of FFNNs in section 5.3). Minimization of both the objective functions may yield locally optimal solutions. To circumvent the local minima problem, we propose an optimization technique based on evolutionary programming (EP) [Fog95]. EP-based search operation tries to escape locally minimal solutions by splitting and merging the clusters or by small perturbation of the cluster centers. In this approach, more than one solution is generated, and the solutions are repeatedly adapted by splitting and merging the clusters or by small perturbations of the cluster centers. Therefore, the initial choice of the cluster centers is not very critical in the proposed EP-based clustering algorithm.

### 6.2.1 Background of Fuzzy K-Means Clustering

The fuzzy K-means algorithm uses iterative optimization procedure to minimize the objective function  $E^f(U, m)$  (given in Equation (6.2)). This objective function is minimized such that the following constraints are satisfied.

$$(i) \mu_k(\mathbf{y}_j) \in [0, 1] \forall j, k; (ii) \sum_{j=1}^n \mu_k(\mathbf{y}_j) > 0 \forall k; (iii) \sum_{k=1}^K \mu_k(\mathbf{y}_j) = 1 \forall j; \quad (6.3)$$

The steps of the algorithm are stated in Fig. 6.1. The FKM algorithm can be made more powerful by using *fuzzy modification of maximum likelihood estimation* (FMLE) [GG89]. The intention of using the FMLE is to obtain better clustering results [GG89] when it is applied after using the FKM. Other than using different form of the distance measure  $d_{jk}$ , the steps of the FMLE algorithm are exactly similar to that of the FKM algorithm. The distance function used in Equation (6.7) is modified here as follows:

$$d_{jk} = \frac{[\det(F_k)]^{\frac{1}{2}}}{P_k} \exp \left[ (\mathbf{y}_j - \mathbf{m}_k)' (\Sigma_k^f)^{-1} (\mathbf{y}_j - \mathbf{m}_k) \right] \quad (6.4)$$

where

$$P_k = \frac{1}{n} \sum_{j=1}^n \mu_k(\mathbf{y}_j) \quad (6.5)$$

and  $\Sigma_k^f$  is the fuzzy covariance matrix for the cluster  $F_k$ .  $\Sigma_k^f$  is defined as [GG89]

$$\Sigma_k^f = \frac{\sum_{j=1}^n (\mu_k(\mathbf{y}_j))^q \delta_{kj} \delta_{kj}'}{\sum_{j=1}^n (\mu_k(\mathbf{y}_j))^q} \quad (6.6)$$

where  $\delta_{kj} = \mathbf{y}_j - \mathbf{m}_k$ ,  $q = (1, \infty)$ ,  $\mathbf{m}_k$  is the cluster center of  $F_k$  and  $\mu_k(\mathbf{y}_j)$  is the fuzzy membership of  $\mathbf{y}_j$  in  $F_k$ .



1. Fix the value of  $q$  and assign the **number** of clusters as  $K$ .  
Define a distance measure between  $\mathbf{y}_j$  and  $\mathbf{m}_k$  as

$$d_{kj} = (\mathbf{y}_j - \mathbf{m}_k)' \Sigma^{-1} (\mathbf{y}_j - \mathbf{m}_k) \quad (6.7)$$

where  $\Sigma$  is a positive definite matrix.

2. Assign  $i = 0$ .
3. Initiate the fuzzy  $K$ -partition  $U^i$ .
4. DO

(a) Set  $i = i + 1$ .

(b) Calculate  $K$  cluster centers  $\{\mathbf{m}_k\}$  of  $U^i$ :

$$\mathbf{m}_k = \frac{\sum_{j=1}^n (\mu_k(\mathbf{y}_j))^q \mathbf{y}_j}{\sum_{j=1}^n (\mu_k(\mathbf{y}_j))^q} \quad k = 1, 2, \dots, K \quad (6.8)$$

(c) Update  $U^{(i+1)}$  by calculating  $U^i$  as follows:

i. Determine the content of the following sets:

$$I_k = \{k \mid 1 \leq k \leq K; d_{kj} = 0\} \quad (6.9)$$

$$\bar{I}_k = \{1, 2, \dots, K\} - I_k \quad (6.10)$$

ii. Compute the new membership values as follows:

$$\text{If } I_k = 0 \quad \mu_k(\mathbf{y}_j) = \frac{1}{\sum_{h=1}^K \left( \frac{d_{kj}}{d_{hj}} \right)^{2/(q-1)}} \quad (6.11)$$

$$\text{else } \mu_k(\mathbf{y}_j) = 0 \quad \forall k \in \bar{I}_k \text{ and } \sum_{k \in I_k} \mu_k(\mathbf{y}_j) = 1 \quad (6.12)$$

END DO UNTIL  $\text{norm}(U^i - U^{(i+1)}) > \eta$

**Fig. 6.1:** Fuzzy  $K$ -means algorithm. Here  $\eta$  is a constant with small value and  $\text{norm}()$  is an appropriate matrix norm.

### 6.2.2 Embedding Evolutionary Programming in **Fuzzy** Clustering

The objective of the proposed clustering algorithm is to find the optimum number of clusters and the optimum position of each cluster center. Formally, it can be treated as the problem of finding the global maximum of the following function:

$$\mathcal{G}(\mathbf{z}) : \mathbb{R}^{nK} \rightarrow \mathbb{R} \quad (6.13)$$

where  $\mathbf{z}$  is an  $nK$  dimensional vector representing  $[\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_K]$  and  $\mathcal{G}(\mathbf{z})$  signifies how good the clustering is. Therefore, EP should be able to find the optimum value of  $\mathbf{z}$  as well as the optimum value of  $K$ .

Now we describe how the above idea can be used in a practical situation (see Fig. 6.2). To cluster an input data set, initially EP needs to create a population of sets of clusters. EP initializes the population using sets of clusters with randomly generated (uniform distribution) cluster centers. Thus  $\nu$  such sets of cluster centers are formed. Each set has any number of cluster centers between two and some prespecified positive integer. The number of cluster centers in each set is determined randomly. These sets are called *parents*. *Modified fuzzy K-means* (MFKM) algorithm clusters the entire data set using the set of parent cluster centers. The MFKM algorithm is described in the next section. A *fitness* value is assigned on each parent set. Each parent is allowed to create one offspring. Thus,  $\nu$  offspring sets of cluster centers are generated. The method of creating the offsprings is described in the next section. The MFKM clusters the entire data set using the set of offspring cluster centers, and then the fitness value of each offspring set is measured. As a result, we obtain  $2\nu$  sets of clusters comprising of parents as well as offsprings. Now the competition phase starts. In this phase, the fitness values of all sets (parents as well as offspring) are compared. For each solution, the algorithm chooses 10 randomly selected opponents from all parents and offsprings with uniform probability. In each comparison, if the conditioned set offers as good performance as the randomly selected opponent, it receives a *win* [PFF95], [SF95]. Based on the wins, sets scoring in the top 50% are designated as parents. All other sets are discarded. Again these parents are used to create offsprings. The whole procedure is continued until the number of generations becomes larger than some prespecified constant. Finally, the set with the maximum fitness value is considered as the desired clustered output.

```

Randomly generate a population of  $\nu$  sets of cluster centers
(call them parents).
FOR each parent
    Cluster the parent set using the MFKM.
    Find the fitness value of the parent set.
END FOR
WHILE (the number of generations is less than a specific number
or the fitness of the best parent is less than a specific value)
    FOR each parent set
        Create an offspring of the parent set.
        Cluster the offspring set using the MFKM
        Find the fitness value of the offspring set.
    END FOR
    Based on the fitness values competition starts among all
    parent sets and offspring sets.
    Survival of the fittest sets (call them parents).
END WHILE
The parent with the highest fitness is considered as the desired
set of clusters.

```

**Fig. 6.2:** The proposed evolutionary programming-based fuzzy clustering algorithm.

### 6.2.3 Implementation Issues

#### 6.2.3.1 Fitness function

In our work the following fitness function is chosen:

$$\text{fitness value} = \frac{1}{\text{total fuzzy hypervolume}} \quad (6.14)$$

where total fuzzy hypervolume ( $V \geq 0$ ) is an index to signify how good the clustering is. The smaller is the total fuzzy hypervolume [GG89], [KNF92], the better is the clustering. Since  $V$  may have any positive value, it appears from Equation (6.14) that the fitness value may be more than one. It is not objectionable as the fitness value is used here for relative comparison only.

The fuzzy hypervolume [GG89] of the cluster  $F_k$  is given by

$$V_k = \sqrt{\det(F_k)} \quad (6.15)$$

The total fuzzy hypervolume, occupied by all the clusters, is defined as

$$V = \sum_{k=1}^K V_k \quad (6.16)$$

Note that we have two objective functions  $E^f$  (in Equation (6.2)) and the total fuzzy hypervolume (in Equation (6.16)) to minimize. Of these two, we are treating only the inverse of the fuzzy hypervolume as the fitness function. The reason is that the evaluation of  $E^f$  in Equation (6.2) requires  $K$  to be predefined and fixed. When  $K$  varies,  $E^f$  for a set with the optimal number of clusters may not attain the minimum value. For example, if the number of clusters of a set is very close to the number of data, then the value of  $E^f$  is close to zero. Obviously, this kind of situation may not signify optimal clustering. Instead of minimizing both objective functions, we could have minimized only the fuzzy hypervolume. But, our search for a better set of clusters becomes more efficient when minimization of  $E^f$  is viewed as a clue to minimize the fuzzy hypervolume. In other words, the fuzzy hypervolume and  $E^f$  are used for *exploration* and *exploitation* in the search space, respectively [RF96].

The next three sections describe the three steps to generate the offsprings:

#### 6.2.3.2 Replication of parents

In the first step, each parent is represented by the number of clusters and cluster centers. In this step these values are copied from the parent to generate a new offspring.

### 6.2.3.3 Mutation

The aim of creating offsprings is to minimize Ef and the fuzzy hypervolume. To minimize  $E^f$ , we come across parametric local minima, and to minimize the fuzzy hypervolume we encounter structural local minima. Parametric local minima and structural local minima are overcome by the parametric mutation and structural mutation, respectively. Using the parametric mutation, each cluster center  $\mathbf{m}_k$ ,  $1 \leq k \leq K$ , is perturbed with Gaussian noise. It can be expressed as

$$\mathbf{m}_k = \mathbf{m}_k + \mathcal{N}(0, T) \quad (6.17)$$

Specifically, the mutation step size  $\mathcal{N}(0, T)$  is a Gaussian random vector with each component having mean 0 and variance T.

The intensity of the parametric mutation should be high when the fitness value of the parent is low and vice versa. It can be accomplished if T is defined for the parent set as

$$T = \alpha \mathcal{U}(0, 1) \left[ \frac{\text{minimum fitness}}{\text{fitness of the set of clusters}} \right] \quad (6.18)$$

where  $\mathcal{U}(0, 1)$  is a uniform random variable over the interval  $[0, 1]$  and  $\alpha$  is a constant ( $\alpha \leq 1$ ). Actually, this equation is already used in Equation (5.42).

The minimum value of fitness function is determined as follows: The fuzzy hypervolume of each cluster is always less than the crisp hypervolume of the cluster comprising of all the input patterns. Hence, we can write

$$V_k < \det \left[ \frac{\sum_{j=1}^n \delta_{0j} \delta_{0j}}{n} \right] \quad (6.19)$$

where  $\delta_{0j} = \mathbf{y}_j - \mathbf{m}_0$  and  $\mathbf{m}_0 = \frac{1}{n} \sum_{j=1}^n \mathbf{y}_j$ . Since  $\mathbf{V} = \sum_{k=1}^K V_k$ , the upper bound for the total fuzzy hypervolume V is

$$K \det \left[ \frac{\sum_{j=1}^n \delta_{0j} \delta_{0j}}{n} \right] \quad (6.20)$$

Therefore, the minimum fitness value is given by

$$\frac{1}{K \det \left[ \frac{\sum_{j=1}^n \delta_{0j} \delta_{0j}}{n} \right]} \quad (6.21)$$

The structural mutation is used to avoid structural local minima and to obtain the optimum number of clusters. The determination of the optimum number of clusters can be considered as a search problem in a structure space where each point represents a

particular set of clusters. If a performance index like fuzzy hypervolume is assigned to each set of clusters, the performance level of all possible sets of clusters forms a surface in the structure space. Thus, determination of the optimum number of clusters is equivalent to finding the lowest point on this surface. However, this search operation becomes complicated as the surface has the following characteristics [Yao93] [MTH89]:

1. The surface is very large since the number of possible sets of clusters can be very high.
2. The surface is nondifferentiable as the change in the number of clusters is discrete.

In order to find the proper number of clusters, i.e., to find the global minimum in the structure space, sometimes one cluster is added to or deleted from an offspring [TG74]. These addition and deletion operations are controlled by the structural mutation. The addition of one cluster to an offspring set is done by splitting an existing cluster of the offspring. To identify a cluster for splitting, it is required to find the cluster (say  $F_k$ ) with the maximum fuzzy hypervolume  $V_k$ . In order to break this cluster into two parts, the center of this cluster, i.e.,  $\mathbf{m}_k$ , is split into two new cluster centers  $\mathbf{m}_k^+$  and  $\mathbf{m}_k^-$ , and then  $\mathbf{m}_k$  is deleted [TG74]. As a result, the number of clusters for this set, i.e.,  $K$  is incremented by one. Here, the cluster center  $\mathbf{m}_k^+$  is formed by adding a certain quantity  $\gamma_k$  to the component of  $\mathbf{m}_k$  which corresponds to the maximum component of  $\sigma_k$  (variance of the  $k$ th cluster), i.e.,  $\sigma_{k_{max}}$ ; and in a similar way  $\mathbf{m}_k^-$  is formed by subtracting  $\gamma_k$  from the same component of  $\mathbf{m}_k$ . One simple way of specifying  $\gamma_k$  is to make it equal to some fraction of  $\sigma_{k_{max}}$ , that is

$$\gamma_k = \alpha \sigma_{k_{max}} \quad \text{where } 0 < \alpha \leq 1 \quad (6.22)$$

Deletion of one cluster from an offspring set is executed by merging two existing clusters of the set. In order to accomplish it, the two closest clusters with centers  $\mathbf{m}_{k_1}$  and  $\mathbf{m}_{k_2}$  are identified for merging. Thereafter, these two clusters are merged by a lumping operation as  $\mathbf{m}_k^* = \frac{1}{n_{k_1} + n_{k_2}} [n_{k_1} \mathbf{m}_{k_1} + n_{k_2} \mathbf{m}_{k_2}]$ , where  $\mathbf{m}_k^*$  is the center of the new cluster and  $n_{k_1}$  is the number of patterns in the cluster with center  $\mathbf{m}_{k_1}$ . Next,  $\mathbf{m}_{k_1}$  and  $\mathbf{m}_{k_2}$  are deleted, and the number of clusters  $K$  is reduced by one. The amount of the structural mutation can be adaptively controlled using Equation (5.50-a and b).

It is important to note that the splitting and merging operations employed in the proposed scheme are quite similar to that of in ISODATA [TG74]. However, unlike in ISODATA, here cluster merging and splitting are executed in a nondeterministic fashion. This inherent nondeterministic property plays a key role in avoiding local minima while

finding the optimum number of clusters, and eventually it guarantees the asymptotic convergence of the EP-based fuzzy clustering scheme towards the global minimum [Fog94a].

#### 6.2.3.4 Modified fuzzy K-means algorithm

By exploiting the mutation in a particular offspring, we obtain the number of clusters and the perturbed cluster centers. However, to calculate the fitness value of this offspring, the input data set needs to be clustered using the perturbed cluster centers. In addition, if the perturbed cluster centers are updated based on the clustered output, then the minimization of  $E^f$  takes place, and as a result, the minimization of the fuzzy hypervolume becomes easy. We exploit the modified fuzzy K-means (MFKM) algorithm to accomplish this task. For an offspring, the MFKM is executed for a certain number of iterations (say  $j$ ) at each generation. Consequently, if the offspring survives  $g$  generations, then it passes through  $g \cdot j$  iterations. The steps associated with the MFKM algorithm are described in Fig. 6.3.

The MFKM algorithm basically remembers the cluster centers at the last generation, and updates the old cluster centers in the current generation. This updating process, however, may get stuck in certain parametric local minima. In order to avoid it, the cluster centers of the offspring at the last generation are perturbed by applying Equation (6.17), and then the cluster centers are used in the current generation for further updating. Although both MFKM and FKM are iterative in nature, the difference between them is that the FKM never uses the old cluster centers in perturbed form. This difference makes the FKM algorithm a deterministic search operation, and thus vulnerable for the parametric local minima.

#### 6.2.4 Results and Discussion

Before using the proposed clustering technique on the opening bid problem, we show the performance of the proposed clustering technique on an artificially generated simple data set. For the sake of visual observation, the dimension of each data is taken as two. We generated 387 data from 9 Gaussian distributions (Fig. 6.4Top). The value of  $\nu$  is set to 4. To enhance the parametric mutation, we used Equation (5.50-a and b). The values of  $q$  and  $s$  are taken as 2 and 2, respectively.  $k_{m1}$  and  $k_{m2}$  are taken as 0.3 and 0.2 such that  $p_m$  varies linearly from 0.2 (for the best set) to 0.5 (for the average set). During the structural mutation only one cluster is added or deleted at a time.  $\alpha$  (used in Equation(6.22)) is

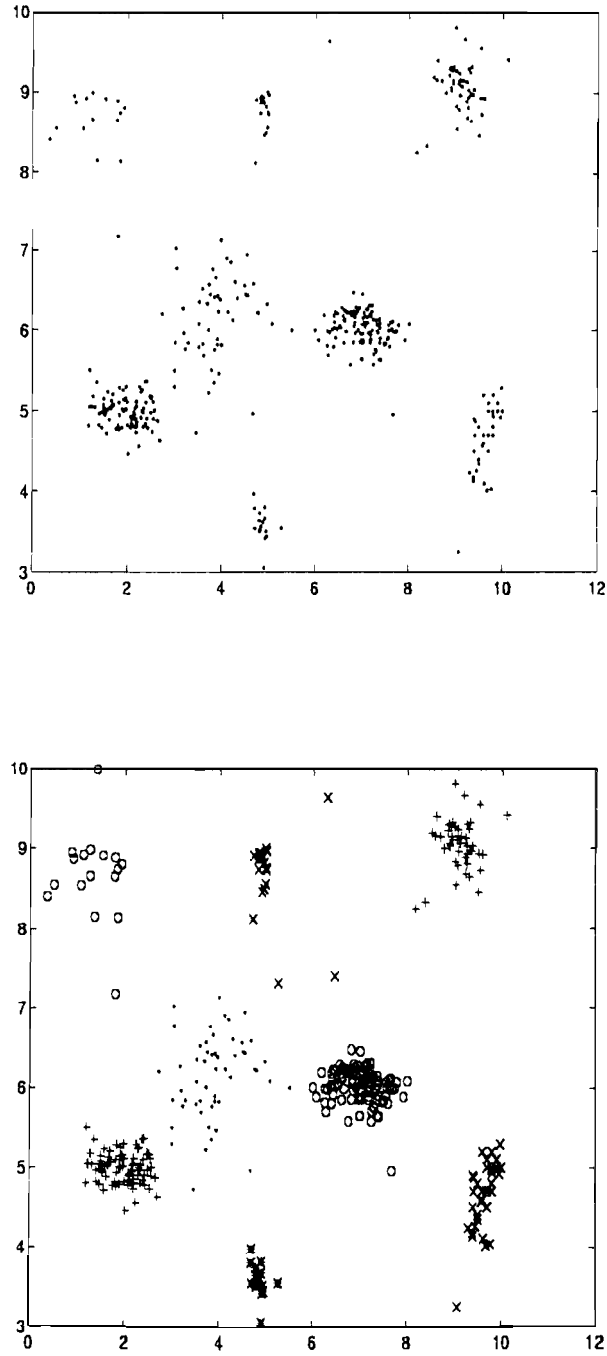
1. If the current generation is the first generation follow this step, else skip it. For each parent set randomly generate the number of clusters, and randomly determine the cluster centers within the range of input patterns. Assume that the number of clusters generated is  $K$ , where  $K$  is in between two and some prespecified integer.
2. Set  $i = 0$ .
3. Find a fuzzy partition  $U^i$  of  $\mathcal{Y}$  by using Equation (6.11), (6.12) and the already known cluster centers.
4. DO
  - (a) Assign  $i = i + 1$ .
  - (b) Calculate the  $K$  cluster centers  $\{\mathbf{m}_k | 1 \leq k \leq K\}$  using the following relation:
 
$$\mathbf{m}_k = \frac{\sum_{j=1}^n (\mu_k(\mathbf{y}_j))^q \mathbf{x}_j + \mathbf{m}_k}{\sum_{j=1}^n (\mu_k(\mathbf{y}_j))^q + 1} \quad (6.23)$$
  - (c) Update  $U^i$  by using Equation (6.11) and (6.12).
- END DO UNTIL  $\text{norm}(U^i - U^{(i+1)}) > q$
5. Repeat step 4 with the distance measure given in Equation (6.4). This step helps to obtain better clustering.

**Fig. 6.3:** Modified fuzzy K-means algorithm. Here  $\eta$  is a constant with small value and  $\text{norm}()$  is an appropriate matrix norm.

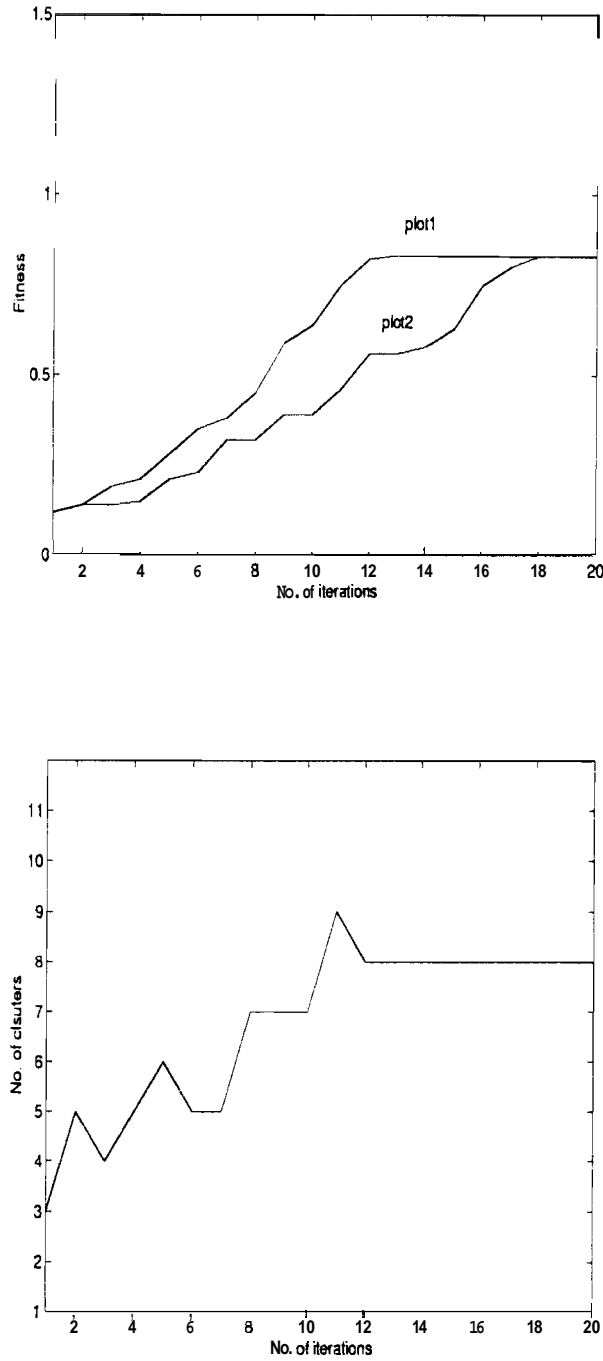


chosen as 0.6. If the value of  $a$  is varied slightly, then the clustering results remain same. Fig. 6.4Bottom depicts the clustered data after using the proposed clustering algorithm. Number of generations and the corresponding fitness of the best set of clusters is shown in Fig. 6.5Top. In fact, this fitness value is average of the fitness values of the best set in ten runs. Here "plot 1" represents a curve when Equation (5.50-a and b) are used for adapting the parameters of the structural mutation. In contrast, "plot 2" represents another curve when Equation (5.50-a and b) are not used for adapting the parameters of the structural mutation. This comparative study demonstrates that Equation (5.50-a and b) enhances the performance of the search process. But it also shows that the enhancement of the performance is not as much as it was while configuring FFNNs in chapter 5. The proposed algorithm finds the optimum number of clusters after 12 generation (see Fig. 6.5Bottom). The clustered output is close to the desired one. The proposed algorithm self-organizes to find the proper number of clusters and proper cluster centers automatically. This figure illustrates the self-organization capability of the proposed algorithm, due to which, the proposed algorithm does not find any problem in clustering, eventhough it starts with wrong number of clusters and incorrect position of the cluster centers. This figure also shows that sets with different structural variation always come during the whole process. In fact, it exhibits that search for better set of clusters (structurally) is carried out all round the process. Even after assigning the number of clusters as nine, the FKM (followed by FMLE) failed to cluster the data set properly. Fig. 6.6Top and Fig. 6.6Bottom show the results of using the FKM (followed by FMLE) on the data set with two different initializations. The clustering results with both the initializations are bad. Apparently the FKM, FMLE combination was stuck in local minima due to improper initializations. If only the FKM algorithm is used, clustering result becomes worse than this result. After the proposed algorithm converges on this data set, the value of  $E_f$  is calculated from Equation (6.2). It is found to be 5% less than the value of  $E_f$  obtained after the FKM (followed by FMLE) converges on this same data set. It demonstrates the usefulness of the proposed method to avoid parametric local minima.

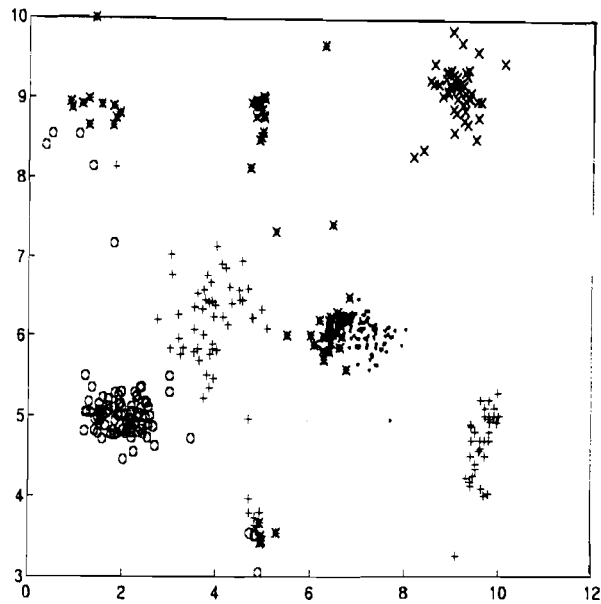
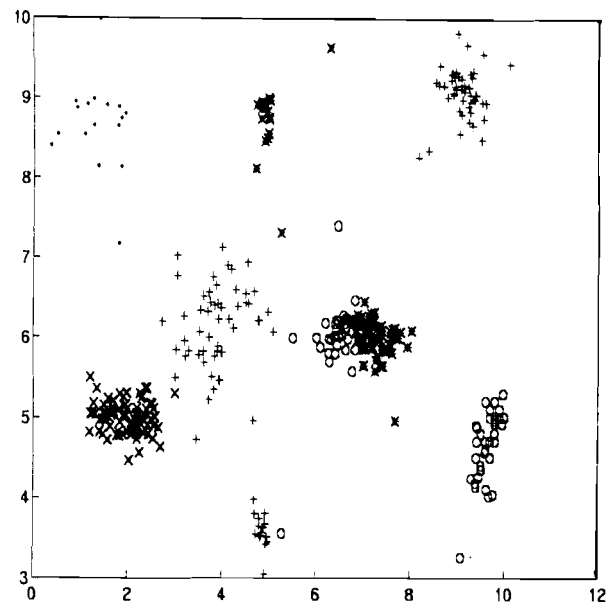
Next we use the proposed method on the opening Bid problem. We considered the training sets "TrainingSet1", "TrainingSet2" and "TrainingSet3" for first, second and third level bids. From "TrainingSet1", we collected the inputs only for Pass bids. We used the proposed clustering scheme to cluster these patterns. Twelve clusters were evolved after 23 generations. Using the similar procedure, we got 8, 6, 7, 5, 6, 3 clusters for the input patterns corresponding to 1C, ID, 1H, 1S and 1N, respectively. From "TrainingSet2", we obtained 5, 4, 5, 2 and 3 clusters corresponding to 2C, 2D, 2H, 2S



**Fig. 6.4:** Top: Eight different Gaussian distributions are used to generate a data set artificially. Bottom: Clustered output by the proposed clustering algorithm. The clustered output is close to the desired one.



**Fig. 6.5:** Top: No. of iterations vs. fitness curve for the best set of clusters in the proposed clustering algorithm. The data set for these clusters are shown in Fig. 6.4. "Plot1" and "Plot2" represent the curve with and without using Equation (5.50-a and b), respectively. Bottom: No. of iterations vs. no. of clusters for the best member of the population.



**Fig. 6.6:** Fuzzy K-means algorithm is used on the data set shown in Fig. 6.4. Top and Bottom: Clustered outputs with two different sets of random initial cluster centers. Due to improper initialization the clustered outputs are not close to the desired one.

and  $2N$ , respectively. clusters. From “TrainingSet3”, we obtained 2, 2, 3 and 2 clusters corresponding to 3C, 3D, 3H and 3S, respectively. We will use these clusters in the next section while constructing the fuzzy-rough neural networks.

Note that in some cases the fuzzy hypervolume does not attain the minimum value with the optimal number of clusters. In this cases, the clustering output may not be good. Moreover, in some cases the clustering is highly subjective. More than one possible way may be present to cluster the input. Moreover, in the proposed clustering algorithm we are assuming that the clusters are ellipsoidal. However, the proposed method may not give good results when this assumption is not valid. For instance, if the clusters are of shell type, the proposed algorithm will not work. The advantage of the proposed method is that one can do other modifications in the given framework.

### 6.3 Fuzzy-Rough Neural Networks

After clustering the next step involves labelling of each cluster. To accomplish it, a three-layer feedforward network can be constructed, where each node in the hidden layer represents the cluster centers and the weights between the hidden and the output nodes represent the class labels attached to the clusters. The main idea of using the fuzzy clustering is that if two input patterns are similar, i.e., close neighbors in the input pattern space, then the class labels associated with them will be same. Since each cluster in the pattern space represents certain common property, it is logical that the patterns from the same cluster will also belong to the same class. When a new pattern is presented at the input layer, the network classifies precisely based on the similarity or neighborhood property. Thus the inherent similarity or neighbourhood property of the clusters leads the network to generalize. In real life cases, however, we cannot extract all the relevant features necessary for the classification. Consequently, two patterns may have the same or similar feature values, but they are not same or similar if the other features, including the existing ones, are accounted for. Therefore, when the input patterns are clustered based on the available features, two apparently similar or neighboring patterns may have different class labels. It makes the output classes *indiscernible* or indistinguishable based on the given set of features. Consequently, the relationship between each cluster and the class labels becomes rough. One way to completely avoid the rough uncertainty is to extract the essential features so that distinct feature vectors are used to represent different objects. But, it may not be possible to guarantee as our knowledge about the system generating the data is limited [SS93]. Another way to avoid rough uncertainty is to

break the clusters further so that they do not contain any pattern from the other clusters. This is difficult as each fuzzy cluster to some extent covers patterns from the other clusters. Moreover, the breaking of clusters means destruction of the similarity property, which in turn means the destruction of the generalization property of the network. In addition, if the clusters are broken too much, then the network training may need large space and high time complexity.

In this section, we attempt to reduce the effect of rough uncertainty, while keeping the similarity property intact. To tackle the similarity property we need fuzzy sets [KY95], and to tackle roughness we need rough sets [Paw82]. Both fuzziness and roughness associated with each modified feature vector is captured using *fuzzy-rough membership functions*. The fuzzy-rough membership function is further exploited to construct a *fuzzy-rough neural network* (FRNN). Basically, the FRNN uses the *fuzzy* uncertainty involved in the input data set and the roughness present in the input-output relationship. One advantage of the classification procedure used in the FRNN is that it is *possibilistic* [KY95]. It is useful because the output of the FRNN will be used again while combining the classification result. Theoretically the FRNN is a powerful classifier as it is a universal approximator [SY98b].

### 6.3.1 Root of Fuzzy-Rough Neural Networks

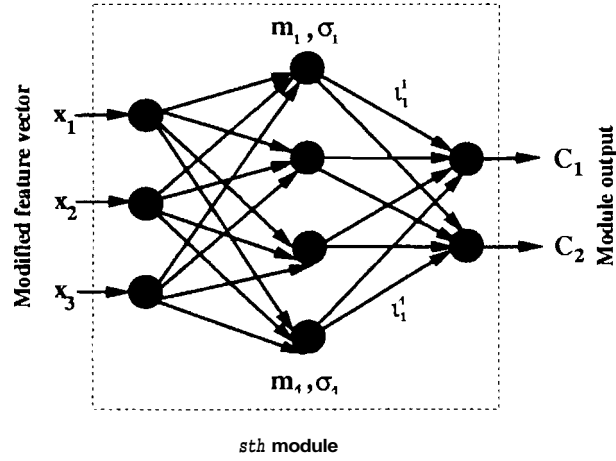
The FRNN is designed such that the outputs of the networks are fuzzy-rough membership values corresponding to the input. The fuzzy-rough membership function of a pattern captures both fuzziness and roughness associated with the pattern. Let,  $\tau_{C_e}(\mathbf{y})$  represent the fuzzy-rough uncertainty of  $y$  in the class  $C_e$ .  $\tau_{C_e}(\mathbf{y})$  is defined as

$$\tau_{C_e}(\mathbf{y}) = \begin{cases} \frac{1}{H} \sum_{j=1}^H \mu_{F_j}(\mathbf{y}) \iota_{C_e}^j(\mathbf{y}) & \text{if } \exists j \text{ with } \mu_{F_j}(\mathbf{y}) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (6.24)$$

where  $\{F_1, F_2, \dots, F_H\}$  are the fuzzy clusters generated by evolutionary programming-based fuzzy clustering algorithm,  $H$  is the number of cluster in which  $y$  has non zero membership and  $\iota_{C_e}^j(\mathbf{y}) = \frac{|F_j \cap C_e|}{|F_j|}$ . Appendix-D contains a detail description about the fuzzy-rough membership functions.

### 6.3.2 Architecture of Fuzzy-Rough Neural Networks

The proposed FRNN is a three layered feedforward network with one hidden layer (Fig. 6.7). The number of nodes in the input, hidden and output layers are equal to



**Fig. 6.7:** A typical fuzzy-rough neural network with three input nodes, four hidden nodes and two output nodes.

the dimension of the input pattern ( $=N$ ), number of the fuzzy clusters present in the input data ( $=H$ ) and number of the classes ( $=C$ ), respectively. When an input pattern  $\mathbf{y} = [y_1, y_2, \dots, y_N]$  is applied at the input layer of the network, the output of the  $j$ th hidden node is

$$o_j^h = \exp \left[ -\frac{(\mathbf{y} - \mathbf{m}_j)(\mathbf{y} - \mathbf{m}_j)}{2\sigma_j^2} \right] \quad (6.25)$$

where  $\mathbf{m}_j$  and  $\sigma_j$  (assuming the spread is **same** along all directions) are the center and spread of the Gaussian function used in the  $j$ th hidden node. The center and spread of the hidden nodes can be determined by making them equal to the mean and variance of the clusters. The mean and variance of each cluster are determined using the evolutionary programming-based fuzzy clustering algorithm, which is described in the previous section. The outputs of the hidden nodes can also be interpreted as the fuzzy membership values. The parameters necessary for the FRNN can be obtained from the parameters defined in the input space (Table 6.1). The output of the  $k$ th output node is

$$o_k^o = \sum_{j=1}^H o_j^h w_{jk} \quad (6.26)$$

where  $w_{jk}$  is the weight from the  $j$ th hidden node to the  $k$ th output node. The output value  $o_k^o$  lies in between 0 and 1 (from Property D.1 of Appendix-D) as the output is the fuzzy-rough membership value corresponding to the input. Moreover, from Property D.6 of Appendix D,  $o_k^o$  is possibilistic.

Table 6.1: The relationship between the parameters used in fuzzy-rough neural networks and input space.

Fuzzy-Rough neural networks	Input Space
No. of the input nodes	= Dimension of the input patterns
No. of the hidden nodes	= No. of the clusters
No. of the output nodes	= No. of the classes
Center of the $j$ th hidden node	= Center of the $j$ th cluster
Width of the $j$ th hidden node	= Width of the $j$ th cluster

### 6.3.3 Training and Testing of Fuzzy-Rough Neural Networks

To design the FRNN, the last task is to adjust the weights between the hidden layer and the output layer through training. Precisely, the weights between the hidden and the output layer reflect the rough-fuzzy membership values. For training, all the weights,  $w_{jk}(0) \forall j, k$  are initialised to zero. For each input training pattern, the weight adjustment is carried out as

$$\Delta w_{jk}(1) = o_j^h * i \quad \forall j, k \quad (6.27)$$

where  $i = 1$  if  $y \in C_k$  else  $i = 0$ . It is interesting to note that the training process takes exactly one iteration. After the whole cycle is over,  $w_{jk}$  represents  $|F_j \cap C_k|$ , i.e.,  $\sum_{y \in C_k} \mu_{F_j}(y)$ . To make  $w_{jk} = \iota_{C_k}^j$ ,  $w_{jk}$  is normalized as  $\frac{w_{jk}}{\sum_k w_{jk}}$  (since  $|F_j| = \sum_k \sum_{y \in C_k} \mu_{F_j}(y) = \sum_k w_{jk}$ ). Since all the hidden nodes are using Gaussian clusters, each input pattern belongs to all the clusters, and hence,  $H = H$ . Finally, the weights are set as  $w_{jk} = \frac{w_{jk}}{H}$  to take care of the term  $H$  involved in Equation (6.24). Note that no bias term is involved here with any node.

In the testing stage, a separate set of test patterns is given as the inputs to the network. For the test input  $y$ , the generated output at the  $c$ th output node is the fuzzy-rough membership value  $\tau_{C_c}(y)$ . Since the fuzzy-rough membership functions are possibilistic (see Property D.6 of Appendix-D), the outputs of the FRNN are also possibilistic.

It can be shown that architecturally (although functionally not) FRNNs are equivalent to radial basis function neural networks [SY98b]. Since radial basis function neural networks are universal approximators [JSM97], FRNNs are also universal approximators.



### 6.3.4 Results and Discussion

Through clustering we have obtained the clusters corresponding to the fifteen classes present in the opening bid problem. This cluster information is used to construct an FRNN. We used “TrainingSet1” to train the network for the first level bids. From section 6.2.4, the number of clusters is 21 (5 for ‘P’, 5 for ‘1C’, ‘6’ for ‘1D’). Hence for the first level bids we used 21 hidden nodes. The resultant FRNN has fifty input nodes, twenty-nine hidden nodes and six output classes. The first row of Table 6.2 shows the classification performance of FFNNs on the first level bids. This result is reproduced from Table 5.1. The second row of Table 6.2 exhibits the performance of the FRNN. While comparing with the FFNN for first level bids, we can observe that the performance of the FRNN is better than that of the FFNN. The time needed to configure the FRNN is also less.

Similarly, FRNNs were constructed for the second and third level bids. The data sets “TrainingSet2” and “TrainingSet3” were used to construct the FRNNs. To test the performance of these two FRNNs, we used “TestSet2” and “TestSet3”. The FRNN for the second level bids consist of fifty-two input nodes and five output nodes. Since the total number of clusters for the second level bids is 19, the number of hidden nodes is chosen as 19. The comparative classification performance of the FRNN and the FFNN is given in Table 6.3. For the third level bids, the FRNN has fifty-two input nodes, nine hidden nodes and four output nodes. The classification performance of the FRNN and the FFNN is compared in Table 6.3.

It can be observed that for first and second level bids, the performance of FRNNs are better than the FRNN. In contrast, the FFNN for third level bids perform better than the FRNN. In lower level bids roughness is very high. FRNNs take roughness into account, and hence, they perform better than FFNNs for lower level bids. The role of rough uncertainty is less for the inputs corresponding to the higher level bids. Hence, in this case the FFNN can approximate the class boundaries more effectively than the FRNN. Some differences between the FFNN approach and the FRNN approach are: 1) FRNNs utilise the structure present in the data explicitly, whereas in FFNNs the use of the structure is implicit. 2) FFNNs with the fuzzy objective functions do not consider rough uncertainty. In contrast, FRNNs take care of rough uncertainty. 3) FFNNs with the fuzzy objective functions perform well when the decision boundary is very complicated. However, the performance degrades as soon as the roughness in the data set becomes high. On the other hand, although the performance of FRNNs is poor in presence of complex decision

Table 6.2: Comparative classification performance of FFNNs and FRNNs for first level bids.

Network	Pass	1C	1D	1S	1H	1N	Overall
FFNN	80.64%	75.14%	79.38%	85.63%	70.44%	73.02%	77.37%
FRNN	86.05%	78.30%	81.62%	79.63%	77.23%	86.75%	81.42%

Table 6.3: Comparative classification performance of FFNNs and FRNNs for second level bids.

Network	2C	2D	2S	2H	2N	Overall
FFNN	75.82%	77.57%	80.24%	78.12%	76.68%	77.68%
FRNN	88.18%	76.12%	81.57%	90.01%	82.51%	83.67%

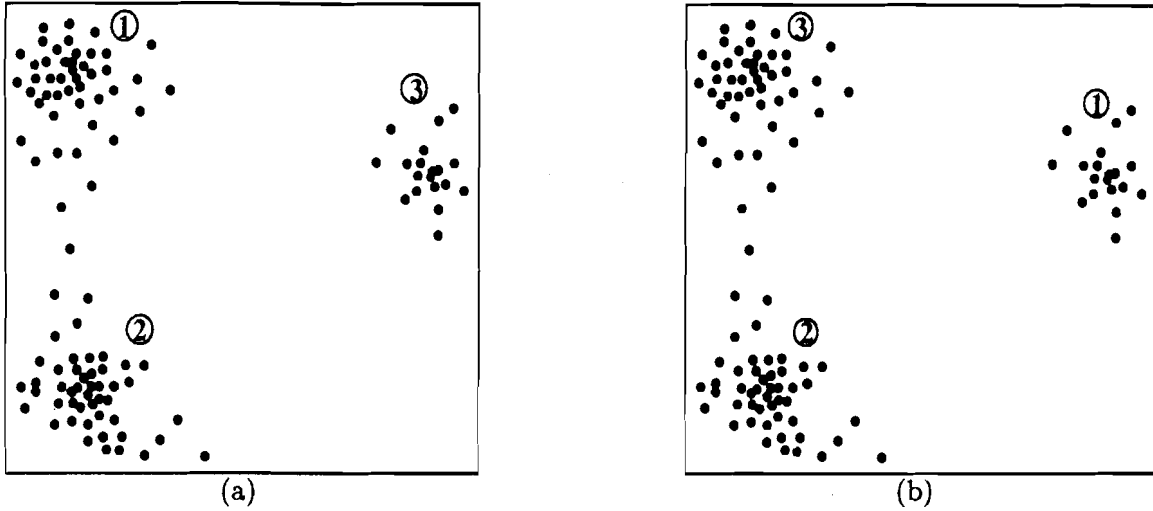
Table 6.4: Comparative classification performance of FFNNs and FRNNs for third level bids.

Network	3C	3D	3S	3H	Overall
FFNN	89.23%	88.52%	93.15%	85.63%	89.13%
FRNN	91.76%	79.88%	89.54%	78.87%	85.01%

boundaries, FRNNs exploit rough uncertainty to enhance the classification performance. The effectiveness of these two approaches depend on the type of the decision boundary and the roughness present in the classification task.

## 6.4 Summary

In this chapter, an evolutionary programming-based clustering algorithm is proposed. The algorithm effectively groups a given set of data into an optimum number of clusters. The algorithm determines the number of clusters and the cluster centers in such a way that locally optimal solutions are avoided. The result of the algorithm does not depend critically on the choice of the initial cluster centers. The clusters are used to construct an FRNN. The parameters for the hidden nodes and the number of hidden nodes are determined from the clusters. The weight between each hidden node and the output class



**Fig. 6.8:** (a) and (b) are two equivalent sets of clusters, which order their clusters differently. Although phenotype representation for both the sets are same, genotype representations are different.

is determined using the rough-fuzzy membership functions. The outputs of the network are fuzzy-rough membership values corresponding to the modified feature vectors.

Like FFNN configuration, in the clustering problem also, an EP-based optimization approach is advantageous over a genetic algorithm-based approach. Here, the permutation problem stems from the fact that in genetic algorithm two functionally identical sets of clusters, which order their clusters differently, have two different genotype representations (see Fig. 6.8). Therefore, the probability of producing a highly fit offspring from them by crossover will be very low.

The difference between FRNNs and radial basis function networks should be noted. The working principle of radial basis function neural networks is similar to that of FRNNs. But, FRNNs consider the rough uncertainty present in the clusters, whereas the radial basis functions do not take roughness into its account. In particular, the use of rough-fuzzy membership functions makes FRNNs more powerful than radial basis function neural networks. Detail comparison between these two networks is given in [SY98b].

When both the input clusters and the output classes are crisp, the outputs of the FRNN are rough membership values (see Property D.4 of Appendix-D). Hence, the resultant FRNN architecture is reduced to a modified architecture, called *rough neural networks*. The architectural difference between FRNNs and rough neural networks is in

the transfer function used in each hidden node. In particular, the transfer function used in the FRNN is of Gaussian type, whereas in the case of rough neural networks the transfer function is a unit gate function, i.e.,

$$o_j^h = \begin{cases} 1 & \text{if } (\mathbf{y} - \mathbf{m}_j)(\mathbf{y} - \mathbf{m}_j) \leq 2\sigma_j^2 \\ 0 & \text{otherwise} \end{cases} \quad (6.28)$$

Evidently, the 1/0 option used in the gate function makes the generalization capability of rough neural networks limited.

When the input clusters are crisp and fine, and the output classes are crisp, then the outputs of the FRNN are the crisp class membership values (see Property D.5 of Appendix-D). The resultant network can be called crisp neural network. The number of hidden nodes of the crisp neural network is equal to the number of inputs. Since the width of each cluster approaches towards zero, the transfer function of each hidden node becomes a unit impulse function, i.e.,

$$o_j^h = \begin{cases} 1 & \text{if } \mathbf{y} = \mathbf{y}_j \\ 0 & \text{otherwise} \end{cases} \quad (6.29)$$

As a result, the weight calculation becomes very simple, i.e.,  $w_{jk} = 1$  if  $\mathbf{y}_j \in C_k$  else  $w_{jk} = 0$ . Thus, the resultant crisp neural network needs a large amount of space, and it works like a look-up table, which does not have any generalization capability, but has a very good memorising power.

# Chapter 7

## FUSION OF CLASSIFICATION RESULTS

### 7.1 Introduction

Till now we have divided the original classification task among small feedforward subnetworks, and we have built modules to accomplish the subclassification tasks. In this chapter we combine the individual solution provided by the modules to obtain the final classification result. The proposed method interprets each subnetwork as a nonlinear filter tailored to the subgroup. The outputs of all the filters can be viewed as a feature vector representing the input. We may call these features *secondary features* to distinguish it from the features that we obtained in chapter 4. In fact, the features what we obtained in chapter 4 undergoes nonlinear filtering and generates the secondary feature vectors. For the sake of brevity we will call the secondary features also features as long as no confusion exists. Each module classifies the input pattern from different angles. Each feature, i.e., the output of each module, can be considered as an evidence in classifying the input. Since the modules are trained locally and the modules cannot resolve the global uncertainties, each of the evidence may support or contradict one another. For instance, the following two conditions may arise:

1. If the classes of two modules are close or overlapping, then for an input, outputs of both the modules will be high. In other words, each of these two modules claims that the input can be classified by the module alone.
2. Due to roughness, an input may completely belong to two different classes. If these classes are from two different modules, then for a similar test input, both the modules will produce high outputs. It indicates that the input belongs to both the modules.

Some modules may cooperate each other also. For instance, the low output value of a module may automatically indicate the high output in some other modules. Due to

the presence of conflicts and cooperations, each feature would have a different degree of importance in classifying the input to a particular class.

To fuse the information supplied by each module, various methodologies like *winner-take-all* [Hay94], *probabilistic (Bayesian) reasoning* [JJ93], *Dempster-Shafer theory* [Sha76], *fuzzy integral* [CK95a] [CK95b] [Yag93] [Cho95] [YF93] [Gra97] [WK92] exist. In the winner-take-all technique, outputs of all the subnetworks are combined by simply choosing the class with the largest output value. This method does not consider the importance of each feature. Since each module is not trained to discriminate all the classes and all the modules are not trained upto the same accuracy, the performance of this scheme is poor [CK95a] [Cho97]. For the information fusion, Bayesian reasoning utilises the importance of each feature. But, while combining the importance of more than one evidence, it relies on probability theory, which cannot discriminate between lack of evidence and negative evidence [KO96]. On the other hand, Dempster-Shafer's theory and the fuzzy integral can distinguish between lack of evidence and negative evidence. As we have discussed in section 2.3.4, the fuzzy integral approach has a way to assess the importance of all groups of information sources towards supporting a particular hypothesis as well as the degree to which each information source supports the hypothesis. In contrast, the Dempster-Shafer theory does not have this advantage [KGT<sup>S</sup>94]. In addition, fuzzy integral is computationally more efficient than the Dempster-Shafer approach. Due to these merits, this chapter applies a fuzzy integral-based fusion method in combining the subnetworks. In particular, a special type of fuzzy integral, known as *Sugeno's fuzzy integral*, is used. Henceforth, we use the term fuzzy integral to mean Sugeno's fuzzy integral.

The behavior of the fuzzy integral in an application depends critically on the importance of the subsets of the features. Therefore, determination of the worth of each feature is very important. In some applications of the fuzzy integral, the importance is supplied subjectively by an expert or it is estimated directly from the data [TK90] [SY98e]. These methods require some kind of prior knowledge about the behavior of the outputs generated by the modules. In many applications, it may be difficult to obtain the prior knowledge. However, it is interesting to note that in the fuzzy integral approach, influence of the other features on a given feature is not considered. Hence, determination of the importance of a particular feature is based on the partial information supplied by the feature itself. A feature is important for a particular class when all the input patterns can be classified correctly to that class only using this feature value. This is possible when all

the input patterns are clustered based only on this feature value and all the input patterns from each cluster have the same class label. When it does not happen, the relationship between the clusters and the output class labels becomes **one-to-many**. It results in rough ambiguity [Paw82]. In most of the cases, the clusters formed in the input space based on each feature value is fuzzy. Therefore, in this article, an attempt is made to determine the importance of each feature using fuzzy-rough set [DP92] theoretic technique.

The chapter is organized as follows: In section 7.2 we discuss the basics of fuzzy measure, fuzzy integral and rough sets. In section 7.3 the proposed method is described. Section 7.4 demonstrates the experimental results.

## 7.2 Background

### 7.2.1 Fuzzy Measure

Let  $\Xi$  be a finite set of elements. A set function  $g : 2^\Xi \rightarrow [0, 1]$  with the following properties is called a fuzzy measure [Sug74]:

$$P1: g(\phi) = 0$$

$$P2: g(\Xi) = 1$$

$$P3: \text{If } U \subseteq V, \text{ then } g(U) \subseteq g(V), \text{ where } U, V \subseteq \Xi$$

The fuzzy measure generalizes the classical measure which plays a crucial role in probability and integration theory. A probability measure  $P$  is characterized by the property of additivity: For all sets  $U$  and  $V$ , if  $U \cap V = \phi$ , then  $P(U \cup V) = P(U) + P(V)$ . In the fuzzy measure, this property of additivity is weakened by the more general property of monotonicity (property P3). Sugeno's  $g_\lambda$  measure is a special type of fuzzy measure [Sug74] which satisfies all the properties of the fuzzy measure, in addition to the following:

$$g(U \cup V) = g(U) + g(V) + \lambda g(U)g(V) \quad (7.1)$$

where  $\lambda > -1$ ,  $U, V \subseteq \Xi$  and  $U \cap V = \phi$ . By varying the values of  $\lambda$ , one can obtain different types of fuzzy measure. For example,  $\lambda = 0$  produces the probability measure.

### 7.2.2 Fuzzy Integral

Let  $\Xi = \{\xi_1, \xi_2, \dots, \xi_S\}$  be a finite set of elements,  $h : \Xi \rightarrow [0, 1]$  be a mapping and  $g$  be a fuzzy measure on  $\Xi$ . Then the fuzzy integral (over  $\Xi$ ) of the function  $h$  with respect to

the fuzzy measure  $g$  is defined as

$$\begin{aligned}\mathcal{F} &= h(\Xi) \circ g() \\ &= \max_{\Omega \subseteq \Xi} \left[ \min \left( \min_{\xi_s \in \Omega} (h(\xi_s)), g(\Omega) \right) \right]\end{aligned}\quad (7.2)$$

where  $1 \leq s \leq S$ . Since both  $h$  and  $g$  map to  $[0, 1]$ ,  $\mathcal{F}$  also lies in  $[0, 1]$ . The above integral can be seen as an extension of Lebesgue integral if product and summation operators are substituted for  $\min$  and  $\max$ , respectively. Intuitively the interpretation of the above relation is as follows: Let us suppose, an object is evaluated using a set of information sources  $E$ . Let  $h(\xi_s) \in [0, 1]$  denote the decision for the object when a single information source  $\xi_s \in \Xi$  is considered. Moreover, suppose  $g(\{\xi_s\})$ , known as fuzzy density, denotes the importance of the source  $\xi_s$ . Instead of a single information source, if a set of sources, namely  $\Omega \subseteq \Xi$ , is taken to evaluate the object, then it is reasonable to consider  $\min_{\xi_s \in \Omega} h(\xi_s)$  as the largest security decision. Evidently,  $g(\Omega)$  expresses the degree of importance or the expected worth of the set  $R$ . Therefore,  $\min \left( \min_{\xi_s \in \Omega} (h(\xi_s)), g(\Omega) \right)$  denotes the grade of agreement between the real possibility  $h$  and the expectation  $g$ . Thus, the fuzzy integral can be interpreted as a search for the maximal grade of agreement between the objective evidence and the expectation. However, the definition can further be simplified if  $h(\xi_s)$ ,  $s = 1, 2, \dots, S$  are ordered in a decreasing manner. Let  $h(\xi_1) \geq h(\xi_2) \geq \dots \geq h(\xi_S)$  (if not,  $E$  is rearranged so that this relation holds). Then Equation (7.2) is simplified to

$$\mathcal{F} = h(\Xi) \circ g() = \max_s \left[ \min (h(\xi_s), g(\Omega_s)) \right] \quad (7.3)$$

where  $\Omega_s = \{\xi_1, \xi_2, \dots, \xi_s\}$ .

In order to evaluate the fuzzy integral, i.e.,  $\mathcal{F}$ , we should have some way to determine  $g(\Omega_s)$  from  $g(\{\xi_s\})$ . For that, we need to use the concept of fuzzy measure. In the next section we will show how to determine the individual fuzzy densities  $g(\{\xi_s\})$ ,  $s = 1, 2, \dots, S$  for each information source from the given data. For the time being, let us suppose that we know the fuzzy densities of the individual sources. But,  $g(\Omega_s)$  is not necessarily equal to  $g(\{\xi_1\}) + g(\{\xi_2\}) + \dots + g(\{\xi_s\})$ . The simple additive property may not hold because there may be some interactions among  $\xi_s$ . If the interactions are cooperative, then  $g(\Omega_s) \geq g(\{\xi_1\}) + g(\{\xi_2\}) + \dots + g(\{\xi_s\})$ . On the contrary, if the



interactions are noncooperative, then  $g(\Omega_s) \leq g(\{\xi_1\}) + g(\{\xi_2\}) + \dots + g(\{\xi_s\})$  [MS89b]. From this discussion, note that probability theory cannot be used to determine the value of  $g(\Omega_s)$ . However, the concept of Sugeno's  $g_\lambda$  fuzzy measure can be exploited here to find the value of  $g(\Omega_s)$ . The procedure is as follows:

$$\begin{aligned}
g(\Omega_1) &= g(\{\xi_1\}) \\
g(\Omega_2) &= g(\{\xi_2\}) + g(\{\xi_1\}) + \lambda g(\{\xi_2\})g(\{\xi_1\}) \\
&= g(\{\xi_2\}) + g(\Omega_1) + \lambda g(\{\xi_2\})g(\Omega_1) \\
&\dots \quad \dots \quad \dots \quad \dots \\
g(\Omega_s) &= g(\{\xi_s\}) + g(\Omega_{s-1}) + \lambda g(\{\xi_s\})g(\Omega_{s-1}) \quad \text{for } 1 < s \leq S
\end{aligned} \tag{7.4}$$

One problem remains still unresolved; that is, how to determine  $\lambda$ , which is the key term to decide the amount of interactions among the information sources. In order to find  $\lambda$ , we use Equation (7.4), and we express  $g(\Xi)$  in terms of the individual fuzzy densities as follows:

$$\begin{aligned}
g(\Xi) &= g(\{\xi_S\}) + g(\{\xi_1, \xi_2, \dots, \xi_{S-1}\}) \\
&\quad + g(\{\xi_S\})\lambda g(\{\xi_1, \xi_2, \dots, \xi_{S-1}\})
\end{aligned} \tag{7.5}$$

$$\begin{aligned}
&= \sum_{s=1}^S g(\{\xi_s\}) + \lambda \sum_{s=1}^{S-1} \sum_{k=s+1}^S g(\{\xi_s\})g(\{\xi_k\}) + \dots \\
&\quad + \lambda^{S-1} g(\{\xi_1\})g(\{\xi_2\}) \dots g(\{\xi_S\})
\end{aligned} \tag{7.6}$$

$$= \left[ \prod_{s=1}^S (1 + \lambda g(\{\xi_s\})) - 1 \right] / \lambda \quad \text{where } \lambda \neq 0 \tag{7.7}$$

From (P2), we know that the value of  $g$  over the whole set  $\Xi$  must be one as no uncertainty is involved. Hence, using  $g(\Xi) = 1$  and Equation (7.5)

$$\prod_{s=1}^S (1 + \lambda g(\{\xi_s\})) = \lambda + 1 \tag{7.8}$$

It is possible to find the value of  $\lambda$  after solving the above  $(S - 1)$ th degree equation. In [TK90], it has been shown that  $\lambda$  has a unique value in  $(-1, 0) \cup (0, +\infty)$  when  $0 < g(\{\xi_s\}) < 1$ ,  $\forall s = 1, 2, \dots, S$ .

## 7.3 Modular Networks with Proposed Fusion Technique

### 7.3.1 Architecture of Modular Networks

In third chapter, the given pattern classification task is subdivided into three subtasks, and one subnetwork is assigned for each subtask. We make this statement slightly general by assuming that the original problem has  $M$  output classes  $\{C_1, C_2, \dots, C_M\}$ , and these classes are divided into  $S$  subnetworks (Fig. 7.1). The  $s$ th subnetwork is assigned to classify a group of classes, represented by  $\{C_{c_{s-1}+1}, \dots, C_{c_s}\}$  with  $c_0 = 0$  and  $c_S = M$ . The output of the  $s$ th subnetwork is  $\{y_{c_{s-1}+1}, \dots, y_{c_s}\}$ , which is expressed in a vector notation as  $\xi_s = [y_{c_{s-1}+1}, \dots, y_{c_s}]$ . The proposed method interprets each subnetwork as a nonlinear filter tailored to the subgroup. Thus, the outputs of all the filters corresponding to an input  $\mathbf{x}$  is viewed as an  $S$  dimensional feature vector. This feature vector is presented as an input to a fuzzy integrator, which computes the value of the fuzzy integral with the help of fuzzy densities. The class label of the input  $\mathbf{x}$  is the class index that yields the maximum value of the fuzzy integral corresponding to  $\psi$ .

### 7.3.2 Training of Modular Networks

When a modular network is used for classification, a given training pattern is input to all the subnetworks and the outputs of the subnetworks are processed to determine the class label. We can decide the class label of the input based on *winner-take-all* policy. It means that the class label of the input pattern is assigned as  $j$ ,  $1 \leq j \leq M$ , if  $y_j = \max_{k=1,2,\dots,M} \{y_k\}$ . However, this type of assignment is not proper as all the subnetworks are independently trained on different sets of data. A better approach is to declare the  $j$ th class winner, if the  $j$ th class correspondences to  $\max_{k=1,2,\dots,M} \{g_k y_k\}$ , where  $g_k$  is the importance associated with the class  $C_k$ . One possible choice for  $g_k$  is the *a priori* probability of the class  $C_k$ . However, the constraint  $\sum_{k=1,2,\dots,M} g_k = 1$  used in probability theory cannot distinguish between lack of evidence and ignorance. Therefore, the concept of fuzzy integral is appealing here. In the fuzzy integral approach, the outputs of the modules are processed further so that the interactions among the outputs are also exploited for the final classification result. Hence, the term  $g_k$  is replaced by a more specific term  $g_k(\{\xi_s\})$ , where  $g_k(\{\xi_s\})$  denotes the importance of  $\xi_s$  in characterizing the class  $C_k$ . With the help of  $g_k(\{\xi_s\})$ ,  $s = 1, 2, \dots, S$ , the fuzzy integral  $\mathcal{F}_k$  for the class  $C_k$  combines the outputs of all the modules, i.e.,  $\xi_s$ ,  $s = 1, 2, \dots, S$ , in a nonlinear fashion. The final class label corresponding to the

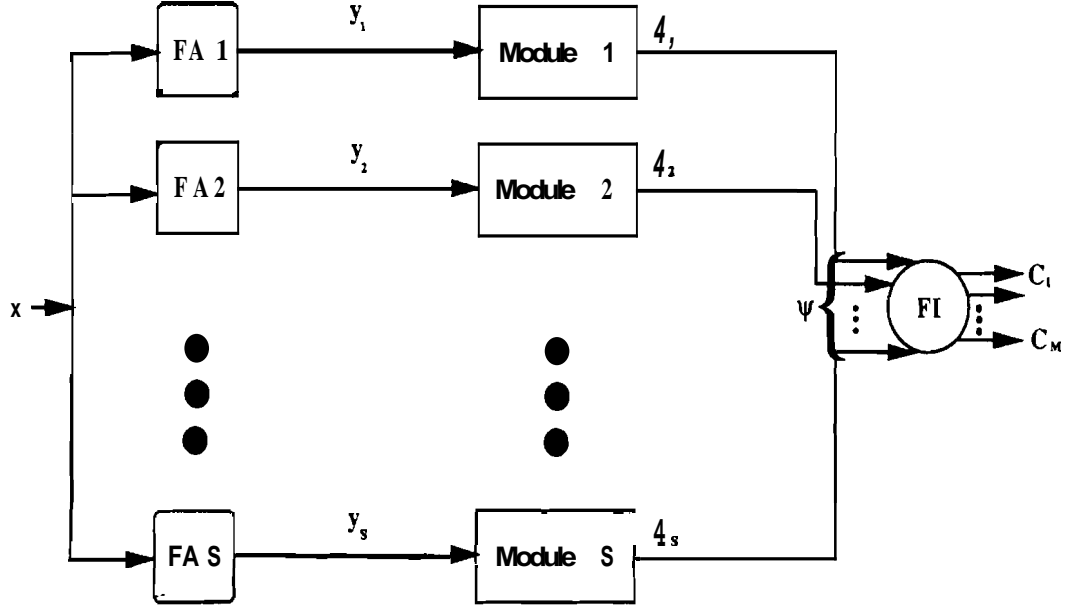


Fig. 7.1: A modular network with  $S$  modules or subnetworks. Initially the input pattern is fed to  $S$  different feature analysers (FA). The feature analysers modify the input by providing different weightage on each feature. The modified feature set is passed to the module connected to the feature analyser. The output of the  $s$ th module is represented by  $\xi_s$ . All the outputs are combined in a nonlinear manner by a fuzzy integrator (FI).  $\psi$  denotes the vector  $[\xi_1, \xi_2, \dots, \xi_S]$ .

input is  $j$ , if  $\mathcal{F}_j = \max_{k=1,2,\dots,M} \{\mathcal{F}_k\}$ . The training of the modular network is comprised of the following two stages:

#### 7.3.2.1 Training of subnetworks

For this stage, separate data sets are prepared to train the subnetworks independently. The training data set for a subnetwork generally consists of the patterns belonging to the classes in its subgroup only. Then, each subnetwork is trained to form the decision surfaces for the classes in its subgroup. Training of the subnetworks varies depending on whether the network is FFNN or FRNN. Both the training strategies are discussed in chapter 5 and chapter 6.

### 7.3.2.2 Pattern matching

This stage of training is needed to compare the  $k$ th class prototype and the feature vector  $\psi$ . Here the partial evaluation  $h_k(\xi_s)$  implies how good the feature  $\xi_s$  alone is to classify the patterns from the class  $C_k$ , and the individual fuzzy density  $g_k(\{\xi_s\})$  signifies the importance of the feature  $\xi_s$  for the class  $C_k$ . Hence, the comparison between the prototypes of  $C_k$  and  $\psi$  can be accomplished in terms of closeness. Roughly speaking, the closeness can be expressed as  $h_k(\xi_1)g_k(\{\xi_1\}) + h_k(\xi_2)g_k(\{\xi_2\}) + \dots + h_k(\xi_S)g_k(\{\xi_S\})$ . When the domain is continuous and the continuity of the function  $h_k(\xi_s)$  is not guaranteed, the closeness can be represented more comfortably by the fuzzy integral  $\mathcal{F}_k$ . Therefore, in this stage it is essential to know the values of: (a) class prototypes, from which the partial evaluation  $h_k(\xi_s)$  can be obtained, and (b) the individual fuzzy density  $g_k(\{\xi_s\})$ .

**Class prototype selection:** The set  $\{x\}$  that contains training inputs from all the classes, are passed through all the subnetworks to generate a set of feature vectors  $\{\psi\}$ . All the feature vectors corresponding to each class (say  $C_k$ ) are collected separately, and the mean of the vectors (say  $\mathbf{m}_k$ ) is calculated. This mean represents the class prototype. In the testing phase, these class prototypes will enable us to compute the partial evaluation  $h_k(\xi_s)$ .

**Evaluation of fuzzy density  $g_k(\{\xi_s\})$ :** The individual fuzzy densities are calculated based on how well the outputs generated by the subnetworks separate all the classes for the training data. Since we have already mentioned that each  $\xi_s$  can be considered as a feature, determining individual fuzzy densities are equivalent to the determination of the importance of each feature. We propose a fuzzy-rough set theoretic approach to determine the individual fuzzy densities, i.e., the importance of the features for a particular class. This approach is described below for the feature  $\xi_s$  and the output class  $C_k$ .

A set of features  $\{\xi_s\}$  is collected by passing a set of training inputs  $\{x\}$  through the  $s$ th subnetwork. Fuzzy K-means algorithm [Bez81] is applied on this feature set. Since the number of clusters is not known, we assume  $K$  is equal to the number of classes  $M$ . While applying the fuzzy K-means clustering on the set  $\{\xi_s\}$ , we can observe the following two points:

1. Some  $\xi_s$  belong to more than one cluster partially as the clusters are overlapping.
2. All  $\xi_s$  from the same cluster may not belong to the same class.

The first type of uncertainty is fuzzy uncertainty. It is generated because the outputs of

the subnetworks are not from  $\{0, 1\}$ . The second type of uncertainty is rough uncertainty. It is generated as the feature  $\xi_s$  is not sufficient to classify all the input patterns  $\{x\}$ . Hence, two different  $\xi_s$ , belonging to the same cluster may represent two different classes. Thus, the relationship between the sth feature and the class labels may be a one-to-many mapping. In other words, the classes are indiscernible or not distinguishable with respect to the sth feature. The sth feature  $\xi_s$  is an important feature if

1. The clusters are compact and wide apart. The less is the fuzzy uncertainty, the more important the feature is [PC86] [PM86].
2. All the elements from a particular cluster belong to the same class. The less is the rough uncertainty, the more important the feature is.

That is, the feature  $\xi_s$  is important if each cluster, generated by the feature, is compact and isolated, and if all the patterns from each cluster represent the same class. Therefore, the presence of more fuzzy and rough uncertainties implies less importance. Note that the presence of any one or both of these uncertainties change the importance of the feature for a particular class. We seek to measure the amount of fuzzy and rough uncertainties involved by using fuzzy-rough sets. Later we will use the quantified value to determine the importance of the sth feature for the kth class.

Based on the feature  $\xi_s$ , the approximation of  $C_k$  by the set of feature vectors  $\{\psi\}$  is expressed here as a fuzzy-rough set. The lack of discriminating power of the feature  $\xi_s$  is due to the fact that we are not considering the other features  $\xi_j$ ,  $j \neq s$ ,  $j = 1, 2, \dots, S$  into account. Here we do not have complete information to classify a particular pattern in the class  $C_k$  based on the information supplied by  $\xi_s$ . To determine the importance of the feature  $\xi_s$  for the class  $C_k$  with such incomplete knowledge, the concept of rough sets can be used. In the terminology of rough set, two patterns  $\psi_p \in \{\psi\}$  and  $\psi_q \in \{\psi\}$  are called indiscernible with respect to the sth feature when the sth component of these two patterns have the same value. Mathematically, it can be stated as

$$\psi_p R^s \psi_q \quad \text{iff} \quad \xi_{sp} = \xi_{sq} \quad (7.9)$$

where  $R^s$  is a binary relation over  $\{+\} \times \{\psi\}$ . Obviously,  $R^s$  is an equivalent relation. Therefore,  $R^s$  partitions  $\{\psi\}$  into a set of equivalent classes, namely  $\{F_1^s, F_2^s, \dots, F_K^s\}$ , where  $K$  is greater than one but less than the cardinality of  $\{\psi\}$ . For continuous features, it is better to consider that  $\psi_p$  and  $\psi_q$  are related if the sth component of the two features are similar (not necessarily strictly equal as in (7.9)). Two patterns from the same cluster

can be considered similar as they have spatial similarity. The resultant equivalence classes become fuzzy clusters. It can be proved [DP92] that the fuzzy clusters  $F_1^s, F_2^s, \dots, F_M^s$  will be present if and only if there exists some similarity relation like (7.9). Moreover, it can be shown that [DP92] the generated clusters will follow weak fuzzy partitioning [DP92]. This situation can be formulated in terms of fuzzy-rough sets. One obvious problem is to decide the number of clusters needed for the task. We are assuming that the number of clusters is equal to the number of classes, i.e.,  $K = M$ .

After showing that fuzzy-rough uncertainty is associated with each  $\xi_s$ , we have represented the approximation of  $C_k$  by  $\{\psi\}$  in terms of fuzzy-rough sets. Now we are ready to quantify the fuzzy-rough uncertainty associated with each  $\xi_s$ . In Appendix-D, we can observe that using fuzzy-rough membership values, we can measure the fuzzy-rough uncertainty associated with each input pattern. Now, a measure of fuzzy-roughness is needed to estimate the average ambiguity in the output class  $C_k$  for the input feature  $\xi_s$ . As a measure we use the concept of *fuzzy-rough entropy* for the sth feature and the kth class as

$$\mathcal{H}_k^s = -\frac{1}{\bar{n} \ln 2} \sum_{\xi_s} \left[ \tau_{C_k}(\xi_s) \ln(\tau_{C_k}(\xi_s)) + (1 - \tau_{C_k}(\xi_s)) \ln(1 - \tau_{C_k}(\xi_s)) \right] \quad (7.10)$$

where  $\tau_{C_k}(\xi_s)$  is the fuzzy-rough membership value of the feature  $\xi_s$  in the class  $C_k$  and  $\bar{n}$  is the number of feature vectors used to determine the importance of the feature. It can be noticed that  $\mathcal{H}_k^s$  increases monotonically in  $[0, 0.5]$  and decreases monotonically in  $[0.5, 1]$ . It reaches the maximum value when  $\tau_{C_k}(\xi_s) = 0.5 \forall \xi_s$ , and minimum value when  $\tau_{C_k}(\xi_s) = 0$  or  $1 \forall \xi_s$  [PB95]. The lower the value of  $\mathcal{H}_k^s$  is, the greater is the number of  $\xi_s$  having  $\tau_{C_k}(\xi_s) \approx 1$  or  $\tau_{C_k}(\xi_s) \approx 0$ , i.e., less is the difficulty in deciding whether  $\xi_s$  can be considered a member of  $C_k$  or not. In particular, when  $\tau_{C_k}(\xi_s) \approx 1$ , greater is the tendency of  $\xi_s$  to form a compact class  $C_k$  in the sth subspace, resulting in less internal scatter in the sth subspace. Moreover, when  $\tau_{C_k}(\xi_s) \approx 0$ ,  $\xi_s$  is far away from the kth class, and hence, the interclass distance increases in the sth subspace. On the other hand, when  $\tau_{C_k}(\xi_s) \approx 0.5$ ,  $\xi_s$  lies in between  $C_k$  and the other classes in the sth subspace. Hence, compactness and interclass distance both decrease in the sth subspace. Therefore, the reliability of  $\xi_s$ , in characterizing the class  $C_k$ , increases as the corresponding  $\mathcal{H}_k^s$  value decreases. Thus,  $\mathcal{H}_k^s$  quantifies the importance of  $\xi_s$  in characterizing the kth class. One way to determine the importance of the sth feature in the kth class is by the term

(1 - 'Xi). Hence, the fuzzy densities can be determined as

$$g_k(\{\xi_s\}) = 1 - \mathcal{H}_k^s \quad \forall s, k \quad (7.11)$$

The procedure to find the fuzzy density can be summarised as follows: We interpret the fuzzy density of a module with respect to an output class as the importance of the module for that class. It is equivalent to the importance of the feature generated by the module (since the module is treated as a feature extractor). The importance of the feature for an output class depends on the fuzzy-roughness associated with the output class for the given feature. We have demonstrated that a set of input patterns can be clustered based on the feature value, and as a consequence, the approximation of the output class by these clusters can be expressed in terms of a fuzzy-rough set. It is possible to quantify the fuzzy-roughness associated with each input pattern for the output class in terms of fuzzy-rough membership functions. The fuzzy-rough ambiguity associated with the output class for the given set of input patterns is measured using the fuzzy-rough entropy. The fuzzy density for the output class is determined from the fuzzy-rough entropy.

The complete training procedure, consisting of training the subnetworks and matching the patterns, is shown in Fig. 7.2.

### 7.3.3 Testing of Modular Networks

A separate set of test patterns is used as inputs to all the subnetworks. The outputs of all the subnetworks corresponding to the input test pattern  $x$  form the feature vector  $\psi = [\xi'_1, \xi'_2, \dots, \xi'_S]$ . To determine the partial evaluation  $h_k(\xi_s)$  from the already recorded class prototypes, we use the following relation [Bez81]:

$$h_k(\xi_s) = 1 / \sum_{h=1}^M (d_k/d_h)^{2/(q-1)} \quad (7.12)$$

where  $d_k$  is the distance between the feature  $\xi_s$  and the prototype of the  $k$ th class, i.e.,  $d_k = (\xi_s - \mathbf{m}_k^s) \Sigma^{-1} (\xi_s - \mathbf{m}_k^s)$ , with  $\mathbf{m}_k^s = [m_{c_{s-1}+1,k}, \dots, m_{c_s,k}]$ . Here,  $\Sigma$  is a positive definite matrix and  $q \in (1, \infty)$  is an index. Generally,  $\Sigma$  is taken as the covariance matrix for the distance between  $\xi_s$  and  $\mathbf{m}_k^s$ , and  $q$  is taken as 2. The value of  $h_k(\xi_s)$  is an indication of how certain we are in the classification of the input  $x$  into the class  $C_k$  using the feature  $\xi_s$ . Here, 1 indicates with absolute certainty that the input  $x$  is from the class  $C_k$ , and 0 means that the input certainly does not belong to the class  $C_k$ . Moreover, from the training the fuzzy densities  $g_k(\{\xi_s\})$ ,  $\forall s, k$ , are known. Hence, using

Use different training sets  $T_s$ ,  $s = 1, 2, \dots, S$  to train all the **subnetworks**. The training set  $T_s$  contains the training input-output pairs only for the sth subnetwork.

Prepare another training set  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{\tilde{n}}\}$  that contains the training Input-output pairs for all the subnetworks. Pass this training set through all the subnetworks to collect the feature vectors  $\psi_p$ ,  $p = 1, 2, \dots, \tilde{n}$  as the outputs.

DO for each  $k = 1, 2, \dots, M$

Record the class prototype  $\mathbf{m}_k$ .

END DO

DO for each  $s$

Apply the fuzzy K-means clustering algorithm with  $M$  clusters on  $\{\xi_{sp} | p = 1, 2, \dots, \tilde{n}\}$ .

DO for each class  $C_k$ ,  $k = 1, 2, \dots, M$

Use (7.11) to compute the fuzzy density  $g_k(\{\xi_s\})$  from  $\{\xi_{sp} | p = 1, 2, \dots, \tilde{n}\}$ .

END DO

END DO

Fig. 7.2: Training of the proposed modular neural network.



```

For the test input pattern  $\mathbf{x}$ , find the outputs  $\xi_s$ ,  $s = 1, 2, \dots, S$ 
for all the subnetworks.
DO for each output class  $C_k$ ,  $k = 1, 2, \dots, M$ 
    DO for each  $\xi_s$ ,  $s = 1, 2, \dots, S$ 
        Compute  $h_k(\xi_s)$  from (7.12) .
    END DO
    Calculate  $\lambda$  from (7.8).
    Calculate  $\mathcal{F}_k$  from (7.3).
END DO
The class label of  $\mathbf{x}$  is  $j$  if  $\mathcal{F}_j = \max_{k=1}^M \{\mathcal{F}_k\}$ .

```

**Fig. 7.3:** Testing of the proposed modular neural network.

Equation (7.3), the fuzzy integral value of  $\mathbf{x}$  corresponding to each output class can be computed. The class label corresponding to the test input is the class index which yields the maximum fuzzy integral value. The fuzzy integral value corresponding to a particular class can also be used as the confidence level in classifying the input to that class. The testing procedure is given in form of an algorithm in Fig. 7.3.

## 7.4 Results and Discussion

In chapter 6, we found that FRNNs are suitable for the first and second level bids, whereas the FFNN with fuzzy mean square error is suited for the third level bids. Hence to construct the modular network, we use two FRNNs for the first and second level bids and one FFNN for the third level bids. The architecture and training strategy for these networks are discussed in chapter 5 and 6. To combine the outputs of the networks, a test set of 600 input patterns was formed. It contains patterns from all the classes. For a given input hand, the output of each network is found. In the first experiment, we applied winner-take-all technique on these outputs to find the corresponding class labels. The class label of the input was chosen as the class label corresponding to the maximum output. The classification performance is shown in the second column of Table 7.1.

Next we apply the fuzzy integral to fuse the outputs of the three networks. To train the fuzzy integral, we used a validation set of size 400. This set contains data from all the classes. Here the importance of each  $\xi_s$  is determined using two different methods. One method is called *frequency-based method*. It was used first by Tahani *et al.* in [TK90]. The fuzzy density corresponding to each  $\xi_s$  is found based on how well this feature alone performs on the validation set. The fuzzy densities are calculated using [CK95b]

$$g_k(\{\xi_s\}) = P_{s,k} / \sum_{j=1}^S P_{j,k} \cdot d_s \quad (7.13)$$

where  $P_{s,k}$  is the classification performance of  $\xi_s$  for the class  $C_k$  on the validation data and  $d_s$  is the desired sum of the fuzzy densities. The output with maximum fuzzy integral value is chosen as the output class label. The classification efficiency using this procedure is depicted in the third column of Table 7.1. Finally the fuzzy densities are calculated using the proposed method. For each module, the class prototypes were recorded. Since the number of classes is 15, we used 15 clusters in the fuzzy K-means algorithm. This information was used to compute the fuzzy densities. The classification results of the proposed method on the same test set are given in the fourth column of Table 7.1.

In Table 7.1, we can observe that the proposed method is performing better than the winner-take-all method. In the winner-take-all method, there is a large variation in the performance among the classes. It is because some classes are highly trained, and hence these classes win for most of the input data. The sum of importance calculated using frequency based method is equal to one. Consequently, if a module is not efficient to classify patterns to any of the classes, then the importance associated with the module for all the classes will be  $\frac{1}{\text{no. of classes}} = \frac{1}{15}$ . On the otherhand, in the proposed method sum of importance may or may not be equal to one. Hence, if a module is not efficient to classify patterns to any of the classes, then the importance associated with the module for all the classes will be zero. This strategy is certainly more attractive than that of frequency-based method. Possibly because of this reason, the proposed method is performing better (overall) than the frequency-based method. Therefore, the better classification performance (overall), makes the proposed method more attractive compared to the other two methods.

In this chapter, finally we have built the complete modular network for the opening bid problem. Although the performance of the resultant bidding system is not the best, our aim was to show that exploitation of uncertainties can make the classifier better. For that we started with a monolithic classifier, which could not be trained. Then we broke the

Table 7.1: Final classification results for the opening bid problem using winner-take-all method, frequency method and proposed method.

	method	method	method
P	90.14%	78.87%	88.34%
1C	56.67%	89.43%	85.91%
1D	57.24%	68.89%	79.83%
1H	88.76%	88.94%	86.54%
1S	86.94%	77.76%	92.53%
1N	83.78%	82.90%	85.89%
2C	46.17%	69.91%	80.15%
2D	77.18%	86.18%	87.89%
2H	95.56%	84.56%	82.15%
2S	67.01%	85.93%	89.04%
2N	75.56%	83.85%	73.78%
3C	56.13%	76.78%	79.23%
3D	96.67%	84.74%	88.14%
3H	87.61%	87.45%	89.14%
3S	94.23%	86.89%	90.01%
Overall	77.31%	82.20%	85.23 %

monolithic classifier and finally integrated the classification results from the subclassifiers. The classification result by the network is off course subjective. It is because the bids produced by the system are tallied by an expert, and if it is accepted by the expert as a valid bid, then the output is considered as correct output. There may be slight change in the performance of the classifier if the expert is changed.

## 7.5 Summary

This chapter applies a fuzzy integral-based technique to combine the outputs of the modules in a modular neural networks. The modules are viewed as nonlinear feature extractors. Hence, for each input the modules generate a feature vector. The fuzzy integral acts here as a weighted closeness measure between the feature vector and the class prototypes.

The weights are determined based on how important the features are for a particular class. The importance of a feature for a particular class is measured in terms of fuzzy-rough ambiguity associated with the concerned output class for the given input feature. The class prototype that is the nearest to the feature vector is designated as the class label of the input pattern corresponding to the feature vector.

The approach adopted in this chapter can also be viewed as a two stage classification scheme [CK92]. The first stage of the classification scheme, accomplished by the subnetworks, is for crude classification. The second stage, which consists of the fuzzy integral, is to fine tune the classification results obtained from the first stage.

The attractive points about the proposed way of calculating the fuzzy densities are

1. It is an objective way of calculating the fuzzy densities. Therefore, it does not need any expert to determine the fuzzy densities. Moreover, unlike other objective approaches, it does not need any information regarding the probability of occurrence of the input patterns. It needs only the facts hidden inside the data.
2. It is conceptually simple and needs simple algorithm. It does not need any complicated learning procedure as used in [KO96] [WW97]. The learning procedures used in [KO96] [WW97] may get stuck in local minima, or may take long time to converge. Especially, if the number of modules are large, then the convergence of this kind of learning algorithm may become very tough [GN94].

In the definition of fuzzy integral, we are using  $\max$  and  $\min$  operators which are noninteractive. It makes the fuzzy integral less sensitive towards the training data. A better approach may be to use the fuzzy integral with OWA operators [Cho95].

# Chapter 8

## SUMMARY AND CONCLUSIONS

### 8.1 Summary of the Thesis

In this thesis, an attempt has been made to deal with uncertainties in classification problems. The objective is (a) to identify the roles of fuzzy, rough and probabilistic uncertainties associated with the given classification problem, and (b) to exploit the associated uncertainties to evolve a pattern classification methodology. Contract Bridge opening bid problem is considered as a case study. The aim is to construct a classifier for the opening bid problem based on the input-output pairs of the data collected from players of the Bridge game. When a hand pattern is presented as an input pattern, the classifier should be able to determine the opening bid. Some salient characteristics of the problem are: The input hand patterns are crisp, the output bids are fuzzy, some output bids are highly probable and the input-output relationship is not unique. Although the problem is complex, the straightforward input representation of the problem enables us to probe more into the classification mechanism.

Before going into the details of the opening bid problem, a comprehensive survey of different pattern classification techniques are presented. The emphasis of the review is on the recent trend to evolve pattern classification methodologies using uncertainties. The review includes the description of the state-of-the-art techniques employed in modular classifiers.

In this thesis, the classification process is described through numerical quantities. Feedforward neural networks (FFNNs) with backpropagation learning algorithm are chosen for the study. In the initial experiments, monolithic feedforward neural networks failed to converge. The reason may be that the classifier is insufficient to handle, resolve and exploit the uncertainties associated with the problem. To make the uncertainty handling easier, an attempt is made to break the the problem into smaller subproblems. The intention is to resolve and exploit the uncertainties locally in each subproblem, followed by a mechanism to treat the uncertainties globally. In order to accomplish it the following

five steps are adopted:

In the first step, all the possible classes are partitioned such that the classes that are close, and the classes for which the frequency of occurrence of the patterns are similar, belong to the same partition. The condition of closeness narrows down the effect of fuzzy uncertainty into a local region, and the condition of similar occurrence makes the learning easier. Each partition forms one subclassification problem. This strategy results in three subclassification problems in the opening bid problem. The output classes for the three subclassifiers correspond to the first, second and third level bids.

In the second step, different feature sets are used for each subclassification problem to increase and decrease the interclass and intraclass distances, respectively. The aim is to make the classification process easier with the derived feature set. It is accomplished by imposing higher weightage on the features that are important for the classes present in the subclassification process. While measuring the importance of a particular feature, influence of the other features present in the input pattern and influence of the unaccounted features are not possible to be taken under consideration. Consequently, two input patterns with the same feature value may be mapped to more than one class. This situation causes the input-output relation to be one-to-many, and hence, rough-uncertainty is generated. Moreover, the classification task involved in the opening bid problem is inherently fuzzy. The more rough and fuzzy uncertainties are associated with a feature, the less is the importance of the feature. Rough-fuzzy entropy is proposed as a criterion function to evaluate the importance of each feature. The fuzzy membership value of each training pattern is determined using possibilistic K-means algorithm. These membership values are used to compute the rough-fuzzy entropy. The rough-fuzzy entropy is minimized iteratively to obtain the optimal importance of each feature for a particular module.

In the third step, a classifier module is designed for each subclassification task. Each module is constructed using direct classification technique. Feedforward neural networks with backpropagation algorithm are used. The inputs of the networks are the modified feature vectors and the outputs are the fuzzy output classes. The backpropagation algorithm is designed to minimize two classes of fuzzy objective functions, namely, fuzzy mean square error and fuzzy cross entropy. The performance of these two algorithms are comparable on the opening bid problem. The generalization capability of the modules are still low. It is because (a) the number of weights and hidden nodes are not minimized, and (b) the training of each module may not be proper as the backpropagation algorithm may get stuck in local minima. In order to reduce these drawbacks, a stochastic learning

strategy using evolutionary programming is adopted in conjugation with the deterministic learning (BP) using fuzzy objective functions. In particular, two objective functions, viz. major (global) and minor (local) objective functions, are minimized simultaneously. The major objective function is the fuzzy mean square error value (or fuzzy cross entropy) over a validation set and the minor objective function is the fuzzy mean square error value (or fuzzy cross entropy) over a training set. Iterative minimization of the minor objective function is carried out to guide the minimization of the major objective function. The iterative procedure is made faster by dynamically adapting the mutation parameters that are used in evolutionary programming.

In another approach, each subclassification task is carried out through clustering. The modified feature vectors that form fuzzy clusters, are clustered using evolutionary programming. The clustering algorithm determines the number of clusters, cluster means and cluster variance automatically. Two objective functions are incorporated. The major objective function decides how many clusters should be there. The minor objective function decides the cluster parameters. The major objective function is minimized stochastically using evolutionary programming-based method, and the minor objective function is minimized using a deterministic iterative method. The resultant clusters are used to construct a fuzzy-rough neural network (FRNN). This network uses the fuzzy uncertainty present in the clusters and the rough uncertainty (due to one-to-many mapping between the clusters and the class labels) in terms of fuzzy-rough membership functions. Compared to the backpropagation algorithm with fuzzy objective functions, the classification performance of FRNNs is better for the modules that deal with the first and second level bids, and worse for the module that deals with the third level bids. Hence, the FRNNs are used for the first and second modules and the FFNN is used for the third module.

In the fifth step, the result of all the classifiers are combined using Sugeno's fuzzy integral. All the modules are supposed to resolve or exploit the fuzzy and rough uncertainties locally. For any input, each module claims that the input can be classified by that module alone. Consequently, each module provides some classification result. A postprocessor is used to determine the output class from such a collection of conflicting evidence. The evidence are aggregated in a nonlinear fashion, and each evidence is weighted differently. To find the weightage associated with each module, the amount of fuzzy and rough uncertainties associated with each module is quantified from global angle. The concept of fuzzy-rough membership functions is used for the purpose of quantification. The outputs of the integrator, i.e., the outputs of the modular network, are the class confidence levels

corresponding to the input pattern.

## **8.2 Contribution of the Thesis**

The contributions of the thesis are

1. Providing an in-depth review on the pattern classification techniques that exploit uncertainty (chapter 2).
2. Formulating the opening bid problem as a pattern classification problem, and addressing different types of uncertainties involved in this problem (section 2.4).
3. Application of modular neural networks to deal with fuzzy, rough and probabilistic uncertainties effectively (chapter 3).
4. Use of possibilistic K-means algorithm to determine the possibilistic membership values of the training inputs (section 4.3.2).
5. Use of rough-fuzzy sets to determine the importance of each feature for classification (chapter 4).
6. Development of backpropagation learning algorithm based on various fuzzy objective functions (section 5.2).
7. Enhancement of the dynamics of evolutionary programming and use of this technique to configure feedforward neural networks (section 5.3).
8. Devising a framework to embed fuzzy clustering algorithms in evolutionary programming paradigm (section 6.2).
9. Introducing the concepts of rough-fuzzy and fuzzy-rough membership functions for classification (Appendix C and D).
10. Proposing fuzzy-rough neural networks to consider the fuzziness as well as the roughness present in the classification problem (section 6.3).
11. Use of fuzzy-rough sets in fuzzy integral to measure the importance of each module (chapter 7).

## **8.3 Conclusion of the Thesis**

The following are some important conclusion of the study:



1. Uncertainties, which apparently **affect** the classification system, can be made useful to the classification system, if treated properly.
2. It is possible to employ the uncertainties associated with the given classification problem to evolve one among many possible solutions for the classification problem.
3. We should be careful not to add large amount of uncertainty while representing the problem. If the representation is straightforward, it becomes easier to analyse the classification task.
4. An attractive way to exploit the uncertainties is to divide the given classification task into simple subtasks, and then combine the individual solutions of the subtasks. The uncertainties in each **subtask** can be considered locally, and the uncertainties in the whole problem can be treated globally while combining the solutions.
5. **Modularisation** in an arbitrary manner may not enable us to construct a good classifier. Because modularisation adds its own uncertainty, which should be handled with care. If we can exploit the uncertainty involved in the original problem, slight increase in the uncertainty due to modularisation may get nullified. Eventually the modularisation may be beneficial to us.
6. When the classification relation is based on subjective data, as we observe in the case of bidding, it is difficult to model the classification system.
7. It is difficult to model a part of the system, when the correlation between the various parts of the system is very high. For example, while building a bidding system, if we do not consider some important aspect like "vulnerability", then the problem becomes difficult.
8. Although opening bid problem is taken as an illustration, problems dealing with uncertainties are common in vision, speech and natural language processing.

#### **8.4 Issues Related to This Thesis Work for Further Study**

In this thesis we have partitioned the classification task based on some prior knowledge. In many classification problems, the domain specific knowledge may be absent or difficult to obtain. It is worthwhile to explore automatic partitioning of the input space so that the overall generalization capability of the whole network is increased.

Bridge players gather their experience from data as well as from some weak rules or knowledge. We have exploited only the data to construct the classification system. The

classification performance could be enhanced if we had used data and rules simultaneously.

We have used possibilistic K-means to obtain the membership values of the input hands. A better approach is to learn the classification function and membership functions simultaneously. In [PK96], this problem is attempted, where each class contains only one cluster. For a general framework, where each class contains more than one cluster, virtually no progress is achieved so far.

The importance of each feature is determined by minimizing the rough-fuzzy entropy iteratively. Rigorous proof is needed to show that this iterative scheme always converges.

Human beings can remember some accidental events and generalize many regular events. In the bidding problem also, players remember the high level bids and **generalize** the lower level bids. The current model cannot handle this *memorisation-generalization* dilemma. It will-be an interesting topic to explore how the memorisation-generalization dilemma can be realized in modular networks.

The feedforward neural networks with fuzzy objective functions, reported in chapter 5, do not capture the rough uncertainty. Hence, it will be interesting to train the network to capture fuzzy as well as rough uncertainties.

We have enhanced the dynamics of evolutionary programming based on some empirical evidence. A more rigorous analysis is needed to analyse the efficiency of the proposed technique. One useful tool for this analysis can be Markov chain.

We have used fuzzy hypervolume to decide the optimal number of clusters. In some cases, it does not give the optimal result. Therefore, we need to develop a better measure to identify the optimal number of clusters.

Although in literature attempts have been made to quantify the generalization capability of classifiers [Vid97], no significant work has been reported to quantify the generalization capability of modular networks.

In statistical pattern recognition, Bayesian classifiers are accepted as benchmarks to compare other classification techniques. Till now no such classifier is developed whose classification efficiency is optimum in presence of fuzzy, rough and probabilistic uncertainties. This kind of model may not be applicable in practice, but it can serve as a benchmark.

# APPENDIX A

## CONTRACT BRIDGE GAME: ISSUES

The game of Contract Bridge offers a rich platform for exploring theories in artificial intelligence. We observe that unlike chess, which is a two-person zero-sum complete-information game, Bridge cannot be tackled by elegant mechanisms like minimax search method [LS95]. This is because Bridge is not a complete information game. Since one does not know the cards held by the opponents, one cannot project the play into the future to try and discover which strategy is most profitable. Instead one has to rely on some knowledge intensive method. Bridge can be classified as a two-side incomplete-information game [LS95]. Further complication is introduced by the fact that each side constitutes of two persons. Therefore communication is vital. Not only does one have to convey information, within the rules of the game of course, to the partner, but one also needs to intercept opponents messages to learn their intentions. Almost as a corollary, at a more sophisticated level one may even want to send out misleading signals to lead opponents astray.

Contract Bridge is played with a regular pack of 52 cards dealt randomly and equally among 4 players. Let us call them North, South, East and West, according to their position on the table. North and South are partners, as are East and West. The cards are ranked in the order Ace, King, Queen, Jack, 10, 9, ..., 2 in each suit. Each player plays a card, in clockwise order, and the highest ranking card wins the trick, then it wins some points. Thirteen such tricks are played, and each time the winner of the precious trick starts play. This constitutes one *deal* or one *hand*.

There are two stages of play in each deal, viz. bidding, followed by the play of cards. The goal in a deal is to maximize *points*. The points essentially depend upon bidding. Bids are made for the number of tricks the side promises to make, given the stated "trump" suit. Eventually the highest bid is accepted in each deal. This is known as the *contract*. Generally, the higher a side bids the more points it is likely to win, provided it can fulfill the contract. That is, if the side can make the number of tricks it has bid for. If it succeeds, it wins some points. Let us call them *success-points*. If it loses, then the

opponents get some points instead, which we can call *penalty-points*.

The straightforward goal in bidding is to bid the highest number of tricks one thinks the side can make. That is, to maximize success-points won. The means used in this process are the following:

1. Evaluation of own hand.
2. Communication with partner.
3. Projection of play.

Among these, the first two are simpler and can possibly be handled by heuristic methods. The third is more difficult, as it would involve constructing plausible distributions (based on the bids heard, and on probability) and then projecting the play. A more complex goal is to make a *sacrifice bid*. It essentially means intentional overbidding, over an opponent bid, with the hope that the penalty-points loss will be lesser than the opponents' expected success-points gain, thus being an overall gain. Even more complex goals are to sabotage the opponents communication. This may mean consuming the bidding space (jamming the communications channel), or even making "false" bids to confuse opponents. In the process, an enterprising planner may make an "advancesacrifice" to "push" the opponents higher than they can manage, or to escape with a lighter penalty. Considering that all these processes happen when the planner can see only one hand, one observes that bidding is probably a more difficult part of the game.

Once bidding is over, the goal for the play stage has been defined. One side has the contract, and is required to make the bid number of tricks. At this stage one player of the contracting side (called the *dummy*) exposes the cards to everybody, while the other (called the *declarer*) plans and executes the play. The opposing side (called *defenders*) are said to *defend* the contract. They are in fact trying to defeat the accomplishment of the contract by the declarer.

One can observe that the situation at this stage is not symmetric. The declarer knows the entire strength of his side, and is in total control of the play of the cards. He is also aware of the entire assets of the defense, in terms of material strength, since they have the remaining 26 cards. Each defender knows only his own hand, and cannot see his partner's hand. Therefore the two defenders have to combine their efforts to try and achieve the goal. This necessarily involves (formal) communication between the two. Both can see the dummy also.

Since the cards of all the players cannot be seen, one cannot project moves into the future. Methods like minimax search are therefore ruled out immediately. Instead, the success of a strategy can only be estimated based on the probabilistic distribution of the cards, and any information gleaned from the communication taking place. The strategies themselves are derived from knowledge about the various known methods of tackling various card combinations.

The straightforward goal in the play of the hand is to make the number of tricks as stated in the contract. The emphasis is on maximising the probability of success. If success is assured, then the goal can be revised to increase the number of tricks won, as some more points can then be gained. If success seems unlikely, then a planner may even choose to minimize losses, i.e., the penalty-points won by the opponents. Like in bidding, the planner may attempt to do better by exploiting the incomplete information that the opponents have. This may introduce complex "meta-level" goals of protecting information, or sending out misleading signals.

Thus, we observe that **unlike** games like chess, where a clear cut strategy of aiming for the minimax value (saddle) points is meaningful, in Bridge one has to largely grapple with incomplete information. In the face of such uncertainty, planning in the game of Bridge can only be a complex knowledge intensive activity.

# APPENDIX B

## EVOLUTIONARY PROGRAMMING AND ROUGH SETS: BASICS

### B.1 Background of Evolutionary Programming

Usually an optimization problem seeks to find the value of a free parameter  $\mathbf{x} \in X$  of the system under consideration, such that a certain quality function  $\mathcal{G} : X \rightarrow \mathfrak{R}$  is minimised (or, equivalently maximised) [BHS97]. This quality function is known as *objective function*. The goal of the minimisation operation is to find  $\mathbf{x}$  corresponding to the global minimum of the objective function. But the presence of local minima, constraints and the other factors like large dimensionality, nonlinearity, nondifferentiability, noisy objective function make the optimization task difficult. If an optimization method can give a solution of  $\mathbf{x}$  which is slightly better than the currently known best solution of  $\mathbf{x}$ , then it is often accepted as a success. The efficiency of the optimization process can be enhanced, if it is carried out in parallel. One such biologically inspired method is *evolutionary programming* [Fog94b] [Fog95], where a population of solutions are probabilistically explored over a sequence of generations to reach the globally optimum solution. Evolutionary programming employs the following steps to find the global minimum of a function  $\mathcal{G}(\mathbf{x}) : \mathfrak{R}^N \rightarrow \mathfrak{R}$ :

1. Initially a population of parent vectors  $\mathbf{x}_i, i=1, 2, \dots, \nu$ , is selected at random (uniformly) from a feasible range in each dimension.
2. An offspring vector  $\hat{\mathbf{x}}_i, i = 1, 2, \dots, \nu$ , is created from each parent  $\mathbf{x}_i$ , by adding a Gaussian random variable with zero mean and predefined standard deviation to each component of  $\mathbf{x}_i$ .
3. A selection procedure then compares the values  $\mathcal{G}(\mathbf{x}_i)$  and  $\mathcal{G}(\hat{\mathbf{x}}_i)$  to determine which of these vectors are to be retained. The  $\nu$  vectors that possess the least value of the objective function become the parents for the new generation.

4. Go to the step 2 unless a satisfactory solution is reached or the number of generations is greater than some prespecified constant.
5. The solution of the problem is  $\mathbf{x}^*$ , where  $\mathcal{G}(\mathbf{x}^*)$  posses the least value in the final population.

## B.2 Background of Rough Sets

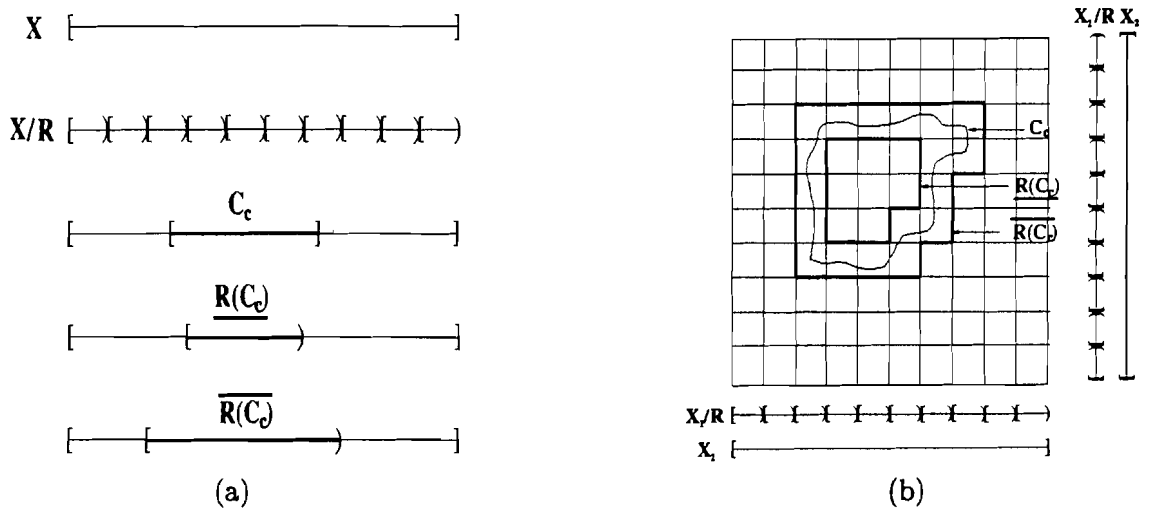
In any classification task the aim is to form various classes where each class contains objects that are not noticeably different. These *indiscernible* or indistinguishable objects can be viewed as basic building blocks (concepts) used to build up a knowledge base about the real world. For example, if the objects are classified according to 'color (red, black) and shape (triangle, square and circle), then the classes are: red triangles, black squares, red circles, etc. Thus, these two attributes make a *partition* in the set of objects and the universe becomes coarse. If two red triangles with different areas belong to different classes, it is impossible for anyone to correctly classify these two red triangles based on the given two attributes. This kind of uncertainty is referred to as *rough uncertainty* [Paw82] [PBSZ95]. The rough uncertainty is formulated in terms of *rough sets* [Paw91]. Obviously, the rough uncertainty can be completely avoided if we can successfully extract the essential features so that distinct feature vectors are used to represent different objects. But it may not be possible to guarantee as our knowledge about the system generating the data is limited.

In any classification problem, two input training patterns  $\mathbf{x}_u$  and  $\mathbf{x}_v$  (where  $\mathbf{x}_u, \mathbf{x}_v \in X$ , the set of all input patterns) are called *indiscernible* with respect to the sth feature, when the sth component of these two patterns have the same value. Mathematically, this indiscernibility can be represented as  $\mathbf{x}_u R^s \mathbf{x}_v$  iff  $x_{us} = x_{vs}$ , where  $R^s$  is a binary relation over  $X \times X$ . Obviously,  $R^s$  is an equivalence relation that partitions the universal set  $X$  into different equivalence classes. This idea can be generalized to take some or all the features into our consideration. Without loss of generality, based on a particular set of features, let  $R$  be an equivalence relation on the universal set  $X$ . Moreover, let  $X/R$  denote the family of all the equivalence classes induced on  $X$  by  $R$ . One such equivalence class in  $X/R$  that contains  $\mathbf{x} \in X$ , is designated by  $[\mathbf{x}]_R$ . In any classification problem, the objective is to approximate the given output class  $C_c \subseteq X$  by  $X/R$ . For the output class  $C_c$ , we can define lower approximation  $\underline{R}(C_c)$  and upper approximation  $\overline{R}(C_c)$ , which approach  $C_c$  as closely as possible from inside and outside, respectively [KY95]. Here,  $\underline{R}(C_c) = \cup\{[\mathbf{x}]_R \mid [\mathbf{x}]_R \subseteq C_c, \mathbf{x} \in X\}$  is the union of all the equivalence classes in  $X/R$

that are contained in  $C_c$ , and  $\overline{R}(C_c) = \cup\{[x]_R \mid [x]_R \cap C_c \neq \phi, x \in X\}$  is the union of all the equivalence classes in  $X/R$  that overlap with  $C_c$ . A rough set  $R(C_c) = \langle \overline{R}(C_c), \underline{R}(C_c) \rangle$  is a representation of the given set  $C_c$  by  $\underline{R}(C_c)$  and  $\overline{R}(C_c)$ . The set difference,  $\overline{R}(C_c) - \underline{R}(C_c)$ , is a rough description of the boundary of  $C_c$  by the equivalence classes of  $X/R$ . The approximation is rough uncertainty free if  $\overline{R}(C_c) = \underline{R}(C_c)$ . When all the patterns from an equivalence class do not carry the same output class labels, rough ambiguity is generated as a manifestation of the **one-to-many** relationship between the equivalent class and the output class labels. For a given  $C_c$  representing certain *concept* of interest, we can characterize  $X/R$  with the following three distinct regions:

1.  $\underline{R}(C_c)$  is called the *positive region*  $POS_R(C_c)$  of  $C_c$ ,
2.  $\overline{R}(C_c) - \underline{R}(C_c)$  is called the *boundary region*  $BND_R(C_c)$  of  $C_c$ ,
3.  $X/R - \overline{R}(C_c)$  is called the *negative region*  $NEG_R(C_c)$  of  $C_c$ .

Two examples of rough sets are shown in Fig. B.1. In the first example (Fig. B.1(a)),  $X$  is a closed interval of real numbers, and  $X/R$  partitions  $X$  into ten semiclosed intervals and one closed interval. The output class  $C_c$ , which is to be approximated by the elements of  $X/R$ , is the closed interval shown in this figure. The rough set approximation of  $C_c$  consists of the two semiclosed intervals,  $\overline{R}(C_c)$  and  $\underline{R}(C_c)$ . In the second example (Fig. B.1(b)), the universal set is  $X = X_1 \times X_2$ , and the equivalence relation  $R$  partitions  $X_1 \times X_2$  into one hundred small squares.



**Fig. B.1:** Rough sets in (a) one and (b) two dimensional domains.



In a classification task, the concept of *rough membership function* is introduced [WZ87] to quantify the rough uncertainty associated with each pattern. The rough membership function  $r_{C_c}(x): X \rightarrow [0, 1]$  of a pattern  $\mathbf{x} \in X$  for the output class  $C_c$  is defined by

$$r_{C_c}(\mathbf{x}) = \frac{|[\mathbf{x}]_R \cap C_c|}{|[\mathbf{x}]_R|} \quad (\text{B.1})$$

where  $|C_c|$  denotes the cardinality of the set  $C_c$ . Rough membership function  $r_{C_c}(\mathbf{x})$  signifies the rough uncertainty associated with the pattern  $\mathbf{x}$  for the output class  $C_c$ . It can be shown that  $r_{C_c}(x) = 0$  or 1 if and only if there is no rough uncertainty associated with the pattern  $\mathbf{x}$  [Paw95] [Paw94]. Evidently, the rough uncertainty associated with  $\mathbf{x}$  is maximum when  $r_{C_c}(x) = 0.5$ .

# APPENDIX C

## ROUGH-FUZZY MEMBERSHIP FUNCTIONS

In a classification task, the indiscernibility relation partitions the input pattern set to form equivalence classes. These equivalence classes try to approximate the given output class. When this approximation is not proper, roughness is generated. The output classes may have fuzzy boundaries. Thus, both roughness and fuzziness appear due to the indiscernibility relation in the input pattern set and the vagueness in the output class, respectively. To model this type of situation, where both vagueness and approximation are present, the concept of rough-fuzzy set [DP90] is proposed. The resultant model is expected to be more powerful than either of rough sets or fuzzy sets.

This appendix provides one scheme to generalize the concept of rough membership functions in pattern classification tasks to rough-fuzzy membership functions. Unlike the rough membership value of a pattern, which is sensitive only towards the rough uncertainty associated with the pattern, the rough-fuzzy membership value of the pattern signifies the rough uncertainty as well as the fuzzy uncertainty associated with the pattern. In absence of fuzziness in the output class, the rough-fuzzy membership function reduces to the original rough membership function. Moreover, when the partitioning in the input set is fine, i.e., each equivalent class contains only one pattern, the rough-fuzzy membership function turns out to be the fuzzy membership function. If the partitioning is fine and the output classes are crisp simultaneously, the rough-fuzzy membership function reduces to the characteristic function. In this appendix, various set theoretic properties of the rough-fuzzy membership functions are discussed. A detail discussion on rough-fuzzy membership functions can be found in [SY].

### C.1 Basics of Rough-Fuzzy Sets

Let  $X$  be a set,  $R$  be an equivalence relation defined on  $X$  and the output class  $C_c \subseteq X$  be a fuzzy set. A rough-fuzzy set is a tuple  $\langle \bar{R}(C_c), \underline{R}(C_c) \rangle$ , where the lower approximation  $\underline{R}(C_c)$  and the upper approximation  $\bar{R}(C_c)$  of  $C_c$  are fuzzy sets of  $X/R$ , with membership functions defined by [DP92]

$$\mu_{\underline{R}(C_c)}([x]_R) = \inf \{ \mu_{C_c}(x) \mid x \in [x]_R \} \quad \forall x \in X \quad (C.1-a)$$

$$\mu_{\bar{R}(C_c)}([x]_R) = \sup \{ \mu_{C_c}(x) \mid x \in [x]_R \} \quad \forall x \in X \quad (C.1-b)$$

$$(C.1-c)$$

Here,  $\mu_{\underline{R}(C_c)}([x]_R)$  and  $\mu_{\bar{R}(C_c)}([x]_R)$  are the membership values of  $[x]_R$  in  $\underline{R}(C_c)$  and  $\bar{R}(C_c)$ , respectively.

### C.2 Definition of Rough-Fuzzy Membership Functions

The rough-fuzzy membership function of a pattern  $x \in X$  for the fuzzy output class  $C_c \subseteq X$  is defined by

$$\iota_{C_c}(x) = \frac{|F \cap C_c|}{|F|} \quad (C.2)$$

where  $F = [x]_R$  and  $|C_c|$  means the cardinality of the fuzzy set  $C_c$ . One possible way to determine the cardinality is to use [Zad78]:  $|C_c| \stackrel{def}{=} \sum_{x \in X} \mu_{C_c}(x)$ . For the ' $\cap$ ' (intersection) operation, we can use  $\mu_{A \cap B}(x) \stackrel{def}{=} \min\{\mu_A(x), \mu_B(x)\} \quad \forall x \in X$ . It must be noted that the concept of rough-fuzzy set is necessary while dealing with ambiguous concepts, whereas the rough-fuzzy membership function is needed when uncertain data are considered.

### C.3 Properties of Rough-Fuzzy Membership Functions

Following are a few important properties of rough-fuzzy membership functions that can be exploited for a classification task.

**Property C.1:**  $0 \leq \iota_{C_c}(x) \leq 1$

**Proof.** Since  $\emptyset \subseteq F \cap C_c \subseteq F$ , the proof is trivial. ■

**Property C.2:**  $\iota_{C_c}(x) = 1$  and 0 if and only if no rough-fuzzy uncertainty is associated with the pattern  $x$ .

**Proof.**

**If part:** If no rough-fuzzy uncertainty is involved, then either (a)  $F \subseteq C_c$ , i.e.,  $\iota_{C_c} = 1$ , or (b)  $F \cap C_c = \emptyset$ , i.e.,  $\iota_{C_c} = 0$ .

**Only if part:** If  $\iota_{C_c}(\mathbf{x}) = 0$ , then the numerator of (C.2) is zero. It implies that  $F \cap C_c = \emptyset$ . On the other hand, if  $\iota_{C_c}(\mathbf{x}) = 1$ , then the numerator of (C.2) is equal to the denominator. It means that  $F \cap C_c = C_c$ , i.e.,  $F \subseteq C_c$ . Both cases imply that no rough-fuzzy uncertainty is involved. ■

**Property C.3:** When the output class  $C_c$  is crisp,  $\iota_{C_c}(\mathbf{x}) = \tau_{C_c}(\mathbf{x})$ .

**Proof.** When the output class  $C_c$  is crisp, Equation (C.2) reduces to (B.1). Hence, the proof follows. ■

**Property C.4:** When the partitioning is fine,  $\mathbb{L}C, (\sim) = \mathbb{P}C, (\sim)$ . Moreover, if the partitioning is fine and the output class  $C_c$  is crisp, then  $\iota_{C_c}(\mathbf{x})$  is equivalent to the characteristic function.

**Proof.** When the partitioning is fine, i.e., each  $F$  consists of a single pattern,  $\iota_{C_c}(\mathbf{x}) = \frac{|\mathbb{L}\mu_{C_c}(\mathbf{x})|}{1} = \mu_{C_c}(\mathbf{x})$ . If  $\mu_{C_c}(\mathbf{x}) \in \{0, 1\}$ , i.e., the output class is crisp, then  $\iota_{C_c}(\mathbf{x})$  becomes the characteristic function. ■

This property and the property C.3 show that both rough and fuzzy membership functions become particular cases of rough-fuzzy membership functions in the absence of fuzziness and, roughness, respectively.

**Property C.5:**  $\iota_{X-C_c}(\mathbf{x}) = 1 - \iota_{C_c}(\mathbf{x})$

**Proof\***  $\iota_{X-C_c}(\mathbf{x}) = \frac{|F \cap (X - C_c)|}{|F|} = 1 - \frac{|F \cap C_c|}{|F|} = 1 - \iota_{C_c}(\mathbf{x})$ . ■

**Property C.6:** If  $\mathbf{x}$  and  $\mathbf{z}$  are two input patterns so that  $\mathbf{x}R\mathbf{z}$  (i.e.,  $\mathbf{x}, \mathbf{z} \in F$ ), then  $\iota_{C_c}(\mathbf{x}) = \iota_{C_c}(\mathbf{z})$ .

**Proof.** It can be derived directly from Equation (C.2). ■

**Property C.7:**  $\iota_{A \cup B}(\mathbf{x}) \geq \max\{\iota_A(\mathbf{x}), \iota_B(\mathbf{x})\}$ , where  $A, B \subseteq X$

**Proof.**  $\iota_{A \cup B}(\mathbf{x}) = \frac{|F \cap (A \cup B)|}{|F|} \geq \frac{|F \cap A|}{|F|} = \iota_A(\mathbf{x})$ . Similarly,  $\iota_{A \cup B}(\mathbf{x}) \geq \iota_B(\mathbf{x})$  ■

**Property C.8:**  $\iota_{A \cap B}(\mathbf{x}) \leq \min\{\iota_A(\mathbf{x}), \iota_B(\mathbf{x})\}$  where  $A, B \subseteq X$ .

**Proof.**  $\iota_{A \cap B}(\mathbf{x}) = \frac{|F \cap (A \cap B)|}{|F|} \leq \frac{|F \cap A|}{|F|} = \iota_A(\mathbf{x})$ . Similarly,  $\iota_{A \cap B}(\mathbf{x}) \leq \iota_B(\mathbf{x})$  ■

**Property C.9:** If  $Z$  is a family of pairwise disjoint subsets of  $X$ , then  $\iota_{\cup Z}(\mathbf{x}) = \sum_{C \in Z} \iota_{C_c}(\mathbf{x})$ .

**Proof.**  $\iota_{\cup Z}(\mathbf{x}) = \frac{|F \cap (\cup Z)|}{|F|} = \frac{|\cup (F \cap Z)|}{|F|} = \sum_{C_c \in Z} \iota_{C_c}(\mathbf{x})$  ■

**Property C.10:** For a  $C$ -class classification problem, rough-fuzzy membership function of a pattern behaves in a possibilistic way provided the fuzzy membership function of the pattern to the output classes is possibilistic.

**Proof.**

$$\begin{aligned}
 \sum_{c=1}^C \iota_{C_c}(\mathbf{x}) &= \sum_{c=1}^C \frac{|F \cap C_c|}{|F|} \\
 &= \frac{\sum_{c=1}^C \sum_{\mathbf{x} \in X} \min\{\mu_F(\mathbf{x}), \mu_{C_c}(\mathbf{x})\}}{|F|} \\
 &= \frac{\sum_{c=1}^C \sum_{\mathbf{x} \in F} \min\{1, \mu_{C_c}(\mathbf{x})\}}{|F|} \\
 &= \frac{\sum_{c=1}^C \sum_{\mathbf{x} \in F} \mu_{C_c}(\mathbf{x})}{|F|} \\
 &= \frac{\sum_{\mathbf{x} \in F}}{|F|} \left( \sum_{c=1}^C \mu_{C_c}(\mathbf{x}) \right) \\
 &= \sum_{c=1}^C \mu_{C_c}(\mathbf{x}) \tag{C.3}
 \end{aligned}$$

Therefore, for crisp and constrained fuzzy classification [PB95], where  $\sum_{c=1}^C \mu_{C_c}(\mathbf{x}) = 1$ , the value of  $\sum_{c=1}^C \iota_{C_c}(\mathbf{x})$  is equal to one. In case of possibilistic classification [PB95]  $0 \leq \sum_{c=1}^C \mu_{C_c}(\mathbf{x}) \leq C$ , and hence,  $0 \leq \sum_{c=1}^C \iota_{C_c}(\mathbf{x}) \leq C$ . Therefore,  $\iota_{C_c}(\mathbf{x})$  behaves in a possibilistic manner. ■

**Property C.11:** For crisp output classes

$$\underline{R}(C_c) = \{\mathbf{x} \in X \mid \iota_{C_c}(\mathbf{x}) = 1\} \tag{C.4-a}$$

$$\overline{R}(C_c) = \{\mathbf{x} \in X \mid \iota_{C_c}(\mathbf{x}) > 0\} \tag{C.4-b}$$

$$BN(C_c) = \overline{R}(C_c) - \underline{R}(C_c) = \{\mathbf{x} \in X \mid 0 < \iota_{C_c}(\mathbf{x}) < 1\} \tag{C.4-c}$$

**Proof.** For the crisp output classes, the above results come directly from Equation (C.1-a) and (C.1-b). ■

Following is a trivial but interesting definition based on the above properties:

A  $C$ -class classification problem for a set of input patterns  $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  is basically an assignment of the rough-fuzzy membership value  $\iota_{C_c}(\mathbf{x}_i)$  on each  $\mathbf{x}_i \in X$ ,  $\forall c = 1, 2, \dots, C$ ,  $\forall i = 1, 2, \dots, n$ . In the rough-fuzzy context,  $C$  partitions of  $X$  are the set of values  $\{\iota_{C_c}(\mathbf{x}_i)\}$  that can be conveniently arranged on a  $C \times n$  matrix  $[\iota_{C_c}(\mathbf{x}_i)]$ . Based on the characteristic of  $[\iota_{C_c}(\mathbf{x}_i)]$ , classification can be of the following three types [PB95]:

(a) **Crisp classification:**

$$B_{hc} = \left\{ [\iota_{C_c}(\mathbf{x}_i)] \in \mathbb{R}^{Cn} \mid \iota_{C_c}(\mathbf{x}_i) \in \{0, 1\} \forall c, \forall i; \sum_{c=1}^C \iota_{C_c}(\mathbf{x}_i) = 1; \right. \\ \left. 0 < \sum_{i=1}^n \iota_{C_c}(\mathbf{x}_i) < n \forall c \right\} \quad (\text{C.5-a})$$

(b) **Constrained rough-fuzzy classification:**

$$B_{fc} = \left\{ [\iota_{C_c}(\mathbf{x}_i)] \in \mathbb{R}^{Cn} \mid \iota_{C_c}(\mathbf{x}_i) \in [0, 1] \forall c, \forall i; \sum_{c=1}^C \iota_{C_c}(\mathbf{x}_i) = 1; \right. \\ \left. 0 < \sum_{i=1}^n \iota_{C_c}(\mathbf{x}_i) < n \forall c \right\} \quad (\text{C.5-b})$$

(c) **Possibilistic rough-fuzzy classification:**

$$B_{pc} = \left\{ [\iota_{C_c}(\mathbf{x}_i)] \in \mathbb{R}^{Cn} \mid \iota_{C_c}(\mathbf{x}_i) \in [0, 1] \forall c, \forall i; 0 < \sum_{i=1}^n \iota_{C_c}(\mathbf{x}_i) < n \forall c \right\} \quad (\text{C.5-c})$$

It is obvious that  $B_{hc} \subset B_{fc} \subset B_{pc}$ .

# APPENDIX D

## FUZZY-ROUGH MEMBERSHIP FUNCTIONS

In a classification task the indiscernibility relation, based on the equivalence of the features of the patterns, partitions the input pattern set into several equivalence classes. These equivalence classes try to approximate the given output class. When the approximation is not proper, the roughness is generated. In most of the real life cases, the value of a particular feature for two patterns may not be exactly same, but similar. Therefore, the indiscernibility relation formulated based on the features do not obey the law of equivalence, and is a matter of degree. Hence, the equivalence relation takes the form of a similarity relation, and the equivalence classes form fuzzy clusters. The situation becomes more complicated because the output classes can be fuzzy too. The roughness appears here due to the indiscernibility relation in the input pattern set, and the fuzziness is generated due to the vagueness present in the output class and the clusters. To model this type of situation, where both approximation and vagueness are present, the concept of *fuzzy-rough sets* [DP90] can be employed. The resultant model is expected to be more powerful than rough sets, fuzzy sets and rough-fuzzy sets.

In this appendix the concept of rough-fuzzy membership functions (see Equation (C.2)) in the classification tasks are generalized to *fuzzy-rough membership functions*. If the clusters are crisp, then fuzzy-rough membership functions are equivalent to rough-fuzzy membership functions. In absence of fuzziness, fuzzy-rough membership functions reduce to the existing rough membership functions. Moreover, under certain conditions fuzzy-rough membership functions are equivalent to fuzzy membership functions and characteristic functions. The concept of fuzzy-rough membership function becomes particularly attractive when we do not have complete knowledge about the human classification system, but we attempt to mimic the vagueness present in the human reasoning. In this appendix, various set theoretic properties of the fuzzy-rough membership functions are described. Details about fuzzy-rough membership functions is given in [SY98a]

## D.1 Background of Fuzzy-Rough Sets

When the equivalence classes are not crisp, they are in form of fuzzy clusters  $\{F_1, F_2, \dots, F_H\}$  generated by a *fuzzy weak partition* [DP92] of the input set  $X$ . The term fuzzy weak partition means that each  $F_j$  is a *normal fuzzy set* (i.e.,  $\max_{\mathbf{x}} \mu_{F_j}(\mathbf{x}) = 1$ ) and  $\inf_{\mathbf{x}} \max_j \mu_{F_j}(\mathbf{x}) > 0$  while

$$\sup_{\mathbf{x}} \min\{\mu_{F_i}(\mathbf{x}), \mu_{F_j}(\mathbf{x})\} < 1 \quad \forall i, j \in \{1, 2, \dots, H\} \quad (\text{D.1})$$

Here,  $\mu_{F_j}(\mathbf{x})$  is the fuzzy membership function of the pattern  $\mathbf{x}$  in the cluster  $F_j$ . In addition, the output classes  $C_c$ ,  $c = \{1, 2, \dots, C\}$  may be fuzzy too. Then the fuzzy set  $C_c$  can be described by means of the fuzzy partitions under the form of an upper and a lower approximation  $\overline{C_c}$  and  $\underline{C_c}$  as follows:

$$\mu_{\underline{C_c}}(F_j) = \inf_{\mathbf{x}} \max\{1 - \mu_{F_j}(\mathbf{x}), \mu_{C_c}(\mathbf{x})\} \quad \forall j \quad (\text{D.2-a})$$

$$\mu_{\overline{C_c}}(F_j) = \sup_{\mathbf{x}} \min\{\mu_{F_j}(\mathbf{x}), \mu_{C_c}(\mathbf{x})\} \quad \forall j \quad (\text{D.2-b})$$

The tuple  $\langle \underline{C_c}, \overline{C_c} \rangle$  is called a *fuzzy-rough set*. Here,  $\mu_{C_c}(\mathbf{x}) = \{0, 1\}$  is the fuzzy membership of the input  $\mathbf{x}$  to the class  $C_c$ . Fuzzy-roughness appears when a fuzzy cluster contains patterns that belong to more than one class.

## D.2 Definition of Fuzzy-Rough Membership Functions

The definition of rough-fuzzy membership function (Equation (C.2)) can be generalized to the following definition of fuzzy-rough membership function [SY98c]:

$$\tau_{C_c}(\mathbf{x}) = \begin{cases} \frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \iota_{C_c}^j(\mathbf{x}) & \text{if } \exists j \text{ with } \mu_{F_j}(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (\text{D.3})$$

where  $\hat{H} (\leq H)$  is the number of clusters in which  $\mathbf{x}$  has nonzero memberships and  $\iota_{C_c}^j(\mathbf{x}) = \frac{|F_j \cap C_c|}{|F_j|}$ . Here,  $\tau_{C_c}(\mathbf{x})$  represents the fuzzy-rough uncertainty of  $\mathbf{x}$  in the class  $C_c$ . When  $\mathbf{x}$  does not belong to any cluster,  $\hat{H}$  is equal to zero, and hence,  $\frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \iota_{C_c}^j(\mathbf{x})$  becomes undefined. In order to avoid this problem,  $\tau_{C_c}(\mathbf{x})$  is made equal to zero when  $\mathbf{x}$  does not belong to any cluster.

## D.3 Properties of Fuzzy-Rough Membership Functions

**Property D.1:**  $0 \leq \tau_{C_c}(\mathbf{x}) \leq 1$



**Proof.** Since  $\phi \subseteq F_j \cap C_c \subseteq F_j$ ,  $0 \leq \iota_{C_c}^j \leq 1$ . Moreover,  $0 \leq \mu_{F_j}(\mathbf{x}) \leq 1$ . Hence, the proof follows. ■

**Property D.2:**  $\tau_{C_c}(\mathbf{x}) = 1$  or 0 if and only if no fuzzy-rough uncertainty is associated with the pattern  $\mathbf{x}$ .

**Proof. If part:** If no fuzzy-rough uncertainty is involved, then  $\mathbf{x}$  must belong completely to all the clusters in which it has non-zero belongingness. It implies  $\mu_{F_j}(\mathbf{x}) = 1$  for which  $\mu_{F_j}(\mathbf{x}) > 0$ . Moreover, all the clusters in which  $\mathbf{x}$  has non-zero belongingness either (a) must be the subsets of the class  $C_c$ , or (b) must not share any pattern with the class  $C_c$ . In other words, the condition (a) implies that  $F_j \subseteq C_c \vee j$  for which  $\mu_{F_j}(\mathbf{x}) > 0$ . Hence,  $\tau_{C_c}(\mathbf{x}) = \frac{1}{H} \sum_{j=1}^H 1.1 = 1$ . Similarly the condition (b) expresses that  $F_j \cap C_c = \phi \vee j$  for which  $\mu_{F_j}(\mathbf{x}) > 0$ . Hence,  $\tau_{C_c}(\mathbf{x}) = \frac{1}{H} \sum_{j=1}^H \mu_{C_c}(\mathbf{x}).0 = 0$

**Only if part:**  $\tau_{C_c}(\mathbf{x}) = 0$  implies that either  $\mathbf{x}$  does not belong to any cluster, or each term under the summation symbol, i.e.,  $\mu_{F_j}(\mathbf{x})\iota_{C_c}^j$  is separately zero. In the first case, there is no fuzzy-roughness associated with  $\mathbf{x}$ . The second case implies that either  $\mu_{F_j}(\mathbf{x})$  or  $\iota_{C_c}^j$ , or both  $\mu_{F_j}(\mathbf{x})$  and  $\iota_{C_c}^j$  are zero. If  $\mu_{F_j}(\mathbf{x}) = 0$ , then the pattern  $\mathbf{x}$  does not belong to the cluster  $F_j$ , and hence, no fuzzy-rough uncertainty is associated with  $\mathbf{x}$ . If  $\iota_{C_c}^j = 0$ , then  $F_j$  and  $C_c$  do not have any pattern common, and therefore, no fuzzy-rough uncertainty exists with  $\mathbf{x}$ . Thus,  $\tau_{C_c}(\mathbf{x}) = 0$  implies that fuzzy-roughness is not associated with the pattern  $\mathbf{x}$ . If  $\tau_{C_c}(\mathbf{x}) = 1$ , then  $\mu_{F_j}(\mathbf{x}) = 1$  and  $\iota_{C_c}^j = 1$ ,  $\forall j = 1, 2, \dots, H$ . It also indicates the absence of fuzzy-roughness.

Note that if fuzzy-rough uncertainty is absent,  $H > 1$  and  $\tau_{C_c}(\mathbf{x}) \neq 0$ , then  $\tau_{C_c}(\mathbf{x})$  never becomes one, rather it approaches towards one. It is because, the condition expressed in (D.1) does not allow  $\mu_{F_j}(\mathbf{x}) = 1$  to be true for more than one cluster. However, it hardly happens in practice as it needs two cluster centers to be same. ■

**Property D.3:** If no fuzzy linguistic uncertainty is associated with the pattern  $\mathbf{x}$ , then  $\tau_{C_c}(\mathbf{x}) = \iota_{C_c}^j(\mathbf{x})$  for some  $j \in \{1, 2, \dots, H\}$ .

**Proof.** If no fuzzy linguistic uncertainty is involved, then  $\mu_{F_j}(\mathbf{x}) = 1$  for some  $j \in \{1, 2, \dots, H\}$ , and  $\mu_{F_k}(\mathbf{x}) = 0$  for  $k \in \{1, 2, \dots, H\}$ ,  $k \neq j$ . Hence,  $\tau_{C_c}(\mathbf{x}) = \iota_{C_c}^j$ ,  $j \in \{1, 2, \dots, H\}$ . ■

**Property D.4:** If no fuzzy linguistic and fuzzy classification uncertainties are associated with the pattern  $\mathbf{x}$ , then  $\tau_{C_c}(\mathbf{x}) = \tau_{C_c}(\mathbf{x})$ .

**Proof.** If no fuzzy linguistic uncertainty is involved, then each cluster is crisp. Consequently, the input pattern belongs to only one cluster. Let it be the  $j$ th cluster. Hence,  $\mu_{F_j}(\mathbf{x}) = 1$  and  $\mu_{F_k}(\mathbf{x}) = 0 \forall k \neq j$ . Since the classification is crisp,  $\tau_{C_c}(\mathbf{x}) = \frac{|F_j \cap C_c|}{|F_j|} = \tau_{C_c}(\mathbf{x})$  (see Equation (B.1)). ■

**Property D.5:** When each cluster is crisp and fine, that is, each cluster consists of a single pattern and the associated cluster memberships are crisp,  $\tau_{C_c}(\mathbf{x})$  is equivalent to the fuzzy membership function of  $\mathbf{x}$  in the class  $C_c$ . If the output class is also crisp, then  $\tau_{C_c}(\mathbf{x})$  is equivalent to the characteristic function of  $\mathbf{x}$  in the class  $C_c$ .

**Proof.** Since each cluster is crisp and fine,  $\tau_{C_c}(\mathbf{x}) = 1 \cdot \frac{\mu_{C_c}(\mathbf{x})}{1} = \mu_{C_c}(\mathbf{x})$ . In addition, if the output class is crisp, then  $\tau_{C_c}(\mathbf{x})$  lies in  $\{0, 1\}$ , and thus, it becomes the characteristic function. ■

**Property D.6:** For a  $C$ -class classification problem with crisp output classes, the fuzzy-rough membership functions behave in a possibilistic manner provided the fuzzy membership function of the pattern to the clusters is possibilistic.

**Proof.**

$$\begin{aligned}
\sum_{c=1}^C \tau_{C_c}(\mathbf{x}) &= \frac{1}{\hat{H}} \sum_{c=1}^C \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \frac{|F_j \cap C_c|}{|F_j|} \\
&= \frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \frac{\sum_{c=1}^C \sum_{\mathbf{z} \in X} \min\{\mu_{F_j}(\mathbf{z}), \mu_{C_c}(\mathbf{z})\}}{\sum_{\mathbf{z} \in X} \mu_{F_j}(\mathbf{z})} \\
&= \frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \frac{\sum_{c=1}^C \sum_{\mathbf{z} \in C_c} \min\{\mu_{F_j}(\mathbf{z}), 1\}}{\sum_{\mathbf{z} \in X} \mu_{F_j}(\mathbf{z})} \\
&= \frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \frac{\sum_{\mathbf{z} \in X} \mu_{F_j}(\mathbf{z})}{\sum_{\mathbf{z} \in X} \mu_{F_j}(\mathbf{z})} \\
&= \frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \tag{D.4}
\end{aligned}$$

Since  $\sum_{c=1}^C \tau_{C_c}(\mathbf{x})$  needs not to be equal to a constant, the resultant classification procedure is possibilistic [KY95] [PB95]. ■

**Property D.7:** If  $\mathbf{x}$  and  $\mathbf{z}$  are the two input patterns with  $\mu_{F_j}(\mathbf{x}) = \mu_{F_j}(\mathbf{z}) \forall j$  and  $\mu_{C_c}(\mathbf{x}) = \mu_{C_c}(\mathbf{z})$ , then  $\tau_{C_c}(\mathbf{x}) = \tau_{C_c}(\mathbf{z})$ .

**Proof.** Directly comes from Equation (D.3). ■

**Property D.8:**  $\tau_{X-C_c}(\mathbf{x}) = \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) - \tau_{C_c}(\mathbf{x})$ .

**Proof.**  $\tau_{X-C_c}(\mathbf{x}) = \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) \frac{|F_j \cap (X-C_c)|}{|F_j|} = \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) \left(1 - \frac{|F_j \cap C_c|}{|F_j|}\right) = \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) - \tau_{C_c}(\mathbf{x})$ .

The definition of complement operator satisfies the following properties:

1. **Boundary condition:** When the clusters and the output classes are crisp, i.e.,  $\tau_{C_c}(\mathbf{x}) = \tau_{C_c}(\mathbf{x})$ ,  $\tau$  behaves like an ordinary complement for rough sets. It means that if  $\tau_{C_c}(\mathbf{x}) = 0$  or 1, then  $\tau_{X-C_c}(\mathbf{x}) = 1$  or 0, respectively.
2. **Monotonicity:** If  $\tau_{C_c}(\mathbf{x}) \leq \tau_{C_c}(\mathbf{z}) \forall \mathbf{x}, \mathbf{z} \in X$ , then  $\tau_{X-C_c}(\mathbf{x}) \geq \tau_{X-C_c}(\mathbf{z})$ .
3. **Continuity:** Obviously,  $\tau_{X-C_c}(\mathbf{x})$  is a continuous function.
4. **Involutivity:**  $\tau_{X-(X-C_c)}(\mathbf{x}) = \tau_{C_c}(\mathbf{x})$ .

■

**Property D.9:**  $\tau_{A \cup B}(\mathbf{x}) \geq \max\{\tau_A(\mathbf{x}), \tau_B(\mathbf{x})\}$

**proof.**  $\tau_{A \cup B}(\mathbf{x}) = \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) \frac{|F_j \cap (A \cup B)|}{|F_j|} \geq \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) \frac{|F_j \cap A|}{|F_j|} = \tau_A(\mathbf{x})$ . Similarly,  $\tau_{A \cup B}(\mathbf{x}) \geq \tau_B(\mathbf{x})$ . Therefore,  $\tau_{A \cup B}(\mathbf{x}) \geq \max\{\tau_A(\mathbf{x}), \tau_B(\mathbf{x})\}$ . ■

**Property D.10:**  $\tau_{A \cap B}(\mathbf{x}) \leq \min\{\tau_A(\mathbf{x}), \tau_B(\mathbf{x})\}$

**Proof.**  $\tau_{A \cap B}(\mathbf{x}) = \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) \frac{|F_j \cap (A \cap B)|}{|F_j|} \leq \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) \frac{|F_j \cap A|}{|F_j|} = \tau_A(\mathbf{x})$ . Similarly,  $\tau_{A \cap B}(\mathbf{x}) \leq \tau_B(\mathbf{x})$ . Therefore,  $\tau_{A \cap B}(\mathbf{x}) \leq \min\{\tau_A(\mathbf{x}), \tau_B(\mathbf{x})\}$ . ■

**Property D.11:** If  $Z$  is a family of *pairwise disjoint* subsets of  $X$ , then  $\tau_{\cup Z}(\mathbf{x}) = \sum_{C_c \in Z} \tau_{C_c}(\mathbf{x})$ .

**Proof.**  $\tau_{\cup Z}(\mathbf{x}) = \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) \frac{|F_j \cap (\cup Z)|}{|F_j|} = \frac{1}{H} \sum_j \mu_{F_j}(\mathbf{x}) \frac{|\cup (F_j \cap Z)|}{|F_j|} = \sum_{C_c \in Z} \tau_{C_c}(\mathbf{x})$  ■

**Property D.12:**  $0 \leq \sum_{c=1}^C \tau_{C_c}(\mathbf{x}) \leq C$ .

**Proof.** If the input pattern does not belong to any cluster, then from Equation D.3  $\tau_{C_c}(\mathbf{x}) = 0 \forall c$ . Thus,  $\sum_{c=1}^C \tau_{C_c}(\mathbf{x}) = 0$ . In pattern classification it can happen when the

input pattern is not from any of the existing classes. On the other hand, when the input pattern belongs to all the classes with fuzzy membership value 1,

$$\begin{aligned}
\sum_{c=1}^C \tau_{C_c}(\mathbf{x}) &= \frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \frac{\sum_{\mathbf{z} \in X} \sum_{c=1}^C \min\{\mu_{F_j}(\mathbf{z}), \mu_{C_c}(\mathbf{z})\}}{\sum_{\mathbf{z} \in X} \mu_{F_j}(\mathbf{z})} \\
&= \frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \frac{\sum_{\mathbf{z} \in X} \sum_{c=1}^C \min\{\mu_{F_j}(\mathbf{z}), 1\}}{\sum_{\mathbf{z} \in X} \mu_{F_j}(\mathbf{z})} \\
&= \frac{1}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \frac{C \sum_{\mathbf{z} \in X} \mu_{F_j}(\mathbf{z})}{\sum_{\mathbf{z} \in X} \mu_{F_j}(\mathbf{z})} \\
&= \frac{C}{\hat{H}} \sum_{j=1}^{\hat{H}} \mu_{F_j}(\mathbf{x}) \tag{D.5}
\end{aligned}$$

Therefore, if the input pattern belongs to all the clusters completely, then  $\sum_{c=1}^C \tau_{C_c}(\mathbf{x})$  attains the maximum value  $C$ . ■

**Property D.13:** *When the clusters and the output classes are crisp,*

$$\underline{R}(C_c) = \{\mathbf{x} \in X \mid \tau_{C_c}(\mathbf{x}) = 1\} \tag{D.6-a}$$

$$\overline{R}(C_c) = \{\mathbf{x} \in X \mid \tau_{C_c}(\mathbf{x}) > 0\} \tag{D.6-b}$$

$$BN(C_c) = \overline{R}(C_c) - \underline{R}(C_c) = \{\mathbf{x} \in X \mid 0 < \tau_{C_c}(\mathbf{x}) < 1\} \tag{D.6-c}$$

**Proof.** For the crisp output classes with crisp clusters, the above results come directly from Equation (D.2-a) and (D.2-b). ■

Following is an interesting definition based on the above properties:

A  $C$ -class classification problem for a set of input patterns  $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  can be looked at as an assignment of the fuzzy-rough membership value  $\tau_{C_c}(\mathbf{x}_i)$  on each  $\mathbf{x}_i \in X$ ,  $\forall c = 1, 2, \dots, C$ ,  $\forall i = 1, 2, \dots, n$ . In fuzzy-rough context,  $C$  partitions of  $X$  are the set of values  $\{\tau_{C_c}(\mathbf{x}_i)\}$  that can be conveniently arranged on a  $C \times n$  matrix  $[\tau_{C_c}(\mathbf{x}_i)]$ . Based on the characteristic of  $[\tau_{C_c}(\mathbf{x}_i)]$  classification can be of the following three types [PB95]:

(a) **Crisp classification:**

$$\begin{aligned}
A_{hc} &= \left\{ [\tau_{C_c}(\mathbf{x}_i)] \in \mathbb{R}^{Cn} \mid \tau_{C_c}(\mathbf{x}_i) \in \{0, 1\} \forall c, \forall i; \right. \\
&\quad \left. \sum_{c=1}^C \tau_{C_c}(\mathbf{x}_i) = 1; \quad 0 < \sum_{i=1}^n \tau_{C_c}(\mathbf{x}_i) < n \forall c \right\} \tag{D.7-a}
\end{aligned}$$

(b) **Constrained fuzzy-rough classification:**

$$A_{fc} = \left\{ [\tau_{C_c}(\mathbf{x}_i)] \in \mathfrak{R}^{Cn} \mid \tau_{C_c}(\mathbf{x}_i) \in [0, 1] \forall c, \forall i; \right. \\ \left. \sum_{c=1}^C \tau_{C_c}(\mathbf{x}_i) = 1; \quad 0 < \sum_{i=1}^n \tau_{C_c}(\mathbf{x}_i) < n \forall c \right\} \quad (\text{D.7-b})$$

(c) **Possibilistic fuzzy-rough classification:**

$$A_{pc} = \left\{ [\tau_{C_c}(\mathbf{x}_i)] \in \mathfrak{R}^{Cn} \mid \tau_{C_c}(\mathbf{x}_i) \in [0, 1] \forall c, \forall i; \quad 0 < \sum_{i=1}^n \tau_{C_c}(\mathbf{x}_i) < n \forall c \right\} \quad (\text{D.7-c})$$

From the above relations, it is obvious that  $A_{hc} \mathbf{C} A_{fc} \mathbf{C} A_{pc}$ .

# BIBLIOGRAPHY

- [Adl94] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, (266):1021–1024., 1994.
- [AH95] H. Adeli and S. L. Hung. *Machine Learning: Neural Networks, Genetic Algorithms and Fuzzy Systems*. John Wiley and Sons, Inc, 1995.
- [Aka74] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.
- [AMMR93] R. Anand, K. Mehrotra, C. K. Mohan, and S. Ranka. An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4:962–963, 1993.
- [Arb95] M. A. Arbib. *The Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge, MA, 1995.
- [ASP94] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–64, January 1994.
- [BCR97] J. M. Benitez, J. L. Castro, and I Requena. Are artificial neural networks black boxes. *IEEE Transactions on Neural Networks*, 8(5):1156–1164, September 1997.
- [BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and Vapnik-Chervonenkis dimension. *Journal of the Association of Computing Machinery*, 36(4):929–965, October 1989.
- [Bel89] R. K. Belew. When both individuals and populations search: Adding simple learning to genetic algorithm. In *Proceedings of Third International Conference on GA* (George Mason University), pages 34–41, June 1989.
- [Bez81] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.
- [Bez94] J. C. Bezdek. The thirsty traveller visits gamont: A rejoinder to "comments on fuzzy set – what are they and why?". *IEEE Transactions on Fuzzy Systems*, 2(1):43–45, February 1994.
- [Bez96] J. C. Bezdek. A review of probabilistic, fuzzy, and neural models for pattern recognition. In C. H. Chen, editor, *Fuzzy Logic and Neural Network Handbook*. McGraw-Hill, Inc, New York, 1996.
- [BH94] J. C. Bezdek and R. J. Hathaway. Optimization of fuzzy clustering criteria using genetic algorithms. In *Proceedings of First IEEE Conference on Evolutionary Computation*, pages 589–594, June 1994.
- [BHS97] T. Back, U. Hammel, and H. P. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, April 1997.
- [Bis95] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford Univ Press, 1995.
- [BL96] N. K. Bose and P. Liang. *Neural Network Fundamentals with Graphs, Algorithms and Applications*. McGraw-Hill Inc., New York, 1996.

- [BL98] A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. In R. Greiner and D. Subramanian, editors, to appear in the special issue of Artificial Intelligence on 'Relevance'. 1998.
- [BMP97] M. Banerjee, S. Mitra, and S. K. Pal. Knowledge-based fuzzy MLP with rough sets. In IEEE International Conference on Neural Networks (Houston, USA), June 9-12 1997.
- [Bow84] S. T. Bow. Pattern Recognition. Marcel Dekker, New York, 1984.
- [BP92] J. C. Bezdek and S. K. Pal. Fuzzy Models for *Pattern* Recognition, Eds. IEEE Press, New York, 1992.
- [BR94] A. K. Bhattacharya and B. Roysam. Joint solution of low-, intermediate-, and high- level vision tasks by evolutionary optimization: Application to computer vision at low snr. IEEE Transactions on Neural Networks, 5(1):83-93, January 1994.
- [BZ95] S. A. Billings and G. L. Zheng. Radial basis function network configuration using genetic algorithm. Neural Networks, 8(6):877-890, 1995.
- [CAMC92] J. J. Choi, R. J. Arabshahi, R. J. Marks, and T. P. Candell. Fuzzy parameter adaptation in neural systems. In Proceedings of IEEE International Conference on Neural Networks, pages 232-238, 1992.
- [CH67] T. M. Cover and P. E. Hart. Nearest neighbor pattern classifier. IEEE Transactions on Information Theory, pages 21-27, 1967 1967.
- [CHL96] O. Cordon, F. Herrera, and M. Lozano. On the bidirectional integration of genetic algorithms and fuzzy logic. In 2nd On line Workshop on Evolutionary Computation (*WEC2*), Nagoya (Japan), pages 13-17, 1996.
- [Cho95] S. B. Cho. Fuzzy aggregation of modular neural networks with ordered weighted averaging operators. Approximate Reasoning, 13:359-375, 1995.
- [Cho97] S. B. Cho. Neural-Network classifier for recognizing totally unconstrained handwritten numerals. IEEE *Transactions* on Neural Networks, 8(1):43-52, January 1997.
- [CK92] S. B. Cho and J. H. Kim. A **two-stage** classification scheme with **backpropagation** neural network classifiers. Pattern Recognition Letters, 13(5):309-331, May 1992.
- [CK95a] S. B. Cho and J. H. Kim. Combining multiple neural networks by fuzzy integral for robust classification. IEEE Transactions on System, Man and Cybernetics, 25(2):380-384, February 1995.
- [CK95b] S. B. Cho and J. H. Kim. Multiple network fusion using fuzzy logic. IEEE Transactions on Neural Networks, 6(2):497-501, March 1995.
- [COB92] J. J. Choi, H. O'Keefe, and P. K. Baruah. Non-linear system diagnosis using neural networks and fuzzy logic. In Proceedings of IEEE International Conference on Fuzzy Systems (*San* Diago), pages 813-820, 1992.
- [Dav91] L. Davis. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, 1991.
- [DH79] R. Duda and P. Hart. Pattern Classification and Scene Analysis. Wiley, New York, 1979.
- [DHR93] D. Driankov, H. Hellendorn, and M. Reinfrank. An Introduction to Fuzzy Control. Springer-Verlag, Berlin, 1993.

- [DJ87] R. Dubes and A. Jain. *Algorithms that Cluster Data*. Prentice Hall, Englewood Cliffs, NJ, 1987.
- [DK82] R. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice Hall, Englewood Cliffs, NJ, 1982.
- [DL97] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3), August 1997.
- [DMC96] M. Dorigo, V. Maniezzo, and A. Coloni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on System, Man and Cybernetics*, 26(1):29–41, 1996.
- [DP80] D. Dubois and H. Prade. *Fuzzy Sets and Systems*. Academic Press, New York, 1980.
- [DP88] D. Dubois and H. Prade. *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. Plenum Press, New York: Plenum, 1988.
- [DP90] D. Dubois and H. Prade. Rough-fuzzy sets and fuzzy-rough sets. *International Journal of General Systems*, 17(2-3):191–209, 1990.
- [DP92] D. Dubois and H. Prade. Putting rough sets and fuzzy sets together. In R. Slowinski, editor, *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Set Theory*. Kluwer Academic Publishers, Dordrecht, 1992.
- [Dud70] R. Duda. Elements of pattern recognition. In J. M. Mendel and K. S. Fu, editors, *Adaptive, Learning and Pattern Recognition Systems*. Academic Press, New York, 1970.
- [DY97] P. J. Darwen and X. Yao. Speciation as automatic categorical modularisation. *IEEE Transactions on Evolutionary Computation*, 1(2):101–108, July 1997.
- [ESY92] P. Eswar, C. C. Sekhar, and B. Yegnanarayana. Use of fuzzy mathematical concepts in character spotting for automatic recognition of continuous speech in Hindi. *Fuzzy Sets and Systems*, 46(1):1–9, February 1992.
- [FK96] H. Frigui and R. Krishnapuram. A comparison of fuzzy shell clustering methods for the detection of ellipses. *IEEE Transactions on Fuzzy Systems*, 4(2):193–199, May 1996.
- [Fog91a] D. B. Fogel. An information criterion introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 2(5):490–497, September 1991.
- [Fog91b] D. B. Fogel. *System Identification through Simulated Evolution: A Machine Learning Approach of Modeling*. Ginn Press, Needham, MA, 1991.
- [Fog94a] D. B. Fogel. Asymptotic convergence properties of genetic algorithms and evolutionary programming. *Cybernetics and Systems*, 25:389–407, 1994.
- [Fog94b] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, January 1994.
- [Fog95] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Learning*. IEEE Press, Piscataway, 1995.
- [Fog98] D. B. Fogel. *Evolutionary Computation: The Fossil Record*. IEEE Press, Piscataway, 1998.
- [FOW66] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.



- [FS89] Y. Fukuyama and M. Sugeno. A new method for choosing the number of clusters for the c-means method. In Proceedings of Fifth Fuzzy Systems Symposium, pages 247–250, (in Japanese) 1989.
- [FS93] D. B. Fogel and P. K. Simpson. Evolving fuzzy clusters. In Proceedings of International Conference on Neural Networks (*San Francisco*), pages 1829–1834, 1993.
- [FSW97] A. Famili, W. M. Shen, and R. Weber. Data preprocessing and intelligent data analysis. *Intelligent Data Analysis*, 1(1), January 1997.
- [Fu68] K. S. Fu. Sequential Methods in Pattern Recognition and Machine Learning. Academic Press, London, 1968.
- [Fu82] K. S. Fu. Syntactic Pattern Recognition and Applications. Prentice-Hall, Englewoods Cliffs, 1982.
- [Fuk89] K. Fukunaga. Introduction to Statistical Pattern Recognition. Academic Press, New York, 1989.
- [GG89] I. Gath and A. B. Geva. Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):773–781, July 1989.
- [GN94] M. Grabisch and J. M. Nicolas. Classification by fuzzy integral: Performance and test. *Fuzzy Sets and Systems*, (65):255–271, 1994.
- [Gol89] D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley, Reading MA, 1989.
- [Gra96] M. Grabisch. The representation and interaction of features by fuzzy measures. *Pattern Recognition Letters*, (17):567–575, 1996.
- [Gra97] M. Grabisch. Fuzzy measures and integrals: A survey of applications and recent issues. In D. Dubois, H. Prade, and R. Yager, editors, *Fuzzy Sets Methods in Information Engineering: A Guided Tour of Applications*. J. Wiley, New York, 1997.
- [Har75] J. Hartigan. Clustering Algorithms. Wiley, New York, 1975.
- [Has95] M. F. Hassoun. Fundamentals of Artificial Neural Networks. MIT Press, Cambridge, MA, 1995.
- [Hau92] D. Haussler. Decision theoretic generalisations of the PAC model for neural net and other learning applications. *Information and Computation*, 100:78–150, 1992.
- [Hay94] S. Haykin. Neural Networks - A Comprehensive Foundation. Macmillan College Publishing Company, New York, 1994.
- [HC96] J. E. Hunt and D. E. Cooke. Learning using an artificial immune system. *Journal of Networks and Computer Applications*, 19:189–212, 1996.
- [HH93] D. R. Hush and B. G. Home. Progress in supervised neural networks. *IEEE Transactions on Signal Processing*, pages 8–39, January 1993.
- [HHMS96] K. J. Hunt, R. Haas, and R. Murray-Smith. Extending the functional equivalence of radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, 7(3):776–771, May 1996.
- [HHVLV94] F. Herrare, E. Herrera-Viedma, M. Lozano, and J. L. Verdegay. Fuzzy tools to improve genetic algorithms. In Proceedings of Second European Congress on Intelligent Techniques and Soft Computing (*Aachen, Germany*), pages 1532–1539, September 1994.

- [HKP91] J. Hertz, A. Krogh, and R. G. Palmer. Introduction to the Theory of Neural Computation. Addison-Wesley, Reading, MA, 1991.
- [HM97] F. Herrera and L. Magdalena. Genetic fuzzy system. In B. Riecan R. Mesiar, editor, Fuzzy Structures: Current Trends, volume 13, pages 93–121. Tatra Mountains Mathematical Publications, 1997.
- [Hol75] J. H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.
- [HP94] K. Hirota and W. Pedrycz. OR/AND neurons in modeling fuzzy set connectives. *IEEE Transactions on Neural Networks*, 2(2):151–161, May 1994.
- [HPA<sup>+</sup>97] R. Hashemi, B. Pearce, R. Arani, W. Hinson, and M. Paule. A fusion of rough sets, modified rough sets, and genetic algorithms for hybrid diagnostic systems. In T.Y. Lin and N. Cercone, editors, Rough Sets and Data Mining. Analysis for Imprecise Data, pages 149–175. Kluwer Academic Publishers, Boston, London, Dordrecht, 1997.
- [HSW89] K. Hornic, M. Stinchcombe, and H. White. Multilayered feedforward networks are universal approximators. *Neural Networks*, 4:359–364, 1989.
- [IFT93] H. Ishibuchi, R. Fujioka, and H. Tanaka. Neural networks that learn from fuzzy if-then' rules. *IEEE Transactions on Fuzzy Systems*, 1(2):85–97, May 1993.
- [INYT95] H. Ishibuchi, K. Nozaki, N. Yarnamoto, and H. Tanaka. Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(3):260–270, August 1995.
- [JJ93] R. A. Jacobs and M. I. Jordan. Learning piecewise control strategies in a modular neural network architecture. *IEEE Transactions on System, Man and Cybernetics*, 23:337–345, 1993.
- [JJNH91] R. A. Jacobs, M. I. Jordan, M. I. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.
- [JK95] J. John and R. Kohavi. Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. In Proceedings of the First International Conference on Knowledge and Data Mining, pages 643–649, 1995.
- [JR94] S. Jockusch and H. Ritter. Self-organizing maps: Local competition and evolutionary optimization. *Neural Networks*, 7(8):1229–1239, 1994.
- [JS93] J. S. R. Jang and C. T. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, 4(1):156–159, January 1993.
- [JSM97] J. S. R. Jang, C. T. Sun, and E. Mijutani. Neuro-Fuzzy and Soft Computing. Prentice-Hall, Englewood Cliffs, NJ, 1997.
- [Kan82] A. Kandle. Fuzzy Techniques in Pattern Recognition. Wiley, New York, 1982.
- [Kan86] A. Kandle. Fuzzy Mathematical Techniques with Applications. Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.
- [KF93] G. S. Klir and T. A. Folger. Fuzzy Sets, Uncertainty and *Information*. Prentice-Hall, Englewood Cliffs, NJ, 1993.
- [KG85] A. Kauffman and M. M. Gupta. Introduction to Fuzzy Mathematics. Van Nostrand Reinhold, New York, 1985.

- [KGG85] J. M. Keller, M. R. Gray, and J. A. Givens. A fuzzy K-nearest neighbor algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, 15(4):580–585, July/August 1985.
- [KGT<sup>+</sup>94] J. M. Keller, P. Gader, H. Tahani, J. H. Chiang, and M. Mohamed. Advances in fuzzy integration for pattern recognition. *Fuzzy Sets and Systems*, (65):273–283, 1994.
- [KH85] J. M. Keller and D. J. Hunt. Incorporating fuzzy membership functions into the perceptron algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 693–699, July/August 1985.
- [Khe88] D. Khemani. *Theme based Planning in an Uncertain Environment*. PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, 1988.
- [KJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, (220):671–680, 1983.
- [KK93] R. Krishnapuram and J. M. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2):98–110, May 1993.
- [KKR92] J. M. Keller, R. Krishnapuram, and F. C. H. Rhee. Evidence aggregation networks for fuzzy logic interface. *IEEE Transactions on Neural Networks*, 3(5):761–769, 1992.
- [KL79] A. Kandel and S. C. Lee. *Fuzzy Switching and Automata: Theory and Applications*. Crane, Russak & Company, New York, 1979.
- [KNF92] R. Krishnapuram, O. Nasraoui, and H. Frigui. The fuzzy c spherical shells algorithm. *IEEE Transactions on Neural Networks*, 3(5):663–671, Sep. 1992.
- [KO96] J. M. Keller and J. Osborn. Training the fuzzy integral. *Information Science*, (15):1–24, 1996.
- [Koh89] T. Kohonen. *Self Organization and Associative Memory*, 3rd edition. Springer Verlag, Berlin, Germany, 1989.
- [Koh90] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, September 1990.
- [Kos93] B. Kosko. *Fuzzy Thinking*. Harper Collins, Glasgow, UK, 1993.
- [Koz92] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT, Press, Cambridge, Massachusetts, 1992.
- [KQ88] J. M. Keller and H. Qui. Fuzzy set methods in pattern recognition. In Kitler, editor, *Proceedings of Fourth International Conference on Pattern Recognition*, Cambridge 28-30 March. Springer Verlag, 1988.
- [KR89] D. Khemani and R. S. Ramakrishna. Bridge: A benchmark for knowledge based planning. *The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology (CC-AI)*, 6(2/3):137–151, 1989.
- [KT92] J. M. Keller and H. Tahani. Implementation of conjunctive and disjunctive fuzzy logic rules with neural networks. *International Journal of Approximate Reasoning*, (6):221–240, 1992.
- [Kun93] S. Y. Kung. *Digital Neural Networks*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [KY95] G. S. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic – Theory and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

- [LD95] S. Loncaric and P. Dhawan. Near-optimal mst-based shape description using genetic algorithm. *Pattern Recognition*, 28(4):571–579, 1995.
- [LI98] B. L. Lu and M. Ito. Task decomposition and module combination based on class relations: A modular neural network for pattern classification. Technical Report BMC TR-98-1, Bio-Mimetic Control Research Center, The Institute of Physical and Chemical Research (RIKEN), 1998.
- [Liu95] Y. Liu. Unbiased estimate of generalization error and model selection in neural network. *Neural Networks*, 8(2):215–219, 1995.
- [LL95] C. T. Lin and Y. C. Lu. A neural fuzzy system with linguistic teaching signals. *IEEE Transactions on Fuzzy Systems*, 3(2):169–189, May 1995.
- [LL96] C. T. Lin and C. S. G. Lee. *Neural Fuzzy Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1996.
- [LS95] G. F. Luger and W. A. Stubblefield. *Artificial Intelligence*. Addison-Wesley, Reading MA, 1995.
- [Man93] M. Mandischer. Representation and evolution of neural networks. In *Proceedings of the International Conference in Innsbruck (Austria)*, pages 643–649, 1993.
- [MCHK94] M. T. Musavi, K. H. Chan, D. M. Hummels, and K. Kalantri. On the generalization ability of neural network classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6), June 1994.
- [Mic92] Z. Michalewicz. *Genetic Algorithm + Data Structure*. Springer Verlag, New York, 1992.
- [MS89a] M. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 1*. San Mateo, CA: Morgan Kauffman, New York, 1989.
- [MS89b] T. Murofushi and M. Sugeno. An interpretation of fuzzy measure and the Choquet integral as an integral with respect to a fuzzy measure. *Fuzzy Sets and Systems*, 29:201–227, June 1989.
- [MTH89] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks and genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*. Morgan Kaufmann, San Mateo, CA, 1989.
- [MYA94] N. Murata, S. Yoshizawa, and S. Amari. Network information criterion - determining the number of hidden units for an artificial neural network model. *IEEE Transactions on Neural Networks*, 5(6):865–872, November 1994.
- [Nat91] B. K. Natarajan. *Machine Learning: A Theoretical Approach*. Morgan Kaufmann, San Mateo, California, 1991.
- [Nee96] A. Neeharika. *Generalization Capability of Feedforward Neural Networks for Pattern Recognition Tasks*. MS Thesis, Indian Institute of Technology, Madras, Department of Computer Science and Engineering, August, 1996.
- [NIT96] K. Nozaki, H. Ishibuchi, and H. Tanaka. Adaptive fuzzy rule-based classification system. *IEEE Transactions on Fuzzy Systems*, 4(3):238–250, August 1996.
- [OC97] M. Lozano O. Cordon, F. Herrera. On the combination of fuzzy logic and evolutionary computation: A short review and bibliography: 1989-1995. In

- W. Pedrycz, editor, *Fuzzy Evolutionary Computation*, pages 57–77. Kluwer Academic Press, 1997.
- [Pal92] S. K. Pal. Fuzzy set theoretic measures for automatic feature evaluation: II. *Information Sciences*, 65:165–179, July-October 1992.
- [Pao89] Y. H. Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading MA, 1989.
- [Paw82] Z. Pawlak. Rough sets. *International Journal of Computer and Information Science*, 11:341–356, 1982.
- [Paw91] Z. Pawlak. *Rough Sets: Theoretical Aspects of Reasoning About Data*. Kluwer, Dordrecht, 1991.
- [Paw94] Z. Pawlak. Vagueness and uncertainty: A rough set perspective. Technical Report ICS Research Report 19/94, Institute of Computer Science, Warsaw University of Technology, Warsaw, Poland, March 1994.
- [Paw95] Z. Pawlak. Rough sets present state and future prospects. Technical Report ICS Research Report 32/94, Institute of Computer Science, Warsaw University of Technology, Warsaw, Poland, 1995.
- [PB94] N. R. Pal and J. C. Bezdek. Measuring fuzzy entropy. *IEEE Transactions on Fuzzy Systems*, 2(2):107–118, May 1994.
- [PB95] N. R. Pal and J. C. Bezdek. On cluster validity for the fuzzy C-means model. *IEEE Transactions on Fuzzy Systems*, 3(3):330–379, August 1995.
- [PBSZ95] Z. Pawlak, J. G. Busse, R. Slowinsky, and W. Ziarko. Rough sets. *Communications of the ACM*, 38(11):89–95, November 1995.
- [PC86] S. K. Pal and B. Chakraborty. Fuzzy set theoretic measure for automatic feature evaluation. *IEEE Transactions on System, Man and Cybernetics*, 16(5):754–760, September/October 1986.
- [Ped90] W. Pedrycz. Fuzzy sets in pattern recognition: Methodology and methods. *Pattern Recognition*, 23(1/2):121–146, 1990.
- [Ped92] W. Pedrycz. Fuzzy neural networks with reference neurons as pattern classifiers. *IEEE Transactions on Neural Networks*, 3(5):770–775, September 1992.
- [PFF95] W. Porto, D. B. Fogel, and L. J. Fogel. Alternative neural network training methods. *IEEE Expert*, pages 16–22, June 1995.
- [PIL96] Y. H. Pao, B. Igelnik, and S. R. LeClair. An approach for neural-net computing with two-objective functions. In *Proceedings of IEEE International Conference on Neural Networks (Washington D.C.)*, pages 181–186, June 1996.
- [PK96] G. Purushothaman and N. B. Karayiannis. Quantum neural networks (QNNs): Inherently fuzzy feedforward neural networks. In *Proceedings of IEEE Conference on Neural Networks (Washington D.C.)*, pages 1085–1090, June 1996.
- [PM86] S. K. Pal and D. Dutta Majumder. *Fuzzy Mathematical Approach to Pattern Recognition*. Wiley (Halsted Press), New York, 1986.
- [PM92] S. K. Pal and S. Mitra. Multilayer perceptron, fuzzy sets and classification. *IEEE Transactions on Neural Networks*, 3(5):683–697, September 1992.
- [PP92] N. R. Pal and S. K. Pal. Higher order fuzzy entropy and hybrid entropy of a set. *Information Sciences*, 61(3):211–231, 1992.

- [PP96] S. K. Pal and N. R. Pal. Soft computing: Goals, tools and feasibility. *Journal of IETE*, 42(5):195–204, July–October 1996.
- [PPB97] N. R. Pal, K. Pal, and J. C. Bezdek. A mixed c-means clustering model. In *Proceedings of IEEE International Conference on Fuzzy Systems (Barcelona, Spain)*, pages 11–21, July 1997.
- [PR93] W. Pedrycz and A. F. Rocha. Fuzzy-set based models of neurons and knowledge-based networks. *IEEE Transactions on Fuzzy Systems*, 1(4):254–266, November 1993.
- [PWZ88] Z. Pawlak, S. K. M. Wong, and W. Ziarko. Rough sets: Probabilistic verses deterministic approach. *International Journal of Man-Machine Studies*, 29:81–95, 1988.
- [Ree93] R. Reed. Pruning algorithms - A survey. *IEEE Transactions on Neural Networks*, 4:740–747, September 1993.
- [RF96] J. M. Redners and S. P. Flasse. Hybrid methods using genetic algorithms for global optimization. *IEEE Transactions on System, Man and Cybernetics*, 26(2):243–258, 1996.
- [RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and McClelland, editors, *Parallel and Distributed Processing*. MIT Press, Cambridge, MA, 1986.
- [Rip96] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, 1996.
- [RPY97] P. P. Raghu, R. Poongodi, and B. Yegnanarayana. Unsupervised texture classification using vector quantization and deterministic relaxation neural network. *IEEE Transactions on Image Processing*, 6(10):1376–1388, October 1997.
- [RRM<sup>+</sup>96] C. Rodriguez, S. Rementeria, J. I. Martin, A. Lafuente, J. Mugerza, and J. Perez. A modular neural network approach to fault diagnosis. *IEEE Transactions on Neural Networks*, 7(2):326–340, March 1996.
- [RY95] P. P. Raghu and B. Yegnanarayana. A combined neural network approach for texture classification. *Neural Networks*, 8(6):975–987, December 1995.
- [RY96] P. P. Raghu and B. Yegnanarayana. Segmentation of Gabor filtered textures using deterministic relaxation. *IEEE Transactions on Image Processing*, 5(12), December 1996.
- [RY97] P. P. Raghu and B. Yegnanarayana. Multispectral image classification using Gabor filters and stochastic relaxation neural network. *Neural Networks*, 10(3):561–572, December 1997.
- [RY98] P. P. Raghu and B. Yegnanarayana. Supervised texture classification using a probabilistic neural network and constrain satisfaction model. *IEEE Transactions on Neural Networks*, 9(3):516–522, May 1998.
- [SB97] M. Scherf and W. Brauer. Feature selection by means of feature weighting approach. Technical Report FKI-221-97, *Forschungsberichte Kunstliche Intelligenz*, Institut fur Informatik, Technische Universitat Munchen, 1997.
- [Sch81] H. P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley, Chichester, 1981.
- [SF95] N. Saravanan and D. B. Fogel. Evolving neural control systems. *IEEE Expert*, pages 23–27, June 1995.

- [Sha76] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.
- [Slo92] R. Slowinsky. *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Set Theory*. Kluwer Academic Publishers, Dordrecht, 1992.
- [SM93] A. Sankar and R. J. Mammone. Growing and pruning neural tree networks. *IEEE Transactions on Neural Networks*, 42(3):291–299, March 1993.
- [SP94] M. Srinivas and L. M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithm. *IEEE Transactions on System, Man and Cybernetics*, 24(4):656–667, 1994.
- [Spe90] D. F. Specht. Probabilistic neural networks. *Neural Networks*, 3:109–118, 1990.
- [SS91] S. Z. Selim and K. A. Sultan. A simulated annealing algorithm for the clustering problem. *Pattern Recognition*, 24(10):1003–1008, 1991.
- [SS93] R. Slowinsky and J. Stefanowski. *Foundations of Computing and Decision Sciences (eds.)*, 18(3-4), Fall 1993.
- [SST93] M. R. A. Sadzadi, S. Sheedvash, and F. O. Trujillo. Recursive dynamic node creation in multilayer neural networks. *IEEE Transactions on Neural Networks*, 4(2):242–256, 1993.
- [Sug74] M. Sugeno. *Theory of fuzzy integrals and its applications*. PhD thesis, Tokyo Institute of Technology, 1974.
- [Sus92] H. J. Sussmann. Uniqueness of the weights of minimal feedforward nets with given input-output map. *Neural Networks*, 5:589–593, 1992.
- [SY] M. Sarkar and B. Yegnanarayana. Application of fuzzy integral in modular networks for contract bridge bidding. Accepted in *Fuzzy Sets and Systems*.
- [SY96] C. C. Sekhar and B. Yegnanarayana. Recognition of stop-consonant-vowel (svc) segments in continuous speech using neural network models. *Journal of Institution of Electronics and Telecommunication Engineers (IETE)*, 42(4 & 5):269–280, July-October 1996.
- [SY98a] M. Sarkar and B. Yegnanarayana. Fuzzy-rough membership functions. Accepted in *IEEE International Conference on Systems, Man and Cybernetics, San Diego, California, USA*, October 11-14 1998.
- [SY98b] M. Sarkar and B. Yegnanarayana. Fuzzy-rough neural networks for vowel classification. Accepted in *IEEE International Conference on Systems, Man and Cybernetics, San Diego, California, USA*, October 11-14 1998.
- [SY98c] M. Sarkar and B. Yegnanarayana. A review on merging some recent techniques with artificial neural networks. Accepted in *IEEE International Conference on Systems, Man and Cybernetics, San Diego, California, USA*, October 11-14 1998.
- [SY98d] M. Sarkar and B. Yegnanarayana. Rough-fuzzy membership functions. In *Proceedings of IEEE International Conference on Fuzzy Systems (Anchorage, Alaska, USA)*, pages 796–801, May 4-9 1998.
- [SY98e] C. C. Sekhar and B. Yegnanarayana. Modular networks and constraint satisfaction model for recognition of stop consonant-vowel (SCV) utterances. In *Proceedings of IEEE International Conference on Neural Networks (Anchorage, Alaska, USA)*, pages 1206–1211, May 4-9 1998.

- [TG74] J. Tau and R. Gonzalez. *Pattern Recognition Principles*. Addison Wesley, Reading, MA, 1974.
- [TI92] H. Tanaka and H. Ishibuchi. Fuzzy expert system based on rough sets and its application to medical diagnosis. *International Journal of General Systems*, 21:83–97, 1992.
- [TIS92] H. Tanaka, H. Ishibuchi, and T. Shigenaga. Fuzzy inference system based on rough sets and its application to medical diagnosis. In R. Slowinski, editor, *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Set Theory*. Kluwer Academic Publishers, Dordrecht, 1992.
- [TK90] H. Tahani and J. K. Keller. Information fusion in computer vision using fuzzy integral. *IEEE Transactions on System, Man and Cybernetics*, 20(3):733–741, May/June 1990.
- [TM87] T. Toffoli and N. Margolus. *Cellular Automata Machines: A New Environment for Modeling*. MIT Press, Cambridge, MA, 1987.
- [TMBC92] S. Thiria, C. Mejia, F. Badran, and M. Crepon. Multimodular architecture for remote sensing operations. In J. E. Moddy, J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems-4*. Morgan Kaufmann, 1992.
- [Vid97] M. Vidyasagar. *A Theory of Learning and Generalization: With Applications to Neural Networks and Control Systems*. Springer Verlag, New York, 1997.
- [Vig70] S. S. Viglione. Application of pattern recognition technology. In J. M. Mendel and K. S. Fu, editors, *Adaptive, Learning and Pattern Recognition Systems*. Academic Press, New York, 1970.
- [WAM97] D. Wettschereck, D. W. Aha, and T. Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithm. *Artificial Intelligence Review*, 1997.
- [WC96] B. A. Whitehead and T. D. Choate. Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction. *IEEE Transactions on Neural Networks*, 7(4):869–880, July 1996.
- [Whi96] B. A. Whitehead. Genetic evolution of radial basis function coverage using orthogonal niches. *IEEE Transactions on Neural Networks*, 7(6):1525–1528, November 1996.
- [WK92] Z. Wang and G. Klir. *Fuzzy Measure Theory*. Plenum Press, New York, 1992.
- [WM97] L. X. Wang and J. M. Mendel. Fuzzy basis functions, universal approximations, and orthogonal least square learning. *IEEE Transactions on Neural Networks*, 3(5):807–814, September 1997.
- [Wro95] J. Wroblewski. Finding minimal reducts using genetic algorithm (extended version). In *Second Annual Joint Conference on Information Sciences (North Carolina)*, pages 186–189, September 1995.
- [WW97] J. Wang and Z. Wang. Using neural networks to determine sugeno measures by statistics. *Neural Networks*, 10(1):183–195, 1997.
- [WZ87] S. K. M. Wong and W. Ziarko. Comparison of the probabilistic approximate classification and fuzzy set model. *Fuzzy Sets and Systems*, 21:357–362, 1987.
- [XB91] X. L. Xie and G. Beni. A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):841–847, August 1991.



- [Yag93] R. R. Yager. Element selection from a fuzzy subset using the fuzzy integral. *IEEE Transactions on System, Man and Cybernetics*, pages 467–477, 1993.
- [Yao93] X. Yao. Evolutionary artificial neural networks. *International Journal of Neural Networks*, 4(3):203–222, 1993.
- [Yeg98] B. Yegnanarayana. *Artificial Neural Networks*. Prentice Hall, New Delhi, India, 1998.
- [YF93] M. Yoneda and S. Fukami. Interactive determination of a utility function represented as a fuzzy integral. *Information Sciences*, 71:43–64, 1993.
- [YKSS95] B. Yuan, G. J. Klir, and J. F. Swan-Stone. Evolutionary fuzzy C-means clustering algorithm. In *Proceedings of First IEEE Conference on Fuzzy Systems (Yokohama)*, pages 2221–2226, March 1995.
- [YL97] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3), May 1997.
- [YL98] X. Yao and Y. Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on System, Man and Cybernetics*, 28(B2), April 1998.
- [Zad65] L. A. Zadeh. Fuzzy sets. *Information and Control*, pages 338–353, 1965.
- [Zad78] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

# PUBLICATIONS

## PAPERS IN JOURNALS

1. M. Sarkar and B. Yegnanarayana, "Feedforward neural network classifiers: Back-propagation learning algorithm with fuzzy objective functions", accepted in *IEEE Transactions on Systems, Man and Cybernetics*.
2. M. Sarkar and B. Yegnanarayana, "Rough-Fuzzy membership functions in classification", accepted in *fuzzy Sets and Systems*.
3. M. Sarkar, B. Yegnanarayana and D. Khemani, "Backpropagation learning algorithms for classification with fuzzy mean square error", *Pattern Recognition Letters*, vol. 19/1, pp 43-51, 1998.
4. M. Sarkar, B. Yegnanarayana and D. Khemani, "A clustering algorithm using an evolutionary programming-based approach", *Pattern Recognition Letters*, vol. 18/10, pp. 975-986, 1997.
5. B. Yegnanarayana, D. Khemani and M. Sarkar, "Neural networks for contract bridge bidding", *Sadhana*, vol. 21, no. 3, pp. 395-413, June 1996.

## PAPERS COMMUNICATED TO JOURNALS

1. M. Sarkar and B. Yegnanarayana, "Fuzzy-Rough membership functions in classification", communicated to *IEEE Transactions on fuzzy Systems*.
2. M. Sarkar and B. Yegnanarayana, "Feedforward neural networks configuration using evolutionary programming", communicated to *Pattern Recognition*.
3. M. Sarkar and B. Yegnanarayana, "Evolutionary programming-based hybrid clustering technique", communicated to *IEEE Transactions on Systems, Man and Cybernetics*.
4. M. Sarkar and B. Yegnanarayana, "Rough-Fuzzy set-based approach for selecting input features in classification", communicated to *IEEE Transactions on Neural Networks*.

## PAPER IN EDITED VOLUME

M. Sarkar and B. Yegnanarayana, "Application of fuzzy-rough sets in fuzzy integral-based modular neural networks", *Rough-Fuzzy Hybridization: A New Trend in Decision-Making*, ed. S. K. Pal and A. Skowron, Springer Verlag (in press).

## PAPERS IN INTERNATIONAL CONFERENCES

1. M. Sarkar and B. Yegnanarayana, "Fuzzy-Rough membership functions", accepted in IEEE International Conference on Systems, Man and Cybernetics, San Diego, California, USA, October 11-14, 1998.
2. M. Sarkar and B. Yegnanarayana, "A review on merging some recent techniques with artificial neural networks", accepted in IEEE International Conference on Systems, Man and Cybernetics, San Diego, California, USA, October 11-14, 1998.
3. M. Sarkar and B. Yegnanarayana, "Fuzzy-Rough neural networks for vowel classification", accepted in IEEE International Conference on Systems, Man and Cybernetics, San Diego, California, USA, October 11-14, 1998.
4. M. Sarkar and B. Yegnanarayana, "Rough-Fuzzy membership functions", in Proceedings of IEEE International Conference on Fuzzy Systems, Anchorage, USA, pp. 796-801, May 3-9, 1998.
5. M. Sarkar and B. Yegnanarayana, "Application of fuzzy-rough sets in modular neural networks", Proceedings of IEEE International Conference on Neural Networks, Anchorage, USA, pp. 741-746, May 3-9, 1998.
6. M. Sarkar and B. Yegnanarayana, "Incorporation of fuzzy classification properties into backpropagation learning algorithm", in Proceedings of IEEE International Conference on Fuzzy Systems, Barcelona, Spain, vol. 3, pp. 1701-1706, July 1-5, 1997.
7. M. Sarkar and B. Yegnanarayana, "Rough-Fuzzy set theoretic approach to evaluate the importance of input features in classification", in Proceedings of IEEE International Conference on Neural Networks, Houston, USA, vol. 3, pp. 1590-1595, June 9-12, 1997.
8. M. Sarkar and B. Yegnanarayana, "Feedforward neural networks configuration using evolutionary programming", in Proceedings of IEEE International Conference on Neural Networks, Houston, USA, vol. 1, pp. 438-443, June 9-12, 1997.
9. M. Sarkar and B. Yegnanarayana, "An evolutionary programming-based probabilistic neural network construction technique", in Proceedings of IEEE International Conference on Neural Networks, Houston, USA, vol. 1, pp. 456-461, June 9-12, 1997.
10. M. Sarkar, B. Yegnanarayana and D. Khemani, "Feedforward neural networks and fuzzy classification", in Proceedings of Fourth International Conference on Advanced Computing, Bangalore, India, pp. 65-72, December 16-18, 1996.

11. M. Sarkar and B. Yegnanarayana, "A clustering algorithm using evolutionary programming", in Proceedings of IEEE International Conference on Neural Networks, Washington, USA, vol. 2, pp. 1162-1167, June 3-6, 1996.
12. M. Sarkar, "Evolutionary programming-based fuzzy clustering", in Proceedings of Fifth Annual Conference on Evolutionary Programming, MIT Press, Cambridge, Massachusetts, San Diego, USA, pp. 247-256, 1996.
13. B. Yegnanarayana, D. Khemani and M. Sarkar, "Hierarchical neural networks - An application in contract bridge", in Proceedings of International Conference on Automation, Indore, India, pp. 9-12, December 12-14, 1995.

## PAPERS IN NATIONAL CONFERENCES

1. M. Sarkar, "Evolutionary programming-based fuzzy clustering and its applications", in Proceedings of the 84<sup>th</sup> Indian Science Congress-1997, Delhi University, New Delhi, India, January 3-8, 1997.
2. M. Sarkar, B. Yegnanarayana and D. Khemani, "Application of neural networks in contract bridge bidding", in Proceedings of National Conference on Neural Networks and Fuzzy Systems, Anna University, Madras, India, pp. 144151, March 16-18, 1995.