UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF
NATURAL & APPLIED SCIENCES

NOVEL CLUSTERING METHODS FOR NEUROFUZZY SYSTEMS DESIGN

**PhD THESIS
IN
ELECTRICAL AND ELECTRONICS
ENGINEERING**

**BY
YUNİS TORUN
SEPTEMBER 2010**

Novel Clustering Methods for NeuroFuzzy Systems Design

PhD Thesis

in

Electrical and Electronics Engineering

University of Gaziantep

Supervisor

Prof. Dr. Gülay TOHUMOĞLU

by

Yunis TORUN

September 2010

T. C.
UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF
NATURAL & APPLIED SCIENCES
ELECTRICAL AND ELECTRONIC ENGINEERING

Name of the thesis	: Novel Clustering Methods for NeuroFuzzy Systems Design
Name of the student	: Yunis TORUN
Exam date	: September 2010

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Ramazan KOÇ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Savaş UÇKUN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Arif NACAROĞLU Co-Supervisor	Prof. Dr. Gülay TOHUMOĞLU Supervisor

Examining Committee Members Title and Name-surname	signature
Prof. Dr. Cüneyt GÜZELİŞ	____________________
Prof. Dr. Ferit Acar SAVACI	____________________
Prof. Dr. Gülay TOHUMOĞLU	____________________
Prof. Dr. Rauf Mirza BABAYEV	____________________
Asst. Prof. Dr. Nurdal WATSUJI	____________________

ABSTRACT

NOVEL CLUSTERING METHODS FOR NEUROFUZZY SYSTEMS DESIGN

TORUN Yunis
PhD in Electric Electronics Eng.
Supervisor: Prof. Dr. Gülay TOHUMĞLU
September 2010, 154 pages

In this thesis, novel clustering methods with optimized parameters in order to have NeuroFuzzy inference systems design are developed. A modified version of Simulated Annealing (SA) optimization and Subtractive Clustering (SC) techniques are adapted to Fuzzy System to form a fuzzy classifier (SASCFC) in order to obtain optimum fuzzy rule base, parameters and to find out most important inputs. Four distinct classifiers namely SASCFC-Type1, Type2, Type3 and Type4 are derived in order to form different optimization scenarios. Although there are some similarities in each type of SASCFC, Type4 has best classification performance because a hybrid feature selection algorithm is also developed in Type4. Two new NeuroFuzzy Classifiers (NFC) are proposed as NFC1 and NFC2. Initial structures of both classifiers are set up via Rival Penalized Competitive Learning (RPCL) based clustering. A new RPCL type gradient descent training algorithm is also proposed for the NFC2. Rule adaptation mechanism is embedded into training of the NFC2 that both parameters and structural optimization performed twice which enables to change the structure of classifiers by adding new rules and deleting unnecessary rules in training phase dynamically. It is found that the proposed classifiers, which are tested on some benchmarks problems, have good performance in comparing to their counterparts in recent literature.

Key Words; Classification, NeuroFuzzy Systems, Simulated Annealing, Clustering, Competitive Learning.

ÖZET

BULANIK SİNİR AĞLARININ TASARIMINDA YENİ KÜMELEME YÖNTEMLERİ

TORUN Yunis
Doktora Tezi, Elektrik Elektronik Müh Bölümü.
Danışman: Prof. Dr. Gülay TOHUMĞLU
Eylül 2010, 154 sayfa

Bu tezde, bulanık sinir ağlarının tasarımında eniyilenmiş parametreleri olan yeni kümeleme yöntemleri geliştirildi. Benzetilmiş Tavlama eniyileme (BT) yönteminin değiştirilmiş hali ile Çıkartımlı Kümeleme (ÇK) yöntemi bulanık sisteme uyarlanarak, eniyilenmiş bulanık kural tabanı, eniyilenmiş parametreler ve en önemli girişleri seçen bulanık sınıflandırıcı (BTÇKBS) türetildi. Farklı eniyileme senaryolarını göstermek amaçlı BTÇKBS-Tip1, Tip2, Tip3 ve Tip4 olmak üzere dört farklı sınıflandırıcı geliştirildi. Her bir BTÇKBS tip sınıflandırıcı bazı benzer özellikler taşımasına rağmen, Tip4 de melez bir özellik seçmede geliştirildiği için sınıflandırıcı daha iyi bir sınıflandırma performansına sahiptir. İki yeni Bulanık Sinir Ağı tabanlı Sınıflandırıcı (BSS), BSS1 ve BSS2 olarak ortaya konuldu. Her iki sınıflandırıcının ağ mimarilerinin kurulmasında Kaybedeni Cezalandırıcı Rekabetçi Öğrenme (KCRÖ) tabanlı kümeleme yöntemi kullanıldı. Yeni bir KCRÖ tabanlı hata geri yayılma öğrenme algoritması BSS2'nin parametrelerinin ayarlanması için geliştirildi. Kural uyum mekanizması BSS2'nin öğrenme sürecine dâhil edilerek hem parametrelerin hem de sınıflandırıcı yapısının öğrenme sürecinde dinamik olarak değiştirilmesi sağlandı. Bazı sınıflandırma problemlerinde test edilerek ortaya konan sınıflandırıcıların, yakın zamandaki çalışmalardaki benzer diğer sınıflandırıcılarla karşılaştırıldığında daha iyi bir sınıflandırma başarısına sahip olduğu bulundu.

Anahtar Kelimeler; Sınıflandırma, Bulanık Sinir Ağı Sistemleri, Benzetilmiş Tavlama, Kümeleme, Rekabetçi Öğrenme.

CONTENT

page

LIST OF TABLES

LIST OF FIGURES

page

LIST OF SYMBOLS

U	:	Membership matrix
J	:	Cost Function
u_{ij}	:	membership degree
r_a	:	Neighborhood radius
D_i	:	Density function for *i*th data
c_i	:	*i*th cluster center
ε^{up}	:	Upper Limit for Density
ε^{down}	:	Lower Limit for Density
$F(S_j)$	:	Cost Function for configuration S_j
F	:	Feature space
$S*$	:	Optimized Configuration
th	:	Output Threshold Value
fi	:	*i*th feature
c_{winner}	:	Winner cluster center
α_{winner}	:	Winner learning coefficient
c_{rival}	:	Rival cluster center
α_{rival}	:	Rival learning rate
R_k	:	*k*th rule
x_i	:	*i*th input
A_{kj}	:	Membership function for *k*th rule *j*th input
E_c	:	Error function for class *c*
p	:	Current pattern
P	:	P input-output pairs
y_c	:	Actual output for class *c*
∇w_s	:	Gradient vector for rule weight
∇c_{ij}	:	Gradient vector for membership function centers
$\nabla \sigma_{ij}$	:	Gradient vector for membership function spreads
φ_w	:	Learning rate for rule weight
φc	:	Learning rate for membership function center
$\varphi\ \sigma$	:	Learning rate for membership function spread
$errorc$	:	Error function for class c
$error_{rival}$	:	Error function for rival class *c*
$\nabla_{\sigma ij}^{winner}$	:	Gradient vector for membership function spreads for winner class
∇_{cij}^{winner}	:	Gradient vector for membership function centers for winner class

$\nabla_{\sigma ij}^{\mathrm{rival}}$	:	Gradient vector for membership function spreads for rival class
$\nabla_{\mathrm{cij}}^{\mathrm{rival}}$	:	Gradient vector for membership function centers for winner class
φ_c^{winner}	:	Learning rate for membership function centers for winner class
φ_c^{rival}	:	Learning rate for membership function centers for rival class
φ_σ^{winner}	:	Learning rate for membership function spread for winner class
φ_σ^{rival}	:	Learning rate for membership function spread for rival class

LIST OF ABREVIATIONS

ANFIS	:	Adaptive NeuroFuzzy Inference System
ANN	:	Artificial Neural Networks
BP	:	Back Propagation
CL	:	Competitive Learning
FCM	:	Fuzzy C-means
FIS	:	Fuzzy Inference System
FSCL	:	Frequency Sensitive Competitive Learning
GA	:	Genetic Algorithm
GD	:	Gradient Descent
Ibk	:	Nearest Neighbor Classifier
k-NN	:	K Nearest Neighbor
LMS	:	Least Mean Square
LSE	:	Least Square Estimator
MF	:	Membership Function
MNN	:	Multi-Layer Neural Network
NB	:	Bayesian Network Classifier
NFC1	:	NeuroFuzzy Classifiers Type1
NFC2	:	NeuroFuzzy Classifiers Type2
NN	:	Neural Network
RBF	:	Radial Basis Function
RMSE	:	Root Mean Square Error
RPCL	:	Rival Penalized Competitive Learning
SA	:	Simulated Annealing
SABPN	:	Simulated Annealing Back Propagation Network
SAFC	:	Simulated Annealing Fuzzy Classifier
SASCFC	:	Simulated Annealing and Subtractive Clustering Fuzzy Classifier
SC	:	Subtractive Clustering
SVM	:	Support Vector Machine
TS	:	Takagi Sugeno Fuzzy Inference System
TSK	:	Takagi Sugeno Kang Fuzzy Inference System

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Professor Gülay TOHUMOĞLU for her support and guidance during the preparation of this thesis.

CHAPTER 1

INTRODUCTION

In the second half of the 20th century the development of fast microprocessors enabled the design and implementation of expert–machine interaction based computation environments.. Soft computing is a practical framework for solving complex problems through the use of human expertise and a prior *k*nowledge about the problem at hand. The main subtitles in the soft computing are artificial neural networks and fuzzy inference systems [1].

Fuzzy Sets and Neurocomputation theories are playing important roles in the area of information processing, especially medical decision making [2-7] , estimation of rainfall [8] , chaotic time series predicting[9], modeling and control of nonlinear system [10-15], fault diagnosis [16, 17] and so on. NeuroFuzzy systems are robust solutions that search presentation of domain knowledge, reasoning on uncertainty, automatic learning, and adaptation. However the design and the definition of parameters effectiveness of these systems is a hard task. Construction ofNeuroFuzzy system suffers from no systematic way to describes how many nodes or membership function, which type inference system will be used, what is the optimal structure for a given problem, what is the optimal learning strategy and how can be computational time and learning stability improve.

In this thesis, it is proposed that designing optimum Fuzzy Classifiers by cooperation of Simulated Annealing (SA) and Subtractive Clustering (SC). The word of optimization for a fuzzy system means finding most proper membership functions centers, obtaining exact number of membership functions, acquiring proper rule base, searching correct level for output transformation and selecting most important inputs in case of large input subspace or feature redundancy. For these purposes, we developed four different models which we called as *SASCFS-Type1, SASCFS-Type2, SASCFS-Type3, SASCFS-Type4.* The proposed classifiers are tested with 12 classification problems which is commonly used in recent literature and they are compared with each others. According to the results, *SASCFC-Type4* has the best classification accuracy. The classifiers are also compared with recent works in the literature and seen that classification accuracies of *SAFCFC-Type1, Type2, Type3* and *Type4* classifiers are higher than their counterparts.

In additional to the SASCFC, two new NeuroFuzzy Classifiers are developed as NeuroFuzzy Classifier1 (NFC1) and NeuroFuzzy Classifier2 (NFC2). Initial structures of both classifiers are set up via Rival Penalized Competitive Learning (RPCL) based clustering method. Parameter tuning of the NFC1 is performed by gradient descent based back propagation batch training algorithm. Rule adaptation mechanism is embedded into training of the NFC2 that both parameters and structural optimization performed twice. It enables to change the structure of classifiers by adding new rules and deleting unnecessary rules, and improve the classifier performance. After initial structure is obtained by RPCL type clustering techniques, parameters of the NFC2 are tuned by incremental RPCL type back propagation algorithm. In fine tuning phase of structure, according to error criteria and rule firing counts criteria’s structure of the NFC2 are re construct and final structure is re tuned by back propagation algorithm.

This thesis is organized as follows. Chapter 2 describes some of the major research accomplishments about Fuzzy and Neural Networks and especially literature survey on NeuroFuzzy adaptation architectures, training methodologies which include parametric and structural identification of NeuroFuzzy system. Chapter 3 briefly describes the common Fuzzy Inference Systems, Neural Network, and common hybridization architectures of NeuroFuzzy systems. Clustering techniques and common NeuroFuzzy Classifiers for pap smear classification problem are demonstrated in Chapter 4 and Chapter 5 respectively. The theory of the SA, modified version of the SA and the proposed SASCFC are given in Chapter 6. The proposed NFC’s are given in Chapter 7. Finally Chapter 8 concludes the thesis.

CHAPTER 2

LITERATURE SURVEY

In this chapter, brief historical development of NeuroFuzzy systems and some valuable studies which aim to design NeuroFuzzy systems more systematically are reviewed. According to recent literature, basic gaps, unsolved problems and untried techniques for constructing of NeuroFuzzy systems are stated at the end of current chapter.

2.1 Brief Survey on Fuzzy System and Neural Network.

First Neural Network (NN) concept was proposed by McCulloch and Pitss in 1943. Their networks act as a certain logic function and the network has no training ability. Hebb described a learning algorithm which was based on the adjustment of the synaptic weights of the neurons in 1949, his work has had a major impact on the later works. Rosenblatt developed the concept of perceptron in the literature. Widrow and Hoff trained the perceptron via LMS (Least Mean Square) learning rule. Werbos developed back propagation algorithm for training the multilayer feed forward (FF) perceptrons but despite of the power of the algorithm it didn’t call attention the NN researcher until 1986. Hopfield proposed the Recurrent NN architecture that network can store information and is able to perform the function of data storage and retrieval. Kohenen presented the self organizing feature map in 1982. Strategy is a kind of the unsupervised learning which is based on competitive learning. Rumelhart re described the back propagation algorithm in 1986, his work has a great impact of later works which dealing with training the networks. Sivilotti showed the realizability of the NN by VLSI realization of the NN in 1987. Broomhead proposed Radial Basis NN which was a smooth passing from Neural Network to Fuzzy Inference System in 1988. Fuzzy inference systems are one of the most famous applications of fuzzy logic and fuzzy set theory which were developed by Zadeh in 1965 [18]. In the last few decades the development of fast microprocessors and embedded processors have enabled the design and implementation of fuzzy inference systems on real world problems such as achieving classification tasks [19, 20], process control [10], decision support [2, 7], pattern recognition [20], robotics [21], bioinformatics [4, 5] and so on.

2.2 Survey on NeuroFuzzy Systems

In the last decades hybridization of Fuzzy Inference System and Neural Network take attention of researchers. Jang (1993) proposed to use the Adaptive NeuroFuzzy Inference System architecture to improve the performance of the fuzzy system [22]. The performance of the fuzzy system relies on two important factors: knowledge acquisition and the availability of human experts. For the first problem, Jang proposed the ANFIS to solve the automatic elicitation of the knowledge in the form of fuzzy if-then rules. For the second problem, that is how the fuzzy system is constructed without using human experts; a learning method based on a special form of gradient descent was used.

Jang and Sun [23] have shown that fuzzy systems are functionally equivalent to a class of radial basis function (RBF) networks, based on the similarity between the local receptive fields of the network and the membership functions of the fuzzy system. Hayashi and Buckley [24] proved that any rule-based fuzzy system maybe approximated by a neural network and also neural network (feedforward, multilayered) may be approximated by a rule-based fuzzy system. Zhang and Kandel [25] proposed a compensatory NeuroFuzzy system for optimization of the fuzzy logic reasoning and for selecting optimal fuzzy operators. Chakraborty et al. used integrated feature analyzing to optimize the NeuroFuzzy system in 2001 [26].

Azeem et al. proposed a generalized fuzzy model (GFM) by extension of Jang’s ANFIS which encompasses both Takagi-Sugeno (T-S) model and composional rule of inference (CRI) model and they proposed a neural network architecture to determine the parameter of the model [27-29]. Lee and Wang [30] used mapping constrained agglomerative clustering techniques to identification of ANFIS structure proposed Type I, II, III type Self Adapting NeuroFuzzy System (SANFIS) in 2005. Javonovic and Relijin, in 2004 [31] proposed a modified ANFIS structure which called as MANFIS with number of rules equals to the number of input membership functions. Without a structural identification of NeuroFuzzy network, Figueiredo et al. [32] proposed a competitive based learning algorithm with offline and online learning phase. Treesatayapun and Uatrongit [33] proposed an adaptive fuzzy rule emulated network with gradient descent algorithm to train the network parameter in 2005. Ouyang et al. developed a NeuroFuzzy network technique to extract TSK type fuzzy rules from given a input-output data set for system modeling in 2005 [34]. They used fuzzy clusters which are generated incrementally from dataset and similar cluster are merged dynamically together input-similarity, output-similarity and output variance tests.

A serious problem facing fuzzy system applications is how to deal with this rule explosion problem. One approach to deal with this difficulty is use hierarchical fuzzy systems. Yu and Marco in 2005 [35], proposed back propagation like algorithm for training the membership of the hierarchical fuzzy neural network which can use less rules to model nonlinear systems with high accuracy. They proposed time varying learning rate which is calculated from data sets and structure. Obviously, the performance of a NeuroFuzzy system depends the network parameters, network structure like numbers of input-output, type and numbers of membership functions and number of training epoch. Zanchettin et al. [36] listed the choices of basic parameters of NeuroFuzzy system and reached and compared their influence to performance in 2005.

A fuzzy inference system can be built by using expert knowledge heuristically. However, expert knowledge may not be adequate to construct a model. In this situation, some systematic ways have been proposed in order to construct fuzzy if-then rule base [37-40]. Some works have been made on how fuzzy system could be extracted from numerical data with genetic algorithm [41, 42], tabu search [43, 44], decision trees [45], evolutionary programming [46, 47] techniques.

Clustering methods are widely used in structure learning phase of both neural network and fuzzy inference based systems as fuzzy c-means, k-means clustering, mountain clustering, subtractive clustering, and agglomerative clustering. Detailed review of clustering algorithms is addresses in [48]. One of the ways of extracting fuzzy rules from numeric data is the use of Subtractive Clustering (SC) method which was proposed by Chiu [49] as a modification of Mountain clustering method [50]. The SC is an unsupervised clustering method because it is not necessary to know how many clusters will be formed. According to review work of Guillaume, there is no theoretical guidance on how optimum SC parameters should be chosen for Fuzzy System [51]. In work of Han M. et al. [52], the input membership functions of Fuzzy Neural Network are firstly obtained with fuzzy space partition, and then the SC algorithm is utilized to get kernel rules and the importance of every rule. A NeuroFuzzy model which has been identified by the SC algorithm has been developed for autonomous parallel parking of a car-like mobile robot in [53]. After constructing the rule base of NeuroFuzzy system by the SC, similar membership functions are merged in order to remove the redundant rules in [54]. Zhao et al. [55], used particle swarm optimization algorithm so as to find the optimal membership functions (MFs), which are initially found by the SC, and consequent parameters of the rule base. Initial membership functions are obtained by the SC method then are tuned by the means of differential evolution in the work of Efektari et al. [56]. In an another work of the same authors [57] , Genetic Algorithm (GA) is used to construct compact fuzzy model by selecting more efficient inputs and to determine the optimum number of rules by finding the optimum SC radius. Another optimization work in which the Nelder-Mead optimization is used, aiming to tune the SC parameters for NeuroFuzzy model, is demonstrated in [58]. The discussions on the effects of parameters of the SC such as squash factor, cluster radius, accept ratio and reject ratio on fuzzy model performance are given in [59]. They proposed that the performance of the model is very sensitive to the cluster radius while the accept ratio and the reject ratio do not have big influence on the performance of model.

Competitive learning (CL) clustering, which is a kind of adaptive version of classical k-mean clustering method, has been developed for unsupervised learning in artificial neural networks and provided us a promising tool for clustering, pattern recognition and vector quantization[60]. However CL has a problem called as dead unit problems [61]. Frequency Sensitive Competitive Learning algorithm (FSCL) tackles dead unit problem by reducing the learning rate of the frequent winners [62]. Although FSCL solves dead unit problem there is another problem which selecting appropriate cluster numbers still opens until the penalization strategy is adopted to classical FSCL, namely Rival Penalized Competitive Learning Clustering (RPCL) was proposed by Xu et al. in 1992[63]. Some studies on improving performance of RPCL are found in [60, 64] and its application in construction of Radial Basis Function network is given[65].

Simulated Annealing (SA) which is an iteratively search algorithm for solving hard combinatory problems, was firstly introduced by Metropolis in 1953 [66]. After the work of Kirkpatrick [67] who applied the SA to solve a combinatory optimization problem successfully, it has been commonly used in the optimization problems. In recent literature, the SA has been widely used in the optimization of artificial intelligence tools [20, 68, 69]. In the work of [70], the SA is used for the optimal tuning of the parameters of a fuzzy controller for a network-based control system. Alizadeh & Ghazanfari [71] combine the SA with chaos concepts in order to construct Fuzzy Cognitive Map (FCM) automatically. In the work of Mohamadi et al. [20], the SA algorithm is used to find optimum fuzzy rule base by modificating, deleting and creating new rules iteratively. Lee et al. [68] used the SA to achieve global optima and improve the convergence speed of multilayer perceptron training in which the gradient descent local optimization method is used. Lin et al. [72], used the SA algorithm both in optimizing the parameters of a back propagation neural network and selecting proper feature for classification task.

In addition to the optimization of classifiers' parameters, feature selection or feature reduction algorithms are widely studied in recent literature [73, 74]. In the case of large number of input space, feature selection does not only provide less computational time but also helps to improve classification accuracy. In the presence of many irrelevant features, modeling or clustering tasks tend to over fit training data [75]. In general, in data mining tasks, feature selection methods can be divided into two categories as filter types and wrapper types. The filter methods try to remove irrelevant or noisy features before it is used by learning algorithms [76]. The wrapper approach uses a heuristic search that evaluates the quality of the feature subset by prediction accuracy of the induction algorithm [77]. Non deterministic methods such as SA and GA based feature selection which search the subspace using stochastic search algorithms, can be regarded as wrapper type algorithms. Sean & Thunsun [78] discussed the filter types feature selection methods as Relief-F, mutual correlation based feature selection and gene selection for three different fuzzy classifiers; TS-

FIS, ANFIS and Fuzzy Nearest Neighbor classifiers. Pizzi& Pedrycz [79] proposed stochastic feature selection algorithm for feature reduction and fuzzy integral for classification. Chiang & Ho introduced [80] rough based feature selection for the classification task of micro array data with Radial Basis Function Neural Network (RBFN). Fisher score (F-score) based feature selection for breast cancer classification with Support Vector Machine (SVM) is performed in [81]. Karabatak&Ince [82] discussed the same issue by using association rules (AR) which is used for dimensionality reduction for breast cancer classification with neural network based classifier. A symmetric uncertainty based filter type feature selection for intrusion detection problem with k-NN and fuzzy k-NN classifiers is proposed in [83]. A Wrapper type feature selection algorithm is implemented to select features for stock trend prediction by using back propagation Neural Network, SVM and k-NN classifiers in [84].

2.3 What Are the Gaps with NeuroFuzzy Systems

Although above mentioned valuable studies have aimed to solve of finding systematic ways for constructing Fuzzy and NeuroFuzzy systems, there are some gaps, unreciprocated questions and unproven methodologies that;

i. There is no guidance study for which clustering algorithms were most successful for Fuzzy and NeuroFuzzy based cervical cancer diagnosis.

ii. Comparisons of artificial intelligence tools as Fuzzy, ANFIS, NN and RBF classifiers for cervical cancer diagnosis haven’t been performed in literature.

iii. There is no methodology or algorithm to select SC parameters. Despite Clustering indexing algorithm tries to find most proper cluster number, they aren’t successful in real world problems.

iv. In classification problem, designing classifier task concerns structural optimization, parameter learning and feature selection. There is no such an algorithm which is capable of constructing optimum fuzzy classifier by dealing not only structure and parameter learning but also feature selection.

v. Wrapper and filter types feature selection algorithm have some benefit and disadvantages. There is no successful study which combines these two algorithms to get more robust feature selection.

vi. Although GD based learning algorithms are broadly used in literature; it is possible to trap local optima by GD optimization. Metaheuristic optimization such as the SA can solve the problem of trapping local optima.

vii. There is no satisfactory work that aims to improve the SA performance by means of decreasing numbers of iteration while accessing global optima.

viii. In some studies RPCL clustering techniques are used structural identification phase of Neural Network. However, they aren’t used structural identification phase of Fuzzy and NeuroFuzzy systems.

ix. While competitive learning is a widely used technique for parameter learning of NeuroFuzzy system, there is no schema to adopt RPCL into the conventional GD optimization.

2.4 Main Contributions of This Thesis

According to the gaps mentioned previous subsection, this thesis aims to contributes to current literature with constructing Fuzzy and NeuroFuzzy systems based classifiers as;

i. Frequently used clustering algorithms in structure identification phase of Fuzzy and NeuroFuzzy systems are tested and compared on cervical cancer detection problem. K-means, Fuzzy C-means, and SC give testing classification accuracies as 82%, 58 % and 80% respectively.

ii. Fuzzy, ANFIS, NN, and RBF based classifiers are performed to detect cervical cancer. Effects of the neighborhood radii of subtractive clustering on initial rule structures are analyzed for Fuzzy and ANFIS based classifiers. It is also analyzed the effects of numbers of neurons for NN and spread of neurons for RBF based pap-smear classifier.

iii. SA optimization method is applied to find optimum radius of neighborhood which is most significance parameter of the SC

iv. A systematic and compact algorithm which able to construct optimum fuzzy classifier by dealing not only structure and parameter learning but also feature selection is developed.

v. A hybrid feature selection method, which merges simple filter and wrapper approaches, is developed.

vi. The SA algorithm which guarantee for finding global optima is applied into Fuzzy system for classification tasks.

vii. A modified version of conventional SA which aims to reach global optima with lower iteration is proposed.

viii. RPCL clustering technique is used in structural identification phase of NeuroFuzzy based classifiers.

ix. A new GD algorithm that mimics RPCL learning strategy is developed for NeuroFuzzy based classifier.

CHAPTER 3

NEUROFUZZY SYSTEMS

In this chapter, background of common Fuzzy Inference System, Neural Network, hybridization of these two tools according to recent literature and learning strategies are briefly introduced. This brief introduction is the beginning point of our study which aims to develop new Fuzzy and NeuroFuzzy systems.

Both Neural Network and Fuzzy Inference System are powerful soft computing tools that there are wide applications in to real world problems. Neural networks do not provide a strong scheme for knowledge representation, while fuzzy logic systems not possess capabilities for automated learning. For example, fuzzy systems are appropriate if sufficient expert knowledge about the model is available, while neural systems are useful if sufficient process data are available or measurable. Both approaches build nonlinear systems based on bounded continuous variables, the difference being that neural systems are treated in a numeric quantitative manner, whereas fuzzy systems are treated in a symbolic qualitative manner.

3.1 Fuzzy Inference System (FIS)

The Fuzzy Inference System (FIS) is a popular computing framework based on the concepts of fuzzy set theory, fuzzy if-then rules, and fuzzy reasoning. The basic structure of a fuzzy inference system which is shown in Fig 3.1 consists of four conceptual components: knowledge base, fuzzification interface, inference engine, and defuzzification interface [10]. The knowledge base contains all the knowledge and it comprises a fuzzy decision rule base and a data base. The data base is the declarative part of the knowledge base which describes definition of objects (facts, terms, and concepts) and definition of membership functions used in the fuzzy rules. The fuzzy rule base is the procedural part of the knowledge base which contains information on how these objects can be used to infer new control actions. The inference engine is a reasoning mechanism which performs inference procedure upon the fuzzy rules and given conditions to derive reasonable control fuzzy decision. The fuzzification interface (or fuzzifier) defines a mapping from a real-valued space to a fuzzy space, and the defuzzification interface (or defuzzifier) defines a mapping from a fuzzy space defined over an output universe of discourse to a real-valued space. The fuzzifier converts a crisp value to a fuzzy number while the defuzzifier converts the inferred fuzzy conclusion to a crisp value.

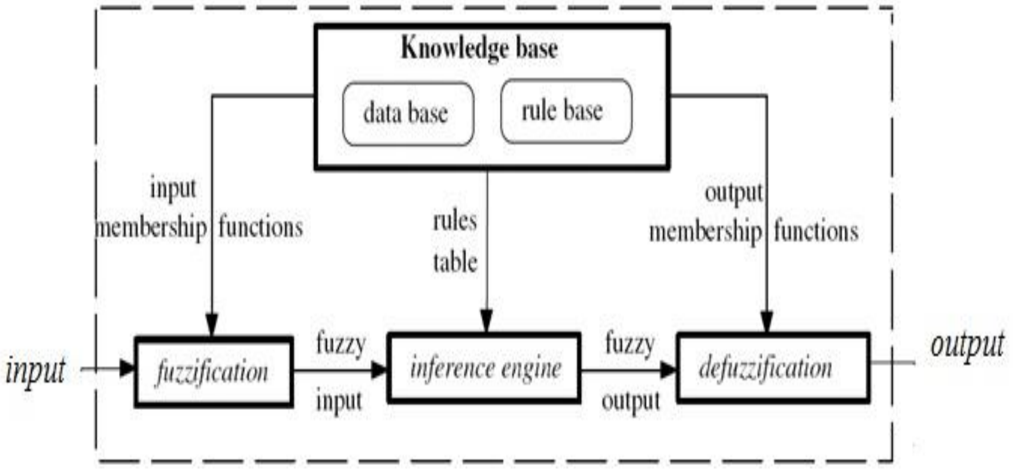


Figure 3.1 Block diagram of Fuzzy Inference System [10]

According to the literature we can classify the types of the fuzzy inference system Mamdani type or conventional fuzzy inference system and Sugeno type fuzzy inference system.

3.1.1 Mamdani Type Inference

These methods of conventional fuzzy system are essentially heuristic and model free. The fuzzy "IF-THEN" rules are obtained based on an expert decision making's action or knowledge. Design of such systems suffers from lack of systematic and consistent approaches. According to the given rule base in Eqn. 3.1, Fig. 3.2 and 3.3 shows how a two-rule fuzzy inference system of the Mamdani type derives which proposed in [85] the overall output z when subject to two crisp input x and y. As seen in the Fig. 3.3 a fuzzy *min* operator is used for Fuzzy AND operator and *max* operator for Fuzzy OR operator. If we adopt *product* and *max* for the Fuzzy AND

and OR operator respectively and use max-product composition instead of the original *max*- *min* composition then the resulting Fuzzy reasoning is shown in Fig. 3.3

$$\begin{aligned} &\boldsymbol{R_1}\ \boldsymbol{;IF}\quad x \quad \boldsymbol{is}\quad A_1 \quad \boldsymbol{AND}\quad y \quad \boldsymbol{is}\ B_1 \quad \boldsymbol{THEN}\quad z \quad \boldsymbol{is}\quad C_1 \\ &\boldsymbol{R_2}\ \boldsymbol{;IF}\quad x \quad \boldsymbol{is}\quad A_2 \quad \boldsymbol{AND}\quad y \quad \boldsymbol{is}\ B_2 \quad \boldsymbol{THEN}\quad z \quad \boldsymbol{is}\quad C_2 \end{aligned} \tag{3.1}$$

Finally, the Fuzzy output converted to the crisp value with defuzzification. The most frequently defuzzification algorithm is the centroid of area, which is defined as;

$$z_{COA} = \frac{\int_Z \mu_{C'}(z) z.dz}{\int_Z \mu_{C'}(z) dz} \tag{3.2}$$

where $\mu_{C'}(z)$ is the aggregated output membership function.

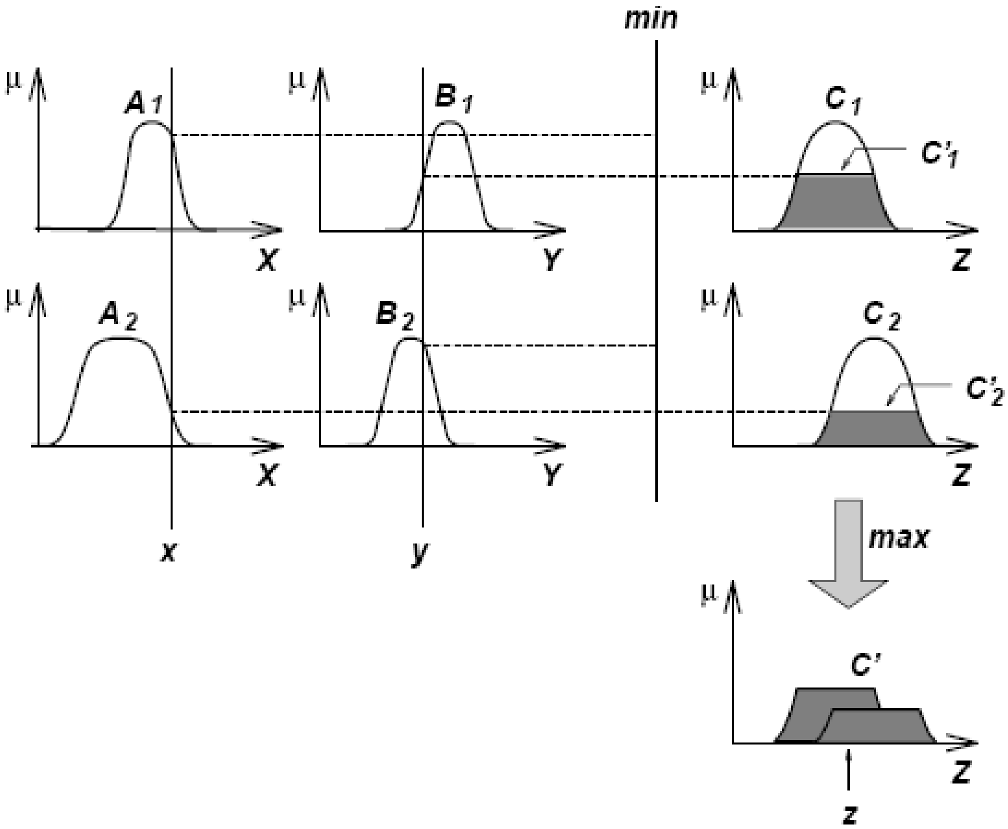


Figure 3.2 Mamdani type Fuzzy System using *min* and *max* for Fuzzy AND and OR operator, respectively

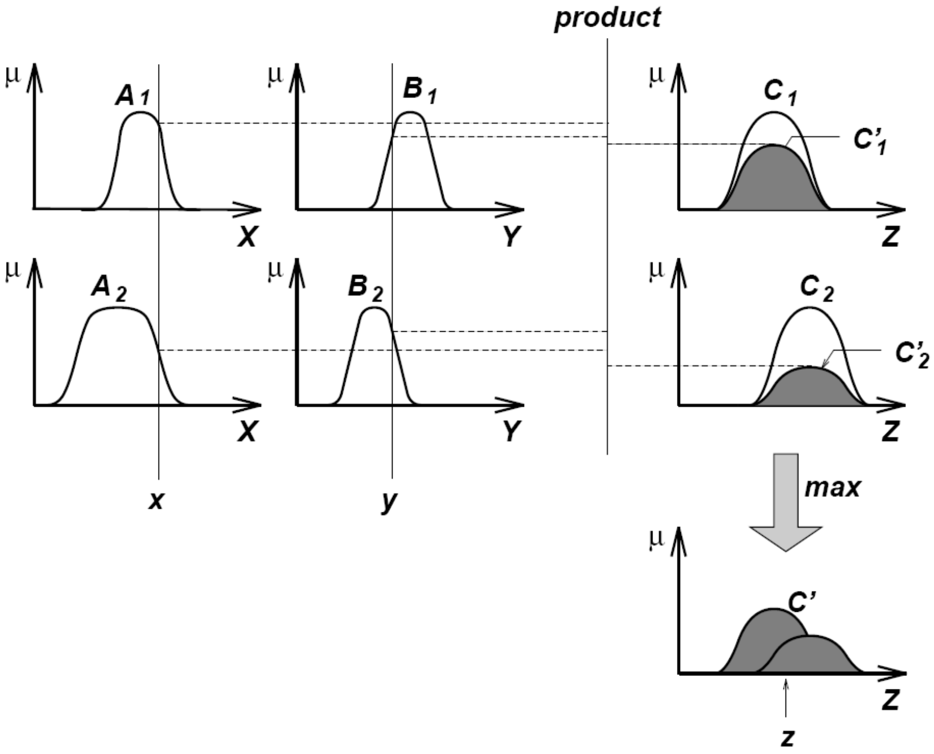


Figure 3.3 Mamdani type Fuzzy System using *product* and *max* for Fuzzy AND and OR operator, respectively

3.1.2 Sugeno Type Inference

The Sugeno fuzzy model (also known as the Tagaki and Sugeno (TS) fuzzy model) was proposed by Takagi, Sugeno and Kang [86, 87] in an effort to develop systematic approach to generating fuzzy rules from a given data set. Typical fuzzy rule in Sugeno fuzzy model has the form ;

$$\boldsymbol{R_i} \ \boldsymbol{; IF} \ x \ \boldsymbol{is} \ A \ \boldsymbol{AND} \ y \ \boldsymbol{is} \ B \ \boldsymbol{THEN} \ z = f(x,y) \tag{3.3}$$

where A and B are fuzzy sets in the antecedent while $z = f(x,y)$ is a crisp function in the consequent. When $f(x,y)$ is a first order polynomial the resulting fuzzy model is called *first-order* Sugeno fuzzy model and when $f(x,y)$ is a constant resulting fuzzy model is called *zero-order* Sugeno fuzzy model which is proposed in [87]. Fig. 3.4. shows the fuzzy reasoning procedure for a first-order Sugeno Fuzzy model. The Aggregator block and defuzzifier block are replaced by the operation of weighting average, thus avoiding time consuming procedure of defuzzification.

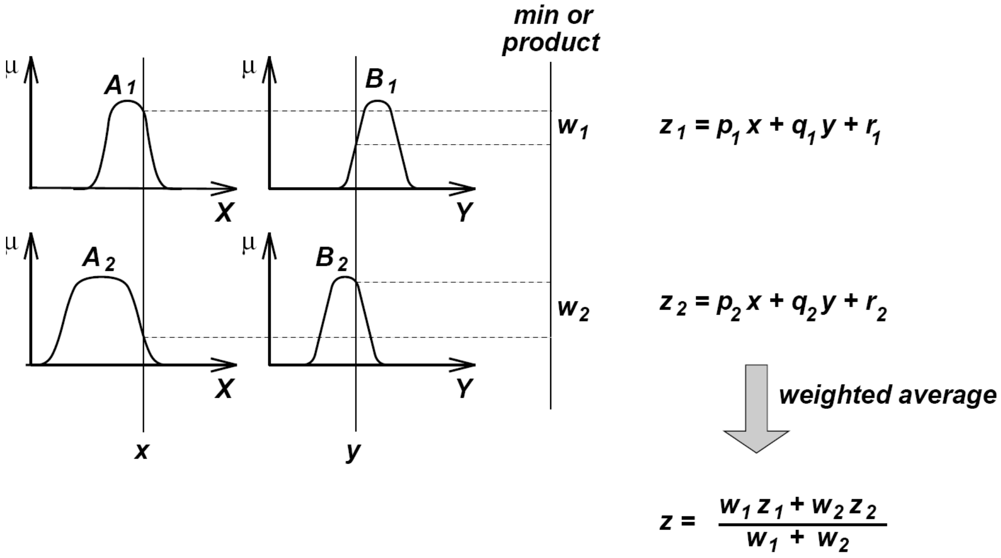

min or product
μ
A1
B1
X
Y
A2
B2
w1
w2
x
y
$z_1 = p_1x + q_1y + r_1$
$z_2 = p_2x + q_2y + r_2$
weighted average
$z = \frac{w_1z_1 + w_2z_2}{w_1 + w_2}$

Figure 3.4 Sugeno type Inference System

3.2 Artificial Neural Network

3.2.1 Neuron model and Single Layer Neural Network

The human brain provides proof of the existence of massive neural networks that can succeed at those cognitive, perceptual, and control tasks in which humans are successful. The brain is capable of computationally demanding perceptual acts (e.g. recognition of faces, speech) and control activities (e.g. body movements and body functions). The advantage of the brain is its effective use of massive parallelism, the highly parallel computing structure, and the imprecise information-processing capability. The human brain has been estimated to contain 50–100 billion (10^{11}) neurons Each neuron is a cell (Fig. 3.5) that uses biochemical reactions to receive, process, and transmit information [88]

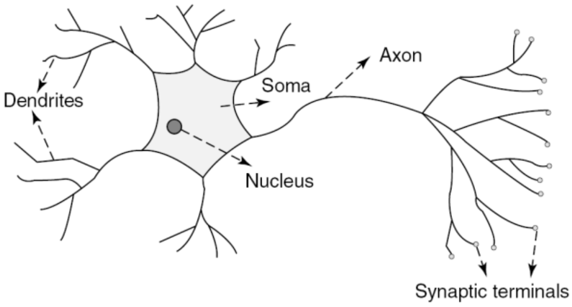


Figure 3.5 Biological Neuron Model

Treelike networks of nerve fibers called *dendrites* are connected to the cell body or soma, where the cell nucleus is located. Extending from the cell body is a single long fiber called the *axon*, which eventually branches into strands and substrands, and is connected to other neurons through synaptic terminals or synapses. The transmission of signals from one neuron to another at synapses is a complex chemical process in which specific transmitter substances are released from the sending end of the junction. The effect is to raise or lower the electrical potential inside the body of the receiving cell. If the potential reaches a threshold, a pulse is sent down the axon and the cell is 'fired'.

Artificial neural networks (ANN) have been developed as generalizations of mathematical models of biological nervous systems. A first wave of interest in neural Networks (also known as *connectionist models* or *parallel distributed processing*) emerged after the introduction of simplified neurons by McCulloch and Pitts (1943).

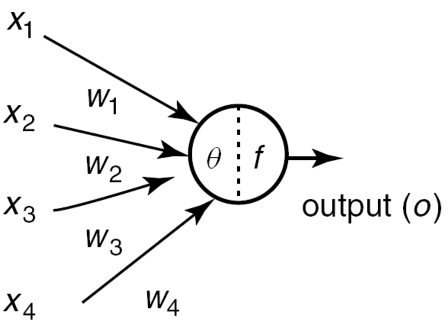


Figure 3.6 Artificial Neuron model

The basic processing elements of neural networks are called *artificial neurons*, or *simply neurons* or *nodes*. In a simplified mathematical model of the neuron, the effects of the synapses are represented by connection weights that modulate the effect of the associated input signals, and the nonlinear characteristic exhibited by neurons is represented by a transfer function. The neuron impulse is then computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance to the chosen learning algorithm.

According to the Fig 3.6 the neuron output signal *O* is given by the following relationship:

$$O = f_\theta(net) = f\left(\sum_{j=1}^{p} w_j x_j\right) \tag{3.4}$$

where w_j is the weight vector, and the function *f(net)* is referred to as an activation (transfer) function. The variable *net* is defined as a scalar product of the weight and input vectors,

$$net = w^T x = w_1 x_1 + ... + w_p x_p \tag{3.5}$$

where T is the transpose of at matrix, and, in the simplest case, the output value *O* is computed as,

$$O = f_\theta(net) = \begin{cases} 1 & if \quad w^T x \geq \theta \\ 0 & otherwise \end{cases} \tag{3.6}$$

where θ is called the threshold level; and this type of node is called a *linear threshold unit*. Since the step function is discontinuous at one point and flat et al. points, it is not suitable for learning procedure based on gradient descent. To overcome this difficulty sigmoid function can be used ;

$$O = f(net) = \frac{1}{1+e^{-w^T x}} \qquad for \;\; \theta{=}0 \tag{3.7}$$

which is a continuous and differentiable approximation to the step function.

3.2.2 Multilayer Neural Network

The basic architecture consists of three types of neuron layers: input, hidden, and output layers [89]. In feed-forward networks, the signal flow is from input to output units, strictly in a feed-forward direction. The data processing can extend over multiple (layers of) units, but no feedback connections are present. A feed forward multi-layer neural network (MNN) has one input layer, one output layer and a number of hidden layers between them. For illustration purposes, consider a MNN with one hidden layer in Fig. 3.7.

The input-layer neurons do not perform any computations; they merely distribute the inputs x_1 to the weights w_{ij}^h of the hidden layer. In the neurons of the hidden layer, first the weighted sum of the inputs is computed according to the Eqn. 3.5;

$$net_j = \left(w_j^h\right)^T x, \qquad j = 1,2,......m \tag{3.8}$$

then is passed through an activation function as described in Eqn. 4.6 and 4.7. The neurons in the output layer are linear and compute the weighted sum of their inputs;

$$y_1 = \sum_{j=1}^{h} w_{jl}^o net_j \tag{3.9}$$

A network with one hidden layer is sufficient for most approximation tasks. More layers can give a better fit, but the training takes longer [11]. Choosing the right number of neurons in the hidden layer is essential for a good result. Too few neurons give a poor fit, while too many neurons result in over-training of the net (poor generalization to unseen data). A compromise is usually sought by trial and error methods [11]

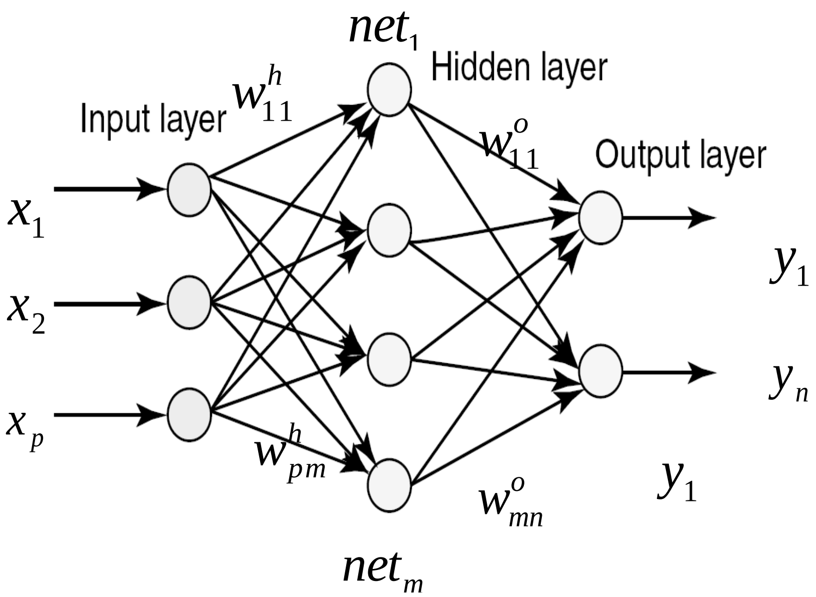


Figure 3.7 A feed forward with one hidden layer

3.2.3 Radial Basis Function Neural Network (RBFNN)

RBFNNs are generally considered as a smooth transition between fuzzy logic and neural networks. Jang has showed the functional equivalence between radial basis function networks and fuzzy inference systems in 1993 [12] and according to his work if the aggregation method, number of receptive units in the hidden layer and the constant terms are equal to those of a Fuzzy Inference System (FIS), then there exists a functional equivalence between RBFNN and FIS. Structurally, RBFNN is two layer network with an architecture depicted in Fig. 3.8. Each neuron in the hidden layer provides a degree of membership value for the input pattern with respect to the basis vector of the receptive unit itself. The output layer is comprised of linear combiners. Radial basis networks may require more neurons than standard feed-forward back propagation networks, but often they can be designed with lesser time. They perform well when many training data are available [90].

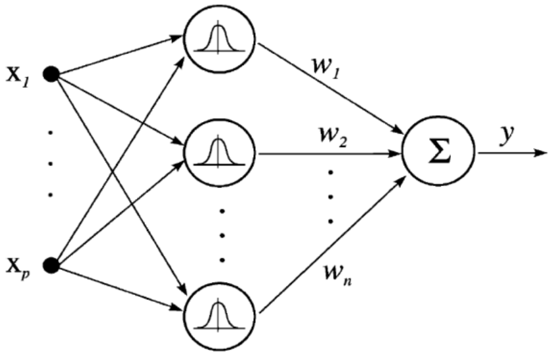


Figure 3.8 Radial Basis Function Neural Network

The output of the neuron is computed as;

$$y = f(x) = \sum_{i=1}^{m} w_i \phi_i(x) \tag{3.10}$$

where *m* is the numbers of hidden neurons in the hidden layer and ϕ_i is the basis function and usual choice for the basis function is the Gaussian function;

$$\phi_i(x) = \exp\left(- \frac{\left\| x \quad - \quad c_i \right\|^2}{2\sigma_i^2} \right) \tag{3.11}$$

The adjustable parameters of the RBFNN are synaptic weights which are only present in the output layer and center c_i and σ_i radii of the basis function.

3.3 NeuroFuzzy System

In most fuzzy systems, fuzzy rules were obtained from the human expert. However, every expert does not want to share his or her knowledge especially in the medical case and there is no standard method that exists to utilize expert knowledge. As a result, ANNs were incorporated into fuzzy systems to be able to acquire knowledge automatically by learning algorithms. The learning capability of the NNs was used for automatic fuzzy if-then rules generation [91].

Both neural networks and fuzzy systems are dynamic, parallel processing systems that estimate input–output functions. They estimate a function without any mathematical model and *learn from experience* with sample data. A fuzzy system adaptively infers and modifies its fuzzy associations from representative numerical samples. Neural networks, on the other hand, can *blindly* generate and refine fuzzy rules from training data [89]. Fuzzy sets are considered to be advantageous in the logical field, and in handling higher order processing easily. The higher flexibility is a characteristic feature of neural nets produced by learning and, hence, this suits data-driven processing better [92].

3.3.1 Why do We Need to Combine Neuro and Fuzzy Approaches?

The integration of neural and fuzzy systems leads to a symbiotic relationship in which fuzzy systems provide a powerful framework for expert knowledge representation, while neural networks provide learning capabilities and exceptional suitability for computationally efficient hardware implementations.

NeuroFuzzy computing is a judicious integration of the merits of neural and fuzzy approaches, enables one to build more intelligent decision-making systems. This incorporates the generic advantages of artificial neural networks like massive parallelism, robustness, and learning in data-rich environments into the system. The modeling of imprecise and qualitative knowledge as well as the transmission of uncertainty is possible through the use of fuzzy logic. Besides these generic advantages, the NeuroFuzzy approach also provides the corresponding application specific merits.

3.3.2 Neural network and Fuzzy System hybridizations.

Buckley and Hayashi [24] have classified fuzzified neural networks as follows. Networks can possess 1) real number inputs, fuzzy outputs, and fuzzy weights; 2) fuzzy inputs, fuzzy outputs, and real number weights; 3) fuzzy inputs, fuzzy outputs, and fuzzy weights. There are several works related to the integration of neural networks and fuzzy inference system. In generally, hybridization can be formulated into three main categories [90]; cooperative, concurrent and integrated NeuroFuzzy models. In the simplest way, a cooperative model can be considered as a preprocessor wherein artificial neural network (ANN) learning mechanism determines the fuzzy inference system (FIS) membership functions or fuzzy rules from the training data. In a concurrent model, neural network assists the fuzzy system continuously (or vice versa). Such combinations do not optimize the fuzzy system but only aids to improve the performance of the overall system. In an integrated model, neural network learning algorithms are used to determine the parameters of fuzzy inference systems. Integrated NeuroFuzzy systems share data structures and knowledge representations. A fuzzy inference system can utilize human expertise by storing its essential components in rule base and database, and perform fuzzy reasoning to infer the overall output value. The derivation of *if- then* rules and corresponding membership functions depends heavily on the *a priori* knowledge about the system under consideration. However there is no systematic way to transform experiences of knowledge of human experts to the knowledge base of a fuzzy inference system. There is also a need for adaptability or some learning algorithms to produce outputs within the required error rate. On the other hand, neural network learning mechanism does not rely on human expertise. Due to the homogenous structure of neural network, it is hard to extract structured knowledge from either the weights or the configuration of the network. The weights of the neural network represent the coefficients of the hyper-plane that partition the input space into two regions with different output values. If we can visualize this hyper-plane structure from the training data then the subsequent learning procedures in a neural network can be reduced. However, in reality, the a priori knowledge is usually obtained from human experts, it is most appropriate to express the knowledge as a set of fuzzy if-then rules, and it is very difficult to encode into a neural network.

There are several works and proposed architectures with integrated NeuroFuzzy systems in the literature, basically depends on the type of inference system. We can classify the integration of NeuroFuzzy systems in two subtitles with Mamdani and Sugeno as classified for fuzzy inference system [90].

3.3.2.1 Mamdani Integrated NeuroFuzzy Systems

Mamdani NeuroFuzzy system uses architecture of Mamdani type inference system, a supervised learning technique (back propagation learning) is applied to learn the parameters of the membership functions. Architecture of Mamdani NeuroFuzzy system is illustrated in Fig.3.9. The detailed function of each layer is as follows:

Layer -1*(input layer):* No computation is done in this layer. Each node in this layer, which corresponds to one input variable, only transmits input values to the next layer directly. The link weight in layer 1 is unity.

Layer-2 *(fuzzification layer):* Each node in this layer corresponds to one linguistic label (excellent, good, etc.) to one of the input variables in layer 1. In other words, the output link represents the membership value, which specifies the degree to which an input value belongs to a fuzzy set, is calculated in layer 2. A clustering algorithm will decide the initial number and type of membership functions to be allocated to each of the input variable. The final shapes of the MFs will be tuned during network learning.

Layer-3 *(rule antecedent layer):* A node in this layer represents the antecedent part of a rule. Usually a T-norm operator is used in this node. The output of a layer 3 node represents the firing strength of the corresponding fuzzy rule.

Layer-4 *(rule consequent layer):* This node basically has two tasks. To combine the incoming rule antecedents and determine the degree to which they belong to the output linguistic label (high, medium, low, etc.). The number of nodes in this layer will be equal to the number of rules.

Layer-5 *(combination and defuzzification layer):* This node does the combination of all the rules consequents using a T-conorm operator and finally computes the crisp output after defuzzification.

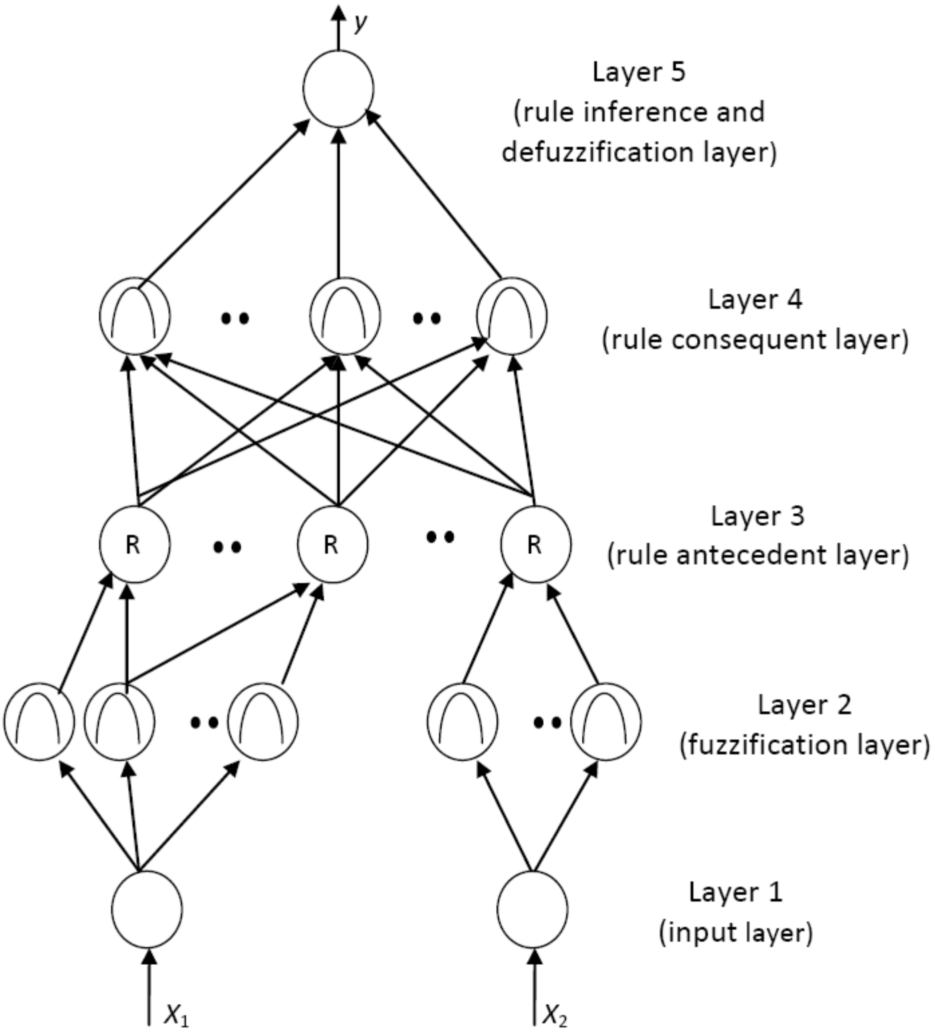


Figure 3.9 Mamdani NeuroFuzzy system [90]

3.3.2.2 Takagi-Sugeno Integrated NeuroFuzzy system

As Sugeno type inference, the consequent part of the inference system is a polynomial instead of a fuzzy membership function in contrast to Mamdani model. Takagi-Sugeno NeuroFuzzy systems as shown in Fig. 3.10 uses polynomial at the antecedent while fuzzy memberships function at the antecedent part. Learning procedure of this type of NeuroFuzzy system combines a mixture of back propagation to learn the membership functions and least mean square estimation to determine the coefficients of the linear combinations in the rule's conclusions.

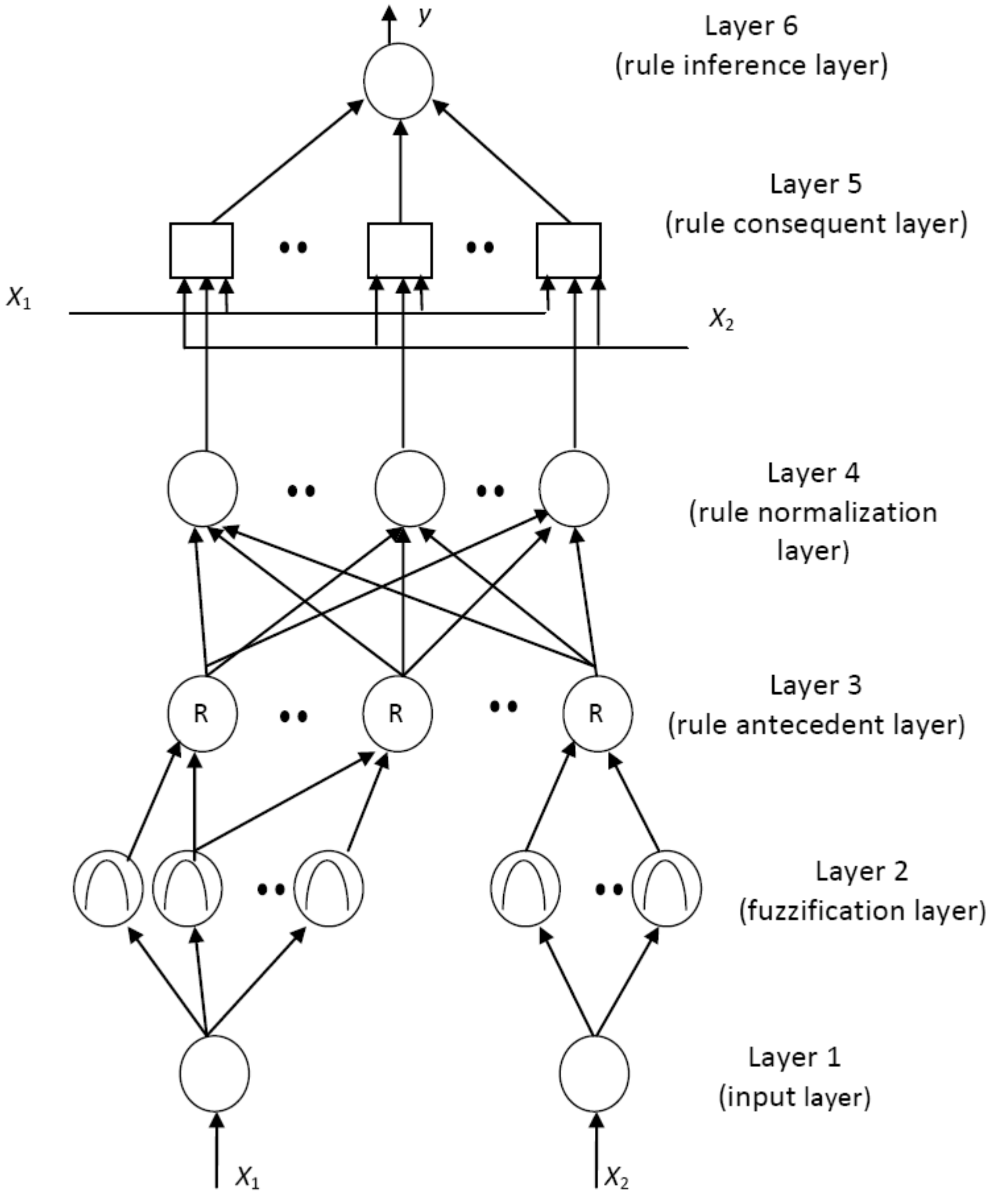


Figure 3.10 Takagi Sugeno NeuroFuzzy system [90]

As pointed out before, there are too many proposed architecture with different name in integrated NeuroFuzzy approach in the literature. Instead of demonstration of every architecture in here, interesting reader can refer to given reference as following. Adaptive NeuroFuzzy inference system (ANFIS)[22], Fuzzy adaptive learning Control (FALCON)[93], generalized approximate reasoning based intelligent control (GARIC) [94], NeuroFuzzy controller (NEFCON) [95], NeuroFuzzy classification (NEFCLASS) [96], NeuroFuzzy function approximation (NEFPROX) [97], Fuzzy Net (FUN) [98], self constructing neural fuzzy inference network (SONFIN) [99], fuzzy inference environment software with tuning (FINEST) [100], evolving fuzzy neural networks (EFuNN) [101], dynamic evolving fuzzy neural networks (dmEFuNN) [102], evolutionary and neural learning of fuzzy Inference System (EvoNF) [103].All of the above techniques and the other that we missed declare here, ANFIS is the most of the common techniques in model based control, function approximation, decision making etc, because of its simple architecture, availability in software tool as Matlab [104] .

3.3.2.3 Adaptive NeuroFuzzy Inference System (ANFIS)

ANFIS is similar with TSK type NeuroFuzzy model [22]. For simplicity, we assume the fuzzy inference system under consideration has two inputs x_1 , x_2 and one output y as depicted in Eqn. 3. 3. For first order Sugeno type output function and with two membership functions at the antecedent part, a rule base can be drawn as;

$$\boldsymbol{R_1}\ \boldsymbol{;IF}\ \ x_1\ \ \boldsymbol{is}\ \ A_1\ \ \boldsymbol{AND}\ \ x_2\ \ \boldsymbol{is}\ B_1\ \ \boldsymbol{THEN}\ \ y\ \ =\ b_1$$

$$\boldsymbol{R_2}\ \boldsymbol{;IF}\ \ x_2\ \ \boldsymbol{is}\ \ A_2\ \ \boldsymbol{AND}\ \ x_2\ \ \boldsymbol{is}\ B_2\ \ \boldsymbol{THEN}\ \ y\ \ =\ b_2 \tag{3.12}$$

Fig. 3.11 shows network representation of these two rules. The neurons on first hidden layer perform fuzzification which computes the membership degrees of input variables, the product node ∏ in the second layer represent the antecedent connective of ***R₁*** and ***R₂*** in Eqn.3.12. The normalization node N and the summation node ∑ realize the fuzzy mean operator. Typically, Gaussian function is used for membership function of the antecedent part as similarly RBFNN with Eqn. 3. 11. The output of the ANFIS model with conjunctive form antecedent is;

$$y = \sum_{i=1}^{r} \gamma_i (x) b_i \tag{3.13}$$

where r is the total number of rules .The normalized rule firing strength for i*th* rule γ_i ,is calculated as

$$\gamma_i = \frac{\prod_{j=1}^{p} \exp\left(-\left(x_j - c_{ij}\right)^2 / 2\sigma_{ij}^2\right)}{\sum_{i=1}^{K} \prod_{j=1}^{p} \exp\left(-\left(x_j - c_{ij}\right)^2 / 2\sigma_{ij}^2\right)}$$

where c_{ij} and σ_{ij} are center and spread of gaussian membership function for i*th* rule of j*th* inputs For first order ANFIS model which is shown in Fig. 3.16 Eqn. 3.14 can be expanded as;

$$y = \sum_{i=1}^{r} \gamma_i(x)\left(a_i^T x + b_i\right) \tag{3.14}$$

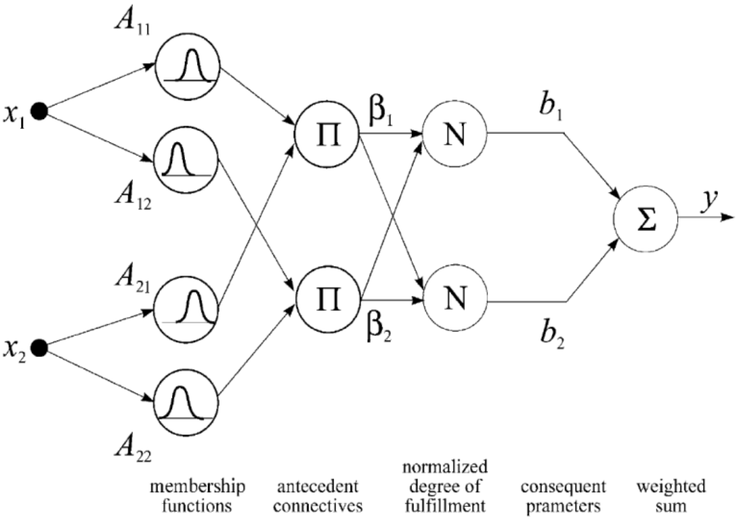


Figure 3.11 Zero order ANFIS with two rules [11].

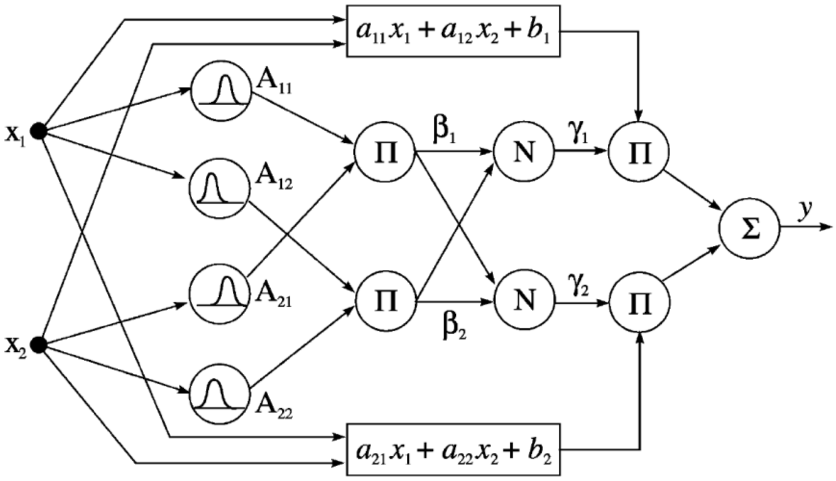


Figure 3.12 First order ANFIS with two rules [11]

3.4 Learning Algorithms

A neural network has to be configured such that the application of a set of inputs produces the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to train the neural network by feeding it teaching patterns and letting it change its weights according to some learning rule. The learning situations in neural networks may be classified into three distinct sorts [88]. These are supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, an input vector is presented at the inputs together with a set of desired responses, one for each node, at the output layer. A forward pass is done, and the errors or discrepancies between the desired and actual response for each node in the output layer are found. These are then used to determine weight changes in the net according to the prevailing learning rule. The term *supervised* originates from the fact that the desired signals on individual output nodes are provided by an external teacher.

In unsupervised learning (or self-organization), a (output) unit is trained to respond to clusters of pattern within the input. In this paradigm, the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified; rather, the system must develop its own representation of the input stimuli. Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. These two characteristics, trial-and error search and delayed reward are the two most important distinguishing features of reinforcement learning [88].

3.4.1 Training Methods for Parametric Identification

Training is the adaptation of weights, centers and radii of membership functions in a NeuroFuzzy network such that the error between the desired output and the network output is minimized. Two steps are distinguished in this procedure [88]:

(1) *Feedforward computation*. From the network inputs *xi* , the outputs of the first hidden layer are first computed. Then using these values as inputs to the second hidden layer, the outputs of this layer are computed, etc. Finally, the output of the network is obtained.

(2) *Parameter adaptation*. The output of the network is compared to the desired output. The difference of these two values, the error, is then used to adjust the weights, center of membership functions, radii of the membership function first in the output layer, then in the layer before, etc., in order to decrease the error (gradient-descent optimization). This backward computation is called error back propagation or gradient descent method [105]

A neural network can be trained in two different modes: online and batch modes. The online method weight updates are computed for each input data sample, and the weights are modified after each sample. An alternative solution is to compute the weight update for each input sample, but store these values during one pass through the training set which is called an *epoch*. At the end of the epoch, all the contributions are added, and only then the weights will be updated with the composite value. This method adapts the weights with a cumulative weight update, so it will follow the gradient more closely. It is called the *batch-training mode* [88].

Multi-layered networks are capable of performing just about any linear or nonlinear computation, and can approximate any reasonable function arbitrarily well. Such networks overcome the problems associated with the perceptron and linear networks. However, while the network being trained may be theoretically capable of performing correctly, back propagation, and its variations may not always find a solution. Picking the learning rate for a nonlinear network is a challenge. As with linear networks, a learning rate that is too large leads to unstable learning. Conversely, a learning rate that is too small results in incredibly long training times. The error surface of a nonlinear network is more complex than the error surface of a linear network. The problem is that nonlinear transfer functions in multilayer networks introduce many local minima in the error surface. As gradient descent is performed on the error surface it is possible for the network solution to become trapped in one of these local minima. This may happen depending on the initial starting conditions. Settling in a local minimum may be good or bad depending on how close the local minimum is to the global minimum and how low an error is required. In any case, be cautioned that although a multilayer back propagation network with enough neurons can implement just about any function, back propagation will not always find the correct weights for the optimum solution. Networks are also sensitive to the number of neurons in their hidden layers. Too few neurons can lead to under fitting. Too many neurons can contribute to over fitting, in which all training points are well fit, but the fitting curve takes wild oscillations between these points. [89]

In practice, there are four types of optimization algorithms that are used to optimize the parameters. The first three methods, gradient descent, conjugate gradients, and quasi- Newton, are general optimization methods whose operation can be understood in the context of minimization of a quadratic error function. Although the error surface is surely not quadratic, for differentiable node functions, it will be so in a sufficiently small neighborhood of a local minimum, and such an analysis provides information about the behavior of the training algorithm over the span of a few iterations and also as it approaches its goal [88].

The fourth method of Levenberg and Marquardt is specifically adapted to the minimization of an error function that arises from a squared error criterion of the form we are assuming. A common feature of these training algorithms is the requirement of repeated efficient calculation of gradients. [88]. Levenberg-Marquardt training is convenient for small and medium size networks, if there is enough memory available [106]

For ANFIS and RBFNN training some hybrid learning method are used with combining GD and least square estimator (LSE) to improve convergence speed. In fact, the computational complexity of the least square LSE is usually higher than GD methods for one-step adaptation. However, for achieving a desired performance level, LSE is usually much faster. Combining LSE and GD can be done by five different ways [23] as follows;

1-) Nonlinear parameters are fixed while linear parameters are identified by one-time application of LSE

2-)All parameters are updated by GD iteratively

3-)LSE is employed only once at the very beginning to obtain initial values of linear parameters and then GD takes over to update all parameters

4-)In each epoch GD used to update nonlinear parameters is followed by LSE to identify the linear parameters.

5-)The outputs of an adaptive network are linearized with respect to the its parameters and then the extended Kalman Filter algorithm is employed to update parameters.

Standard LSE methods may be faced to over-parameterize problem even if the error converge to the zero. To overcome this problem some modification on LSE as global RLE, local LSE [11] exist in literature.

3.4.2 Structural Identification Methods

A NeuroFuzzy system should be able to *learn* linguistic rules and/or membership functions, or optimize existing ones. There are three possibilities [107]: 1) the system starts without rules, and creates new rules until the learning problem is solved. Creation of a new rule is triggered by a training pattern which is not sufficiently covered by the current rule base; 2) the system starts with all rules that can be created due to the partitioning of the variables and deletes insufficient rules from the rule base based on an evaluation of their performance; 3) the system starts with a rule base with a fixed number of rules. During learning, rules are replaced by an optimization process.

Taha and Ghosh [108] have considered additional issues related to rule extraction. These include the granularity of explanation, modifiability, theory refinement capability (to handle incompleteness, inconsistency, and/or inaccuracy of initial domain knowledge), stability/robustness to corruption in data/knowledge, and scalability for large datasets/rule bases. Unfortunately, most of the available literature on rule generation does not provide such rigorous assessment on their pros and cons. There is also a preponderance of *specific purpose* techniques that are designed to work with a particular ANN architecture. This limits the scope of comparing the various techniques in a single framework [109].

The number of rules and the rule structure in a NeuroFuzzy system plays an important role both system performance and training time. With too few rules, the network may be unable to learn the relationships amongst the data and the error will fail to fall below an acceptable level. Thus, selection of rules is a crucial decision. Basically for rule extraction, which is also called structure identification, there are several methods in the literature. Some of the major of them are template based membership function [11] which is a kind of partition methods [23], clustering methods [28] and other methods that uses some data mining and artificial intelligence tool as the Genetic Algorithm(GA) [43, 110, 111].

CHAPTER 4

CLUSTERING METHODS

Developing the fuzzy system in conventional approach, the membership function and the consequent model are fixed by the designer according to the priori information if available and system is tuned via trial and error. If the priori information is not available but input output data set is available then, the structure of the fuzzy system can be obtained by using a structural of identification methods

The number of rules and the rule structure in a NeuroFuzzy system plays an important role both system performance and training time. With too few rules, the network may be unable to learn the relationships amongst the data and the error will fail to fall below an acceptable level. Thus, selection of rules is a crucial decision. Basically for rule extraction, which is also called structure identification, there are several methods in the literature. Some of the major of them are template based membership function [11] which is a kind of partition methods [23], clustering methods [28, 49, 112] and other methods that uses some data mining and artificial intelligence tool as the Genetic Algorithm(GA’s) [38, 110, 111]. Data clustering interesting approach for finding similarities in data and putting similar data into groups. Clustering partitions a data set into several groups such that the similarity within a group is larger than that among groups [113].

In this chapter, three of the most common clustering techniques; K-Means, Fuzzy C-Means and Subtractive Clustering, in structure identification of NeuroFuzzy system are described and applied to detect cervical cancer as standalone classifiers that it is the first work for comparison of clustering methods for cervical cancer detection.

4.1 K-Means Clustering

The K-means clustering, or Hard C-means clustering [114], is an algorithm based on of dissimilarity (or distance) measure is minimized [114]. In most cases this dissimilarity measure is chosen as the Euclidean distance.

A set of *n* vectors x_j, $j = 1,2,3.....n$, are to be partitioned into *c* groups G_i $i = 1,2,3......, c$. The cost function, *J*, based on the Euclidean distance between a vector x_k in group *j* and the corresponding cluster center c_i , can be defined by:

$$J = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \left(\sum_{k, x_k \in G_i} \left\| x_k - c_i \right\|^2 \right) \tag{4.1}$$

$$U = \begin{bmatrix} u_{11} & . & . & u_{1n} \\ . & . & . & . \\ . & . & . & . \\ u_{c1} & . & . & u_{cn} \end{bmatrix}_{c \times n} \tag{4.2}$$

The partitioned groups are defined by a $c \times n$ binary membership matrix U in Eqn. 4.2, where the element u_{ij} is 1 if the j *j*th data point belongs x_j to group i, and 0 otherwise. Once the cluster centers c_i are fixed, the minimizing Eqn. 4.1 for u_{ij} can be derived as follows:

$$u_{ij} = \begin{cases} 1 & \left\| x_j - c_i \right\|^2 \le \left\| x_k - c_i \right\|^2, \mathit{for} \quad k \ne i \\ 0 & \qquad \mathit{otherwise} \end{cases} \tag{4.3}$$

On the other hand, if the membership matrix is fixed, i.e. if u_{ij} is fixed, then the optimal center c_i that minimize Eqn. 4.1 is the mean of all vectors in group i *:*

$$\left| G_i \right| = \sum_{j=1}^{n} u_{ij} \tag{4.4}$$

$$c_i = \frac{1}{\left| G_i \right|} \sum_{k, x_k \in G_i} x_k \tag{4.5}$$

4.2 Fuzzy C-Means Clustering

Fuzzy C-means clustering (FCM) [113], relies on the basic idea of Hard C-means clustering (HCM) [114] , with the difference that in FCM each data point belongs to a cluster to a degree of membership grade, while in HCM every data point either belongs to a certain cluster or not. So FCM employs fuzzy partitioning such that a given data point can belong to several groups with the degree of belongingness specified by membership grades between 0 and 1. However, FCM still uses a cost function that is to be minimized while trying to partition the data set. The membership matrix U is allowed to have elements with values between 0 and 1. However, the summation of degrees of belongingness of a data point to all clusters is always equal to unity:

$$U = \begin{bmatrix} u_{11} & . & . & u_{1n} \\ . & & & . \\ . & & & . \\ u_{c1} & . & . & u_{cn} \end{bmatrix}_{c\times n} \tag{4.6}$$

$$\sum_{i=1}^{c} u_{ij} = 1 \qquad \forall j = 1,2,3....n \tag{4.7}$$

where u_{ij} is between 0 and1.

The cost function for FCM is a generalization of Eqn. 4. 1;

$$J(U, c_1, c_2, c_3......., c_c) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{m} \left\| x_j - c_i \right\|^2 \tag{4.8}$$

where; x_j is jth data vector, c_i is ith cluster center and m is weighting exponent in $[1, \propto)$

The condition for cost function to reach its minimum is;

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^{m} x_j}{\sum_{j=1}^{n} u_{ij}^{m}} \tag{4.9}$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{c}\left(\frac{\|c_i - x_j\|^2}{\|c_k - x_j\|^2}\right)^{\frac{2}{m-1}}} \tag{4.10}$$

The algorithm works iteratively through the preceding two conditions until the no more improvement is noticed.

4.3 Subtractive Clustering

The subtractive clustering method assumes each data point is a potential cluster center and calculates a measure of the likelihood that each data point would define the cluster center, based on the density of surrounding data points [49,114]. The density measurement at a data point x_i is defined as;

$$D_i = \sum_{j=1}^{n} \exp\left(-\frac{\|x_i - x_j\|^2}{\left(\frac{r_a}{2}\right)^2}\right) \tag{4.11}$$

where $r_a \in [0, \propto)$ is neighboring radius. The density value of i th data point will be larger one if it has many neighboring data points and the distance between the data points and its location is small. The first cluster center is chosen as x_{c1}which has largest density value, D_{c1}. For second cluster center, the effect of the first cluster center is subtracted in determination of the new density values, which follows:

$$D_i = D_i - D_{c1} * \exp\left(-\frac{\|x_i - x_{c1}\|^2}{\left(\frac{r_b}{2}\right)^2}\right) \tag{4.12}$$

where $r_b \in [0, \propto)$ is neighborhood that has measurable reduction in density measurement. According to Eqn.4.12, the data points which are near the first cluster center x_{c1} will reduce the measured density significantly. The data point x_{c2} which corresponding to the larger density value according to the Eqn. 12 is chosen is selected for second cluster center. The selection of next cluster centers process is carried out iteratively, until the stopping criteria achieved. For the next cluster centers, if the measured density function is greater than a certain threshold as given Eqn13, then the i th data point is selected as c_k cluster and algorithm try to find the other cluster centers. If Eqn.13 is not provided and if the measured density is less than a lower threshold as described in Eqn. 14, then the algorithm stops.

$$D_k > \varepsilon^{up} * D_1 \tag{4.13}$$

$$D_k < \varepsilon^{down} * D_1 \tag{4.14}$$

4.4 Case Study

Cervical cancer is the second most common cancer type in women, with 500,000 new cases reported each year and 250,000 deaths worldwide. Eighty percent of the deaths occur in developing countries due to the lack of widespread screening programs [115]. Although soft computing and artificial intelligent tools are found in a lot of real world problems, there aren’t adequate applications for cervical cancer detection.

4.4.1 The Pap-Smear Problem

Using a small brush, a cotton stick or wooden stick, a specimen is taken from the uterine cervix and transferred onto a thin, rectangular glass plate (slide). The specimen (smear) is stained using the Papanikolaou method [116, 117]. This makes it possible to see characteristics of cells more clearly in a microscope. The purpose of the smear screening is to diagnose pre-malignant cell changes before they progress to cancer. Smears contain mainly two types of cells: squamous epithelial cells and columnar epithelial cells in Figure 4.1.

Dysplastic cells are cells that have undergone pre-cancerous changes. They generally have larger and darker nuclei and have a tendency to cling together in large clusters. Squamous dysplasia is divided into three classes: mild, moderate and severe as shown in Fig. 4. 1. E-G. Mild dysplastic cells have enlarged and light nuclei. For moderate dysplastic cells, the nuclei are larger and darker [118, 119]. We used the data which was collected in [116, 117] and it contains 500 cells with the following distribution:

Normal - Columnar epithelial, 50 cells.

Normal - Parabasal squamous epithelial, 50 cells.

Normal - Intermediate squamous epithelial, 50 cells.

Normal - Superficial squamous epithelial, 50 cells.

Abnormal - Mild squamous non-keratinizing dysplasia, 100 cells.

Abnormal - Moderate squamous non-keratinizing dysplasia, 100 cells.

Abnormal - Severe squamous non-keratinizing dysplasia, 100 cells.

4.4.2 Clustering Results

We use the K-means, FCM and Subtractive clustering methods in order to classify the pap-smear data. The data has consist of 500 cells with 24 features which means all cell contains 24 dimension that it is impossible to show clustered data and clusters centers. The Matlab codes are written for each clustering methods. In order to measure the clustering accuracy results, we computed the RMSE and True Classification ratio.

We used the 80 % of data for training our classification methods which are based on K-means, FCM and Subtractive Clustering methods. After finding the cluster centers, we used 20 % of data for testing and computing the classification accuracy. The number of clusters into which data set is to be portioned is two; the cell which is classified as normal and the cell which is classified as abnormal. Each clustering algorithm is presented with the training data set, and as a result of two cluster centers are produced. The data in the testing set is then tested against the found cluster centers and analysis of the result is conducted for pap-smear classification task.

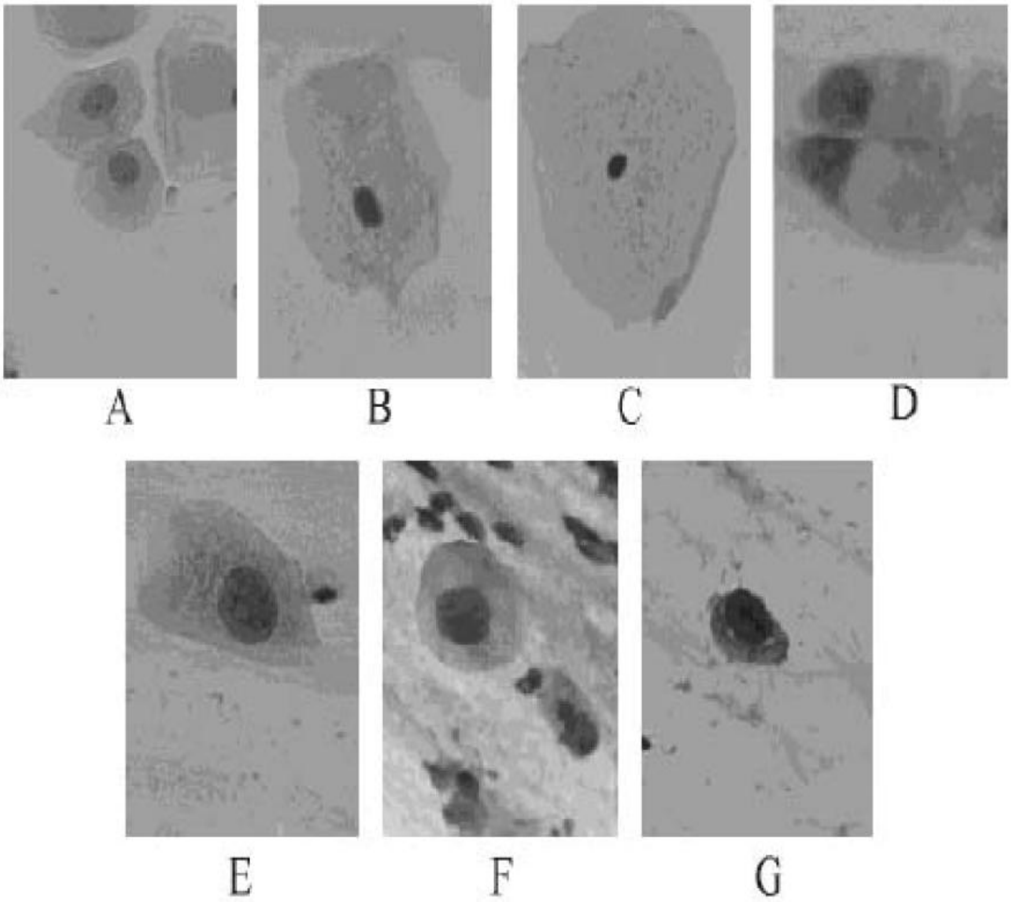


Figure 4.1 Some of the cells found in cervix: (A) parabasal, (B) intermediate, (C) superficial squamous epithelia, (D) columnar epithelium, (E-F) mild, moderate, and severe non keratinizing dysplasia [116, 117]

4.4.3 Results of K-means Clustering

For pap-smear clustering and classification with K-means, after calculation of the clustering centers, the testing vectors are assigned to their respective clusters according to the distance between each vector and each cluster centers. Error measurement is realized by RMSE and accuracy is measured as the percentage of correctly classified cells. The RMSE value is calculated as 0.424 and testing accuracy is calculated as % 82 which means 82 cells are truly classified. Number of iterations is 8 for but it depends on the initial clustering centers, for testing the second and third times we found same RMSE and accuracy value but iteration counts varied between 6 and 13. Visual result of clustering of pap-smear nucleus versus cytoplasm is plotted in Fig. 4. 2., and cost function versus iteration number in Fig. 4.3.

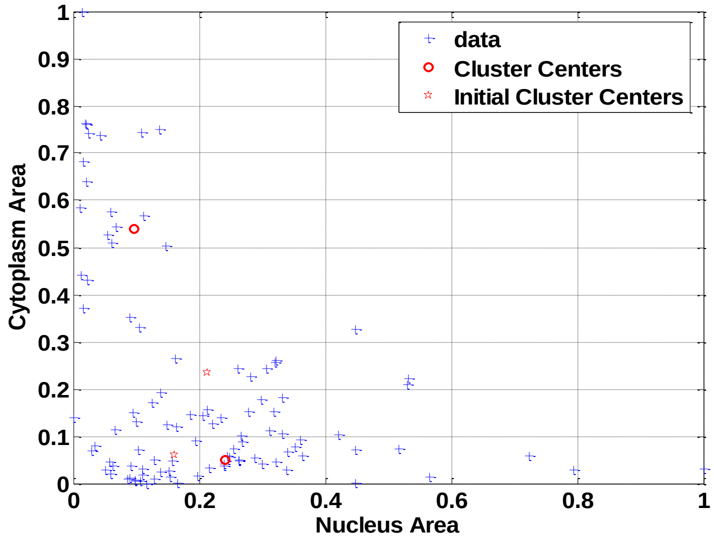


Figure 4.2 K-means Clustering result for pap-smear data with feature1 and feature2

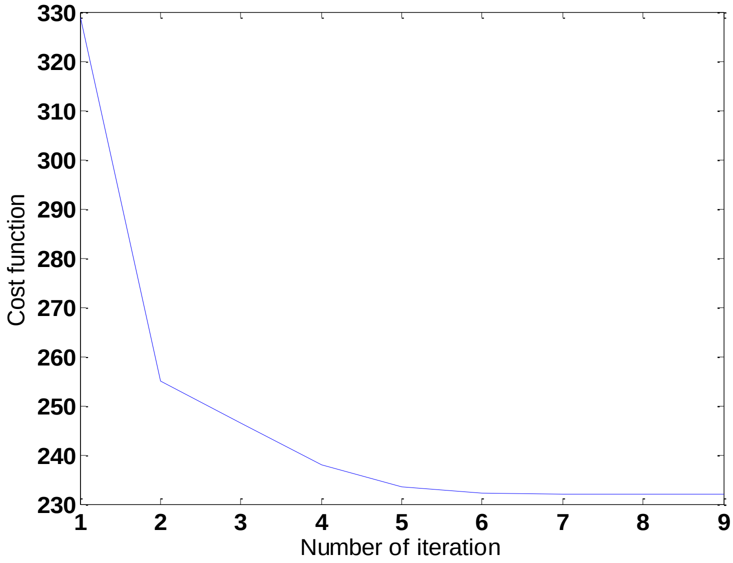
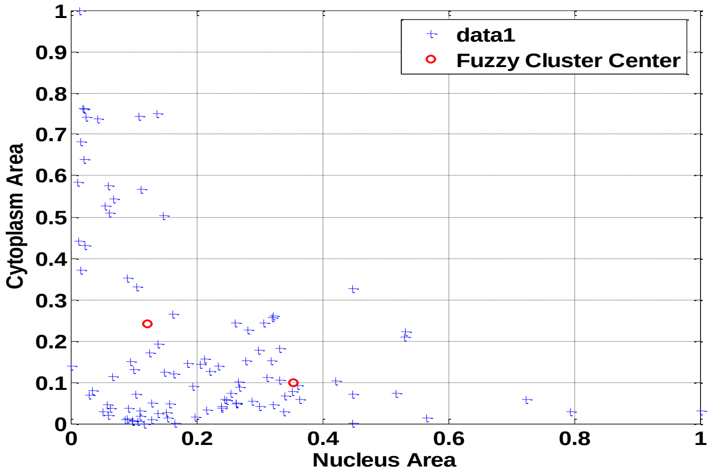


Figure 4.3 Cost function versus iteration for K-means clustering of pap-smear data

4.4.4 Results of FCM Clustering

Basic difference between FCM and K-means is membership matrix. In FCM, the membership matrix contains the membership degrees of data points to against cluster center instead of 1 or 0 as K-means. FCM algorithm firstly initializes the membership matrix then computes the clustering centers. The initial membership matrix effects the system performance with only the total iteration number, it doesn’t effect on the RMSE and classification accuracy. For assigning the each data to against cluster center, a defuzzification procedure is applied because the membership degrees of the membership matrix are fuzzy values and must be mapped to the crisp values as 0 and 1. Clustering and classification of pap-smear with FCM clustering technique are shown in Fig.4.10 are assigned to their respective clusters according to distance between each vectors and each cluster centers. The RMSE value is calculated as 0.648074 and testing accuracy is calculated as % 58 which means 58 cells are truly classified for FCM clustering. Number of iteration is 30 for current test but it varied between the value 24 and 32, based on the initial membership matrix, for testing the second and third times we found same RMSE and accuracy value. Visual result of clustering of pap-smear nucleus versus cytoplasm is plotted in Fig. 4. 4, and cost function versus iteration number in Fig. 4. 5.



.

Figure 4.4 FCM clustering result for pap-smear data with feature1 and feature2

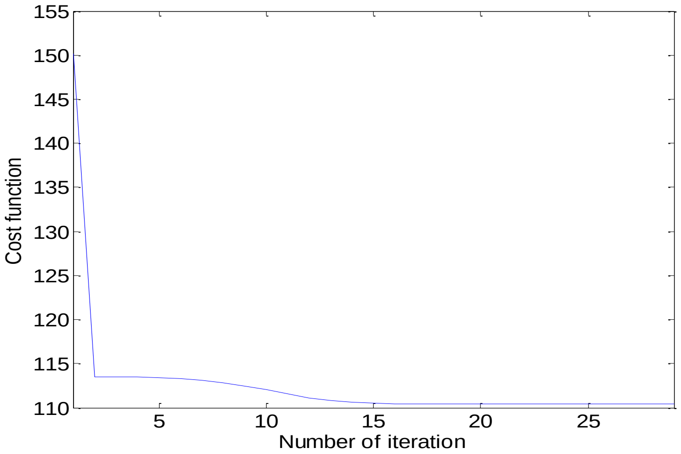


Figure 4.5 Cost function versus iteration FCM clustering of pap-smear data

According to the found results, RMSE and classification accuracy, the FCM algorithm is not success as K-means. The total computational time also bigger for FCM clustering than the K-means because of Fuzzy calculations take more time than crisp calculation, and total iteration number also bigger than the K-means’s iteration number.

4.4.5 Results of Subtractive Clustering

Subtractive clustering method tries to compute the density values of each vectors and select the cluster center against the highest density value instead of do trying to minimize a cost function. In fact basic difference of this methods from previous methods is that subtractive is an unsupervised method in which that the numbers of clusters are not necessarily given, because the algorithm tries to find cluster centers and also numbers of clusters. For given problem, that the numbers of clusters are known, as two cluster, one of them represents normal cells the other represent abnormal cell, we used algorithm as supervised algorithm that number of cluster is two.

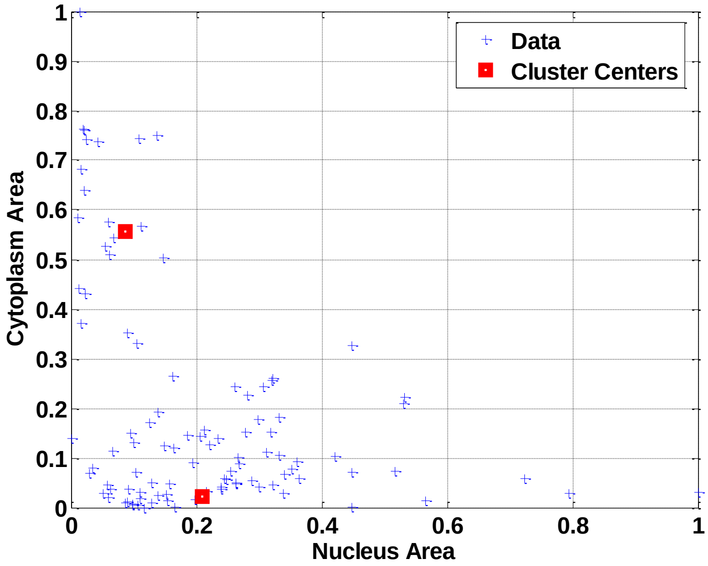


Figure 4.6 Subtractive clustering result for pap-smear data with feature1 and feature2.

Clustering and classification of pap-smear with Subtractive clustering method are shown in Fig. 4.6. The RMSE value is calculated as 0.447214 and testing accuracy is calculated as % 80 which means 80 cells are correctly classified. Compared to K-means and FCM, this result is a little behind the accuracy achieved in K-means techniques

4.5 Summary of Results

According to results for pap smear classification, best performance is achieved by K-means clustering algorithm. However FCM clustering can’t give good classification accuracy. Performance of Subtractive Clustering is almost same with K-means. Because of K-means and FCM are supervised clustering algorithms that it is needed to know how many clusters would be formed, Subtractive Clustering method which doesn’t need to know how many clusters would be formed is chosen for initial structure construction for Fuzzy and NeuroFuzzy Systems in next chapter of this thesis.

CHAPTER 5

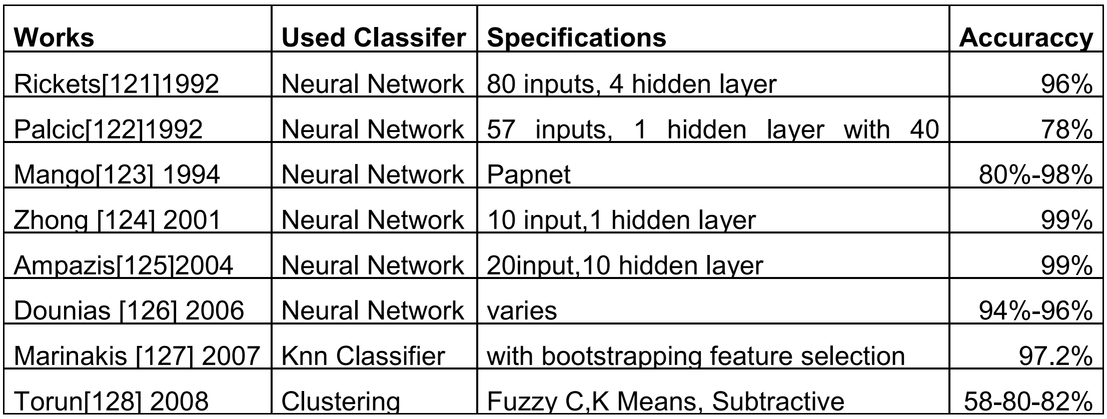
NEUROFUZZY SYSTEMS FOR CERVICAL CANCER DETECTION OR PAP SMEAR CLASSIFICATION TASK

In this chapter, the subtractive clustering method which is explained in previous Chapter, is used to in order to obtain initial structure of TS-FIS and ANFIS. Effects of the neighborhood radii of subtractive clustering on initial rule structures are analyzed with classification accuracy measurement. On the other hand, the number of neurons and numbers of layers play important role in NN for acquiring satisfactory classification results and acceptable consumption time. In fact for realization such a classifier with hardware needs minimum numbers of elements, minimum computational time, and maximum true classification. We analyzed the effects of numbers of neurons for NN and spread of neurons for RBF based pap-smear [117] classifier.

As a major human health concern, cancer has become a focus for worldwide research. The provision of more accurate diagnostic techniques might allow various cancers to be identified at an earlier stage and, hence, allow for earlier application of treatment [120]. Cervical cancer is the second most common cancer in women, with 500,000 new cases reported each year and 250,000 deaths worldwide [115]. According to the work of Ling, *et al.*, in 2008, the death rate from cervical cancer has been reduced significantly through the adoption of population-wide screening programs in developed countries [115]. Lots of generally, Table5.1 summarizes the basic valuable research on pap-smear test. It can be generalized that Neural Networks (NN) are commonly used in past decade and new research are done in current years with cooperation of other artificial tools to optimize the network for more accurate classification ratio, decreasing the computational time.

According to recent literature on bioscience as biomedical decision making, importance of feature detection has a big influence on modeling performance. For feature selection we used three algorithms. First one that we proposed a simple integrated feature selection to TS, ANFIS, NN and RBF classifier. Second that we used ranking of feature based on seperability criteria and thirdly we used principle component analyze (PCA) for reduction of feature space. The performance of classifiers and computational times are demonstrated.

Table 5.1 Previous works on pap-smear data classification for cervical cancer diagnosis



Works	Used Classifer	Specifications	Accuraccy
Rickets[121]1992	Neural Network	80 inputs, 4 hidden layer	96%
Palcic[122]1992	Neural Network	57 inputs, 1 hidden layer with 40	78%
Mango[123] 1994	Neural Network	Papnet	80%-98%
Zhong [124] 2001	Neural Network	10 input,1 hidden layer	99%
Ampazis[125]2004	Neural Network	20input,10 hidden layer	99%
Dounias [126] 2006	Neural Network	varies	94%-96%
Marinakis [127] 2007	Knn Classifier	with bootstrapping feature selection	97.2%
Torun[128] 2008	Clustering	Fuzzy C,K Means, Subtractive	58-80-82%

The data consists of 500 cells with 24 features which mean all cell contains 24 dimensions. In order to measure the clustering accuracy results, we computed the RMSE and True Classification ratio. We used the 80 % of data for training the classification methods which are based on TS FIS, ANFIS, NN and RBF NN. After finding the cluster centers, we used 20 % of data for testing and computing the classification accuracy.

5.1 Effects of Radii on TS-FIS

As described in Chapter4, the radii of neighborhoods of density function determines the number of clusters which is used for setup structure of TS-FIS [49]. The numbers are rules are directly relevant with the number of clusters which found from subtractive clustering. The obtained inference system is shown in Fig. 5.1. We seek the optimum value of radii for current classification task. As shown in Fig. 5.2, small radii causes in big size of rule base while acquiring almost 90% true classification accuracy. The optimum radii value is 0.75 which causes 7 rules with %95 true classification ratio.

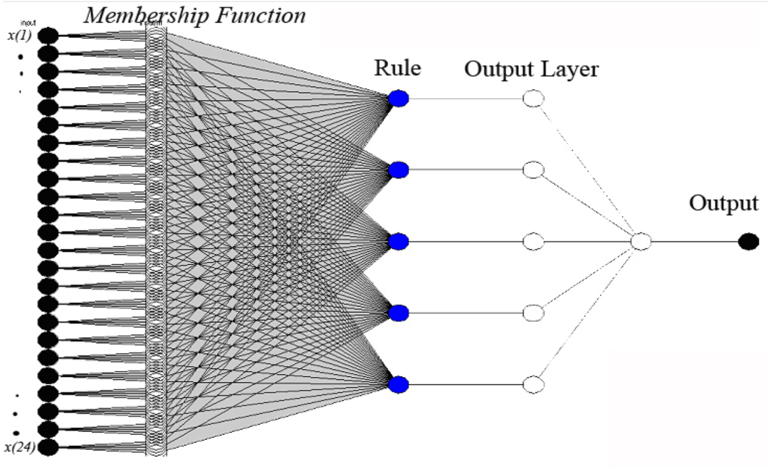


Figure 5.1 Network structure of TS-FIS and ANFIS for pap smear classification

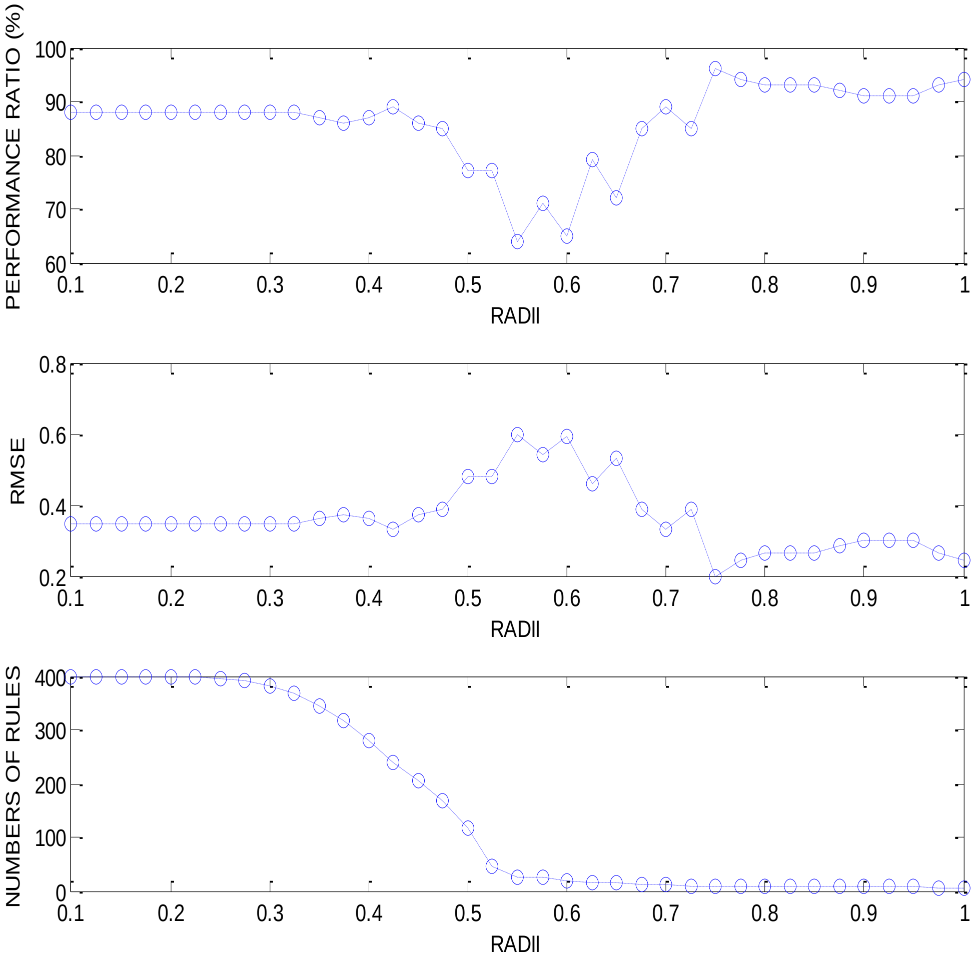


Figure 5.2 Effect of radii on TS FIS classifier.

5.2 Effects of Radii on ANFIS

ANFIS uses TS type inference system in its kernel while uses back propagation and least square estimator to calculate optimum membership function parameters [22]. So basic power of ANFIS is collection of power of fuzzy inference system with training capability of NN. According to subtractive clustering method for obtaining initial structures of ANFIS is shown in Fig. 5.1. The learning task is performed to minimization of a cost function of membership functions and output error. Output error is propagated back to tune the parameters of the membership functions via using a gradient descent method. In fact the output layer of ANFIS is linear and the linear coefficients can be optimized more rapidly by using LSE instead of back propagations methods [22]. The effects of radii on true classification ratio, RMSE and numbers of rules are shown in Fig. 5.3. The maximum classification accuracy is achieved with the radii at 0.785. The number of rules is 5 while achieving a 98 % true classification. The training is followed up after obtaining initial structure to a desired epoch number with a value 50.

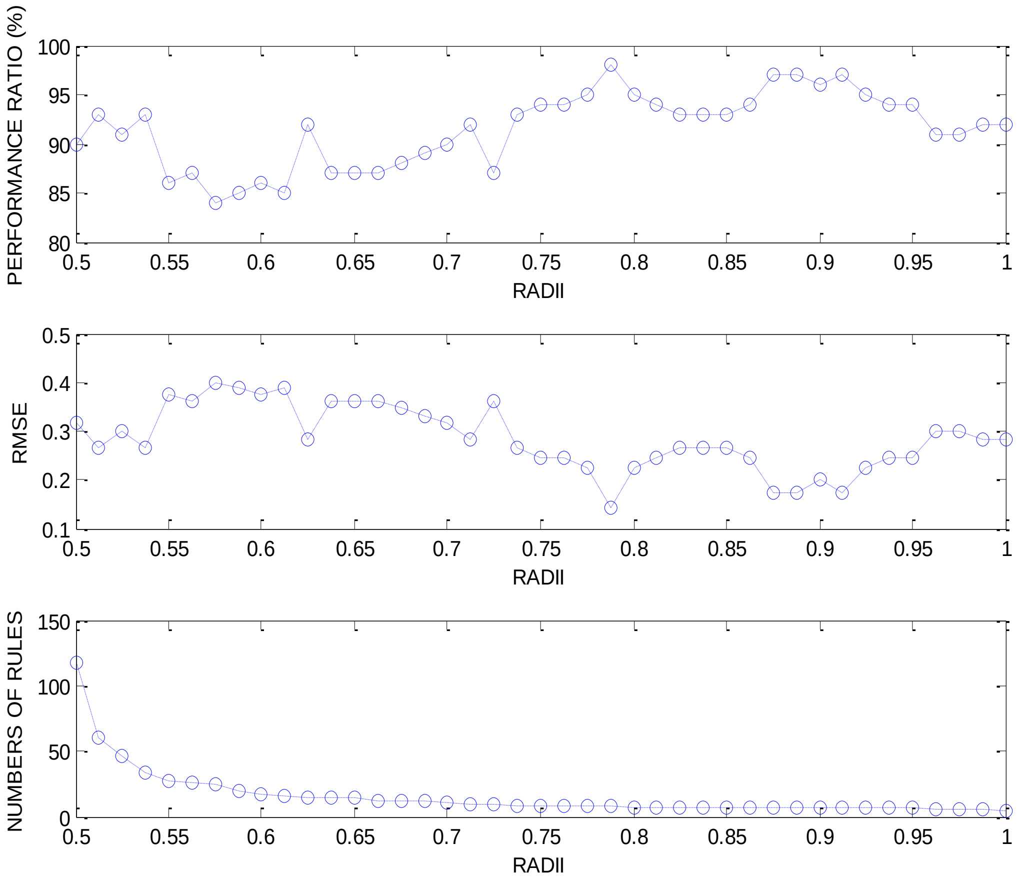


Figure 5.3 Effect of radii on ANFIS classifier.

5.3 Effects of number of neurons on Feed Forward Neural Network

We setup a network with structure with 24 input neurons, two hidden layer with 10 neurons in each and an output neuron at output layers as shown in Fig. 5.4. The learning task of NN is a nonlinear optimization problem that optimizes the weights between neurons and biases values of each neuron [89]. The learning algorithm is chosen as Levenberg-Marquardt which is a second order gradient descent based method for optimization problem. The hidden neurons are hyperbolic tangent sigmoid transfer function which can be differentiable so it is convenient for back propagation algorithms. The initial parameters are used as random values between 0 and 1 for weights and biases.

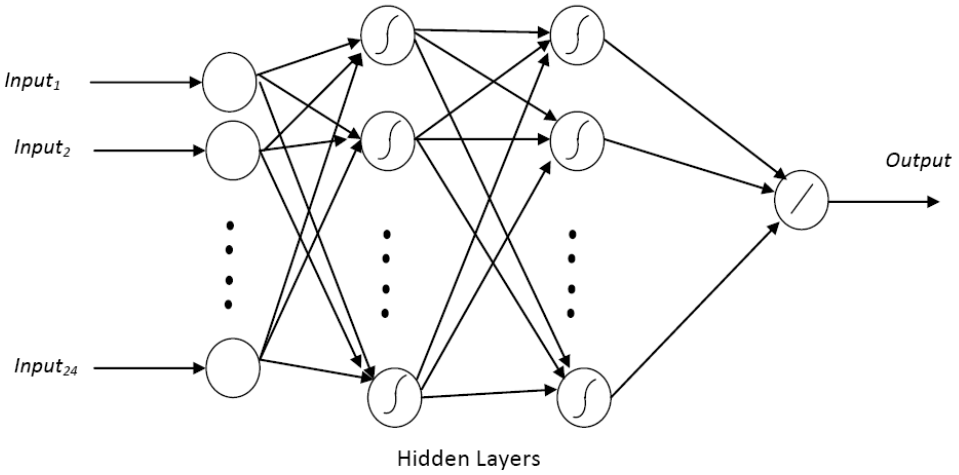


Figure 5.4 Structure of Neural Network with two hidden layer for pap smear classification

Table 5.2 Effects of number of neurons in first and second layer

		Second Layer									
	#	1	2	3	4	5	6	7	8	9	10
First Layer	1	67	86	86	84	86	86	86	86	85	89
	2	92	70	80	91	90	91	93	89	91	82
	3	64	87	87	86	90	92	92	92	91	91
	4	92	90	94	94	93	95	92	93	**96**	84
	5	91	79	93	94	95	92	91	93	93	93
	6	77	67	69	91	93	**96**	93	94	92	91
	7	92	88	91	75	90	93	95	92	92	89
	8	90	90	85	92	92	92	86	92	82	94
	9	85	93	77	93	88	93	92	95	88	94
	10	67	72	94	90	93	91	92	92	69	93

The performance of the classifier based on NN is evaluated according to the numbers of neurons. Neuron size plays important role in classification and modeling task. If the number of neurons is not adequate, then the network can’t represent the model. Otherwise if the number of neurons is too many that cause increasing the computational time and error because of gradient descent may fall in any local minima point. The performance of network is shown in Table 5.2. As shown in the table, the maximum accuracy is acquired with 6x6 and 4x9 neurons.

5.4 Effect of Spread of Neurons on Radial Basis Function Neural Network

RBF NN uses radial basis transfer function in its hidden layer. The basic difference of sigmoid transfer function which is commonly used in FNN design with radial basis function is sigmoid neurons can have outputs over a large region of the input space, while radial basis function neurons only respond to relatively small regions of the input space [88,106].

The result is that the larger the input space (in terms of number of inputs, and the ranges those inputs vary over) the more radial basis function neurons required. The spread parameter which allows the sensitivity of radial basis function neuron must be large enough that the neurons respond to overlapping regions of the input space. But too large spread cause a network whose performance isn’t good due to the large overlap of the input regions of the radial basis neurons that cause all the neurons produce an output which is close to 1, and so cannot be used to generate different responses.

The accuracy of classification oscillates while increasing the spread can be shown in Fig. 5.5. Small spread cause the underlapping of input regions of radial basis function neurons produce an output which is close to 0. That causes more neurons are needed to overlap the input regions. Although the total number of radial basis function of neurons is 350, due to the small spread value as 0.1 which is shown in Fig. 5.6, the classifier true classification ratio is almost 30% that can’t be reasonable. The maximum accuracy is acquired while spread is 0.45 as % 94 true classification ratio.

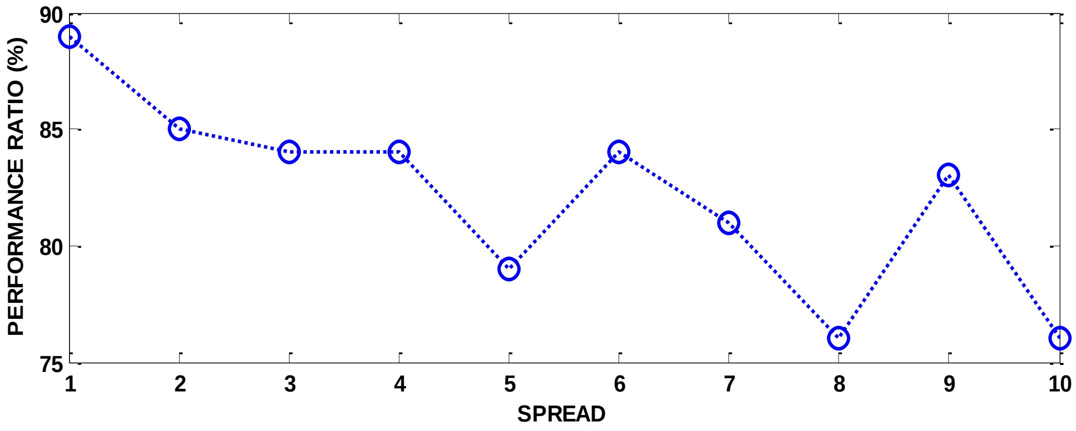


Figure 5.5 Effect of spread on radial basis function neural network performance (radial basis overlapping neurons)

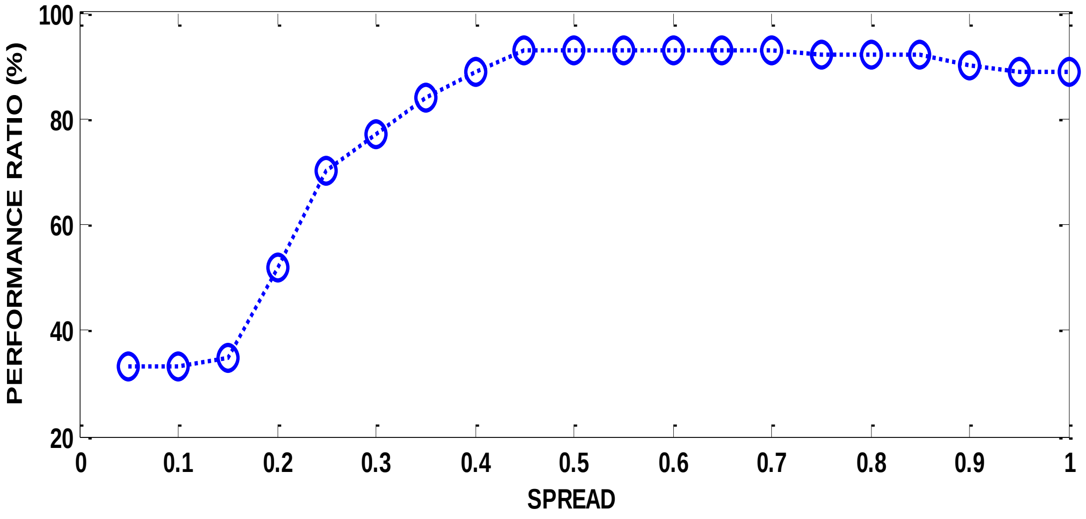


Figure 5.6 Effect of spread on radial basis function neural network performance (radial basis under lapping neurons)

5.5 Analyzing the Input Space

It is important to know the effect of each input to the output. Modeling task deals with ruling relations between inputs and outputs. But in many cases, the input dimensions that describe the object which will be modeled may be very large and include irrelevant, redundant, and noisy information. In these cases, the classifier performance gets worst while computational time increasing. Feature selection or feature reduction is a technique of selecting a subset of relevant features for building robust learning models. According to work of Abe and Kudo in 2006 [73], the main benefits of feature selection are as follows;

- Reducing computational cost and storage requirements
- Dealing with the degradation of classification efficiency due to the finiteness of training sample sets
- Reducing training and prediction time
- Facilitating data understanding and visualization

The performance of the classifier is the classification error or true classification value. Up to this section, we divide the data into two categories, first one is training data which covers the 80% of all data set and second one is test data which covers the 20% of all data. The division was performed without any criteria. Depending upon need, the classifier could over fit the training data with a strategy. Therefore, we need to access its error rate on an independent data set not used for training process to predict the performance of a classifier.

The k-fold cross validation relies on a random portioning of the data set into k parts. Then, one part is used for testing while remainder is used for training. This procedure is repeated k-1 times in order to use every part once for testing. Finally k classification estimates are averaged to yield a robust overall classification value.

The inputs features of pap-smear data set are shown in Fig. 5.7. It can be seen that some features are very similar to each other while some of them are irrelevant to outputs. Redundancy can be seen by plotting a feature to another feature. In fact for high redundancy, for sample values of two features are close the each other that means distance between samples are too small in comparing to the other features. It can be seen in Fig. 5.7 that, features,1-6, 8-9, 17-19, 16-18, 2-10, 23-24, 2-24, 18-24 and 21-22 have high redundancy as shown in Fig. 5.8.and 5.9.

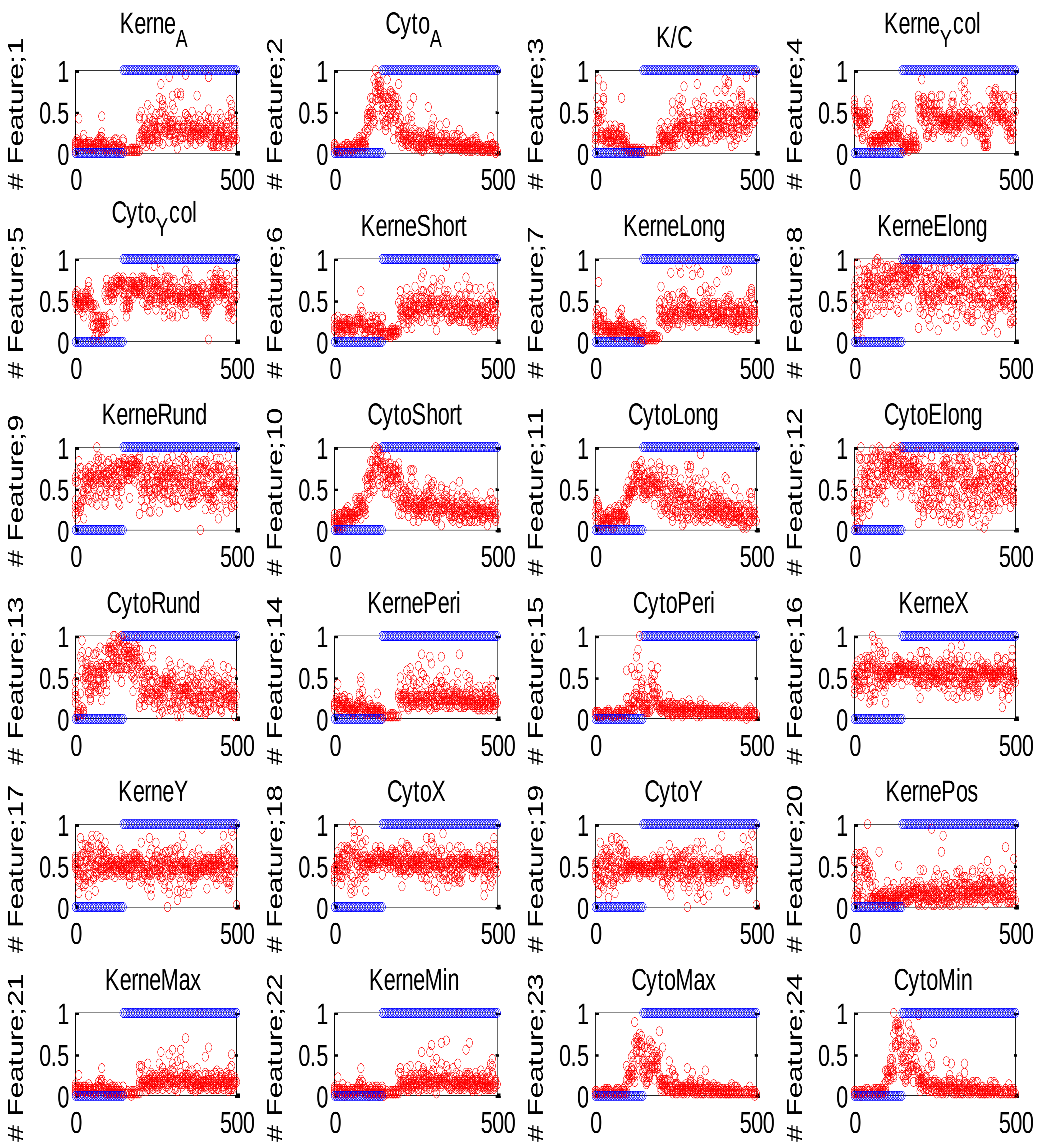


Figure 5.7 Features of pap-smear data set

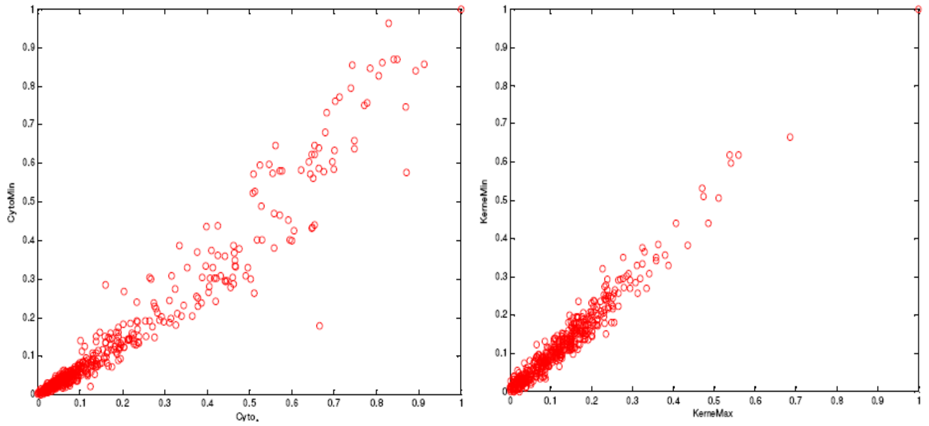


Figure 5.8 Features 2-24, 21-22

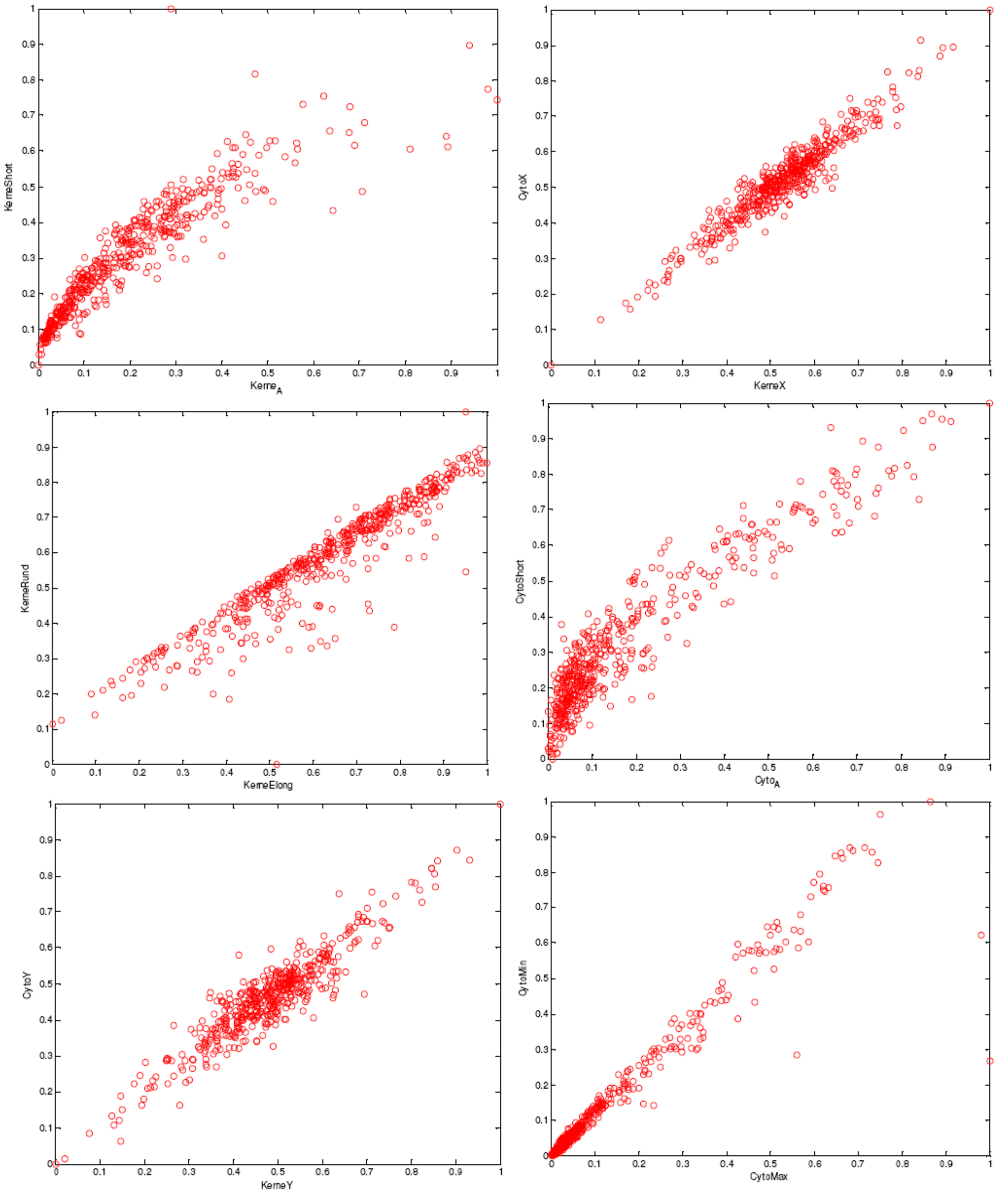


Figure 5.9 Features 2-15, 1-6, 8-9, 17-19, 16-18

5.5.1 Correlation Based Feature Selection (CBFS)

Correlation is a method for establishing the degree of probability that a linear relationship exists between two measured quantities. When there is no correlation between the two quantities, then there is no tendency for the values of one quantity to increase or decrease with the values of the second quantity. Selecting features that correlate strongest to the classification variable is known as maximum relevance selection [129]. In simplicity, if a feature has a big influence on the output, then the correlation of both will result a value close to the 1.

1st step: *set a threshold value for acceptances of feature*

↓↓

2nd step : *calculate the correlation of Feature "i" to the output*

↓

3rd step: *accept the features whose correlation coefficients greater than*

↓

4th step: *compose new input space with selected features*

↓

5th step: *evaluate the classifier(TS-FIS,ANFIS,NN,RBF)*

↓

6th step: *is performance is OK ?*

No; turn the line 1st step and decrease the threshold value

Yes; Accept the feature subset for classification task

Figure 5.10 Iterative correlation based feature selection algorithm

The above proposed algorithm in Fig. 5.10, aims to select the most powerful features in classification tasks, iteratively selects the features and in each iteration a performance criteria is compared. If the desired accuracy value is achieved then algorithm stop and optimum feature subset is selected. According to result of the algorithm the selecting features versus iteration counts are shown in Fig. 5.11. According to algorithm the initial feature space is reduced to 12 inputs instead of 24 inputs. Finally at the end of iteration only 1 feature (Feature 6) is selected for input of classifier.

After evaluating the corresponding m file for TS-FIS, ANFIS, FF NN based classifiers, the results of classification values are observed in Table5.3, Table5.4, Table5.5, respectively.

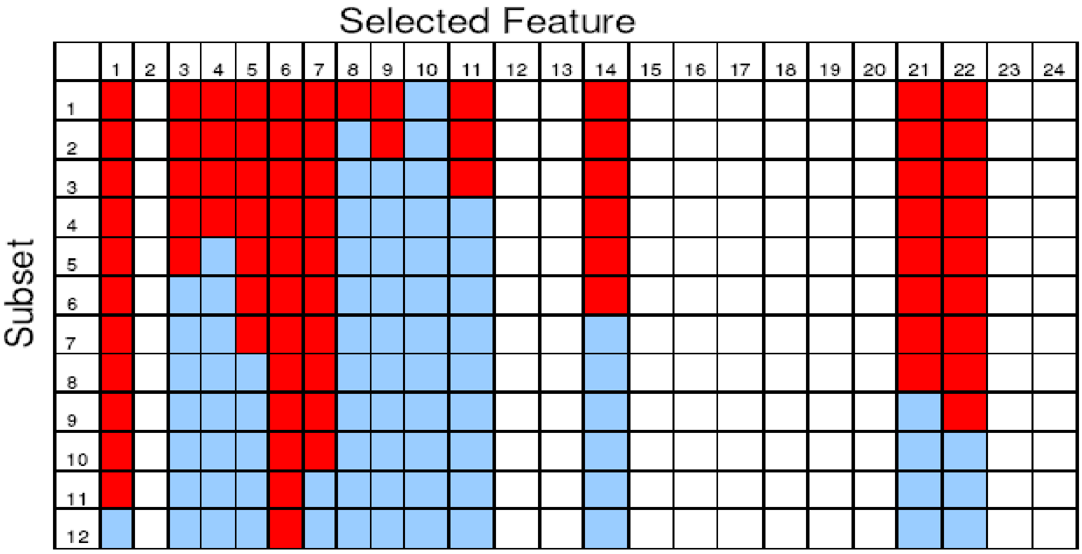


Figure 5.11 Feature selection with correlation based feature selection

In TS based classifier whose results are shown in Table5.3, the maximum accuracy value is achieved with subset 2 which contains 11 features. It can be seen that the performance of the classifier get worst while decreasing numbers of feature. But there is an improvement on the classification between subset one (F:1,3,4,5,6,7,8,9,11,14,21,22) and subset two(F:1,3,4,5,6,7,9,11,14,21,22). ANFIS based classification achieved its best performance while input space was subset1 as shown in Table 5.4. This accuracy is equal to the previous best accuracy of TS with one input more. At the end of table that, classifier is performed only one input (with feature1) and achieved as a 81% true classification ratio. While it is expected that ANFIS has a better performance, it is found that not better than TS. Another advantage of TS is computational time because there is no training phase while setup TS type FIS.

In NN based classifier whose results are shown in Table5.5, the maximum accuracy value is achieved with subset 1 which contains 12 features as like in ANFIS classifier. It can be seen that the performance of the classifier get worst while decreasing numbers of feature. It can be seen that every classifier has some 100% classification ratios for some subset. We can conclude that TS type FIS and ANFIS has better performance than NN with a difference with 0.8 % which correspond one sample is classified wrongly.

Table 5.3 The performance of TS-FIS classifier with selected feature by CBFS

Subset	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10	Means
1	98	96	90	100	94	94	92	98	94	94	**95**
2	100	92	98	98	96	100	92	94	88	100	**95.8**
3	92	98	92	96	96	100	88	96	94	92	94.4
4	96	96	94	92	96	92	92	96	92	88	93.4
5	80	86	88	80	84	94	84	98	84	96	87.4
6	82	86	94	88	76	86	88	84	96	88	86.8
7	82	78	82	84	84	84	88	84	86	88	84
8	88	88	86	90	82	76	80	82	78	88	83.8
9	88	94	74	76	80	86	88	76	80	80	82.2
10	82	84	74	76	82	84	80	76	86	80	80.4
11	90	80	84	84	72	82	72	74	66	72	77.6
12	86	68	74	74	86	78	72	74	76	78	76.6

Table 5.4 The performance of ANFIS classifier with selected feature by CBFS

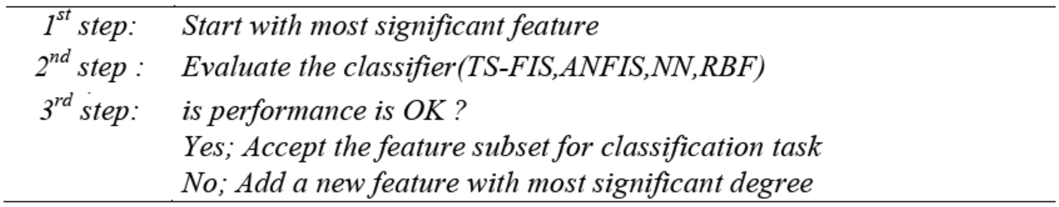
Subset	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10	Means
1	96	94	98	90	96	98	96	96	96	98	**95.8**
2	96	92	98	98	92	94	98	92	94	92	**94.6**
3	96	92	98	98	96	94	98	94	92	96	**95.4**
4	94	92	96	94	96	92	96	94	86	100	94
5	92	94	92	94	90	96	94	96	96	96	94
6	96	98	88	90	88	90	96	90	92	96	92.4
7	90	86	90	88	90	82	84	92	94	88	88.4
8	80	94	86	90	84	88	88	92	94	92	88.8
9	88	92	88	92	94	90	88	88	88	84	89.2
10	94	86	88	88	90	98	84	82	84	92	88.6
11	94	80	76	90	88	90	84	88	90	84	86.4
12	84	78	80	76	78	90	82	80	82	86	81.6

Table 5.5 The performance of NN classifier with selected feature by CBFS

Subset	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10	Means
1	94	96	94	92	94	98	96	90	94	98	**94,6**
2	92	94	92	90	90	94	96	94	94	92	92,8
3	96	90	92	80	94	98	92	96	88	92	91,8
4	96	88	90	92	90	88	96	92	98	82	91,2
5	90	92	92	100	92	90	90	92	86	94	91,8
6	92	92	94	96	92	96	86	94	92	92	92,6
7	84	94	82	86	82	88	90	84	82	94	86,6
8	84	82	82	86	84	86	90	88	90	84	85,6
9	84	86	82	82	88	88	86	84	86	88	85,4
10	90	94	86	86	90	86	88	92	94	72	87,8
11	88	82	86	74	82	78	88	84	82	90	83,4
12	82	72	88	74	84	84	72	84	88	78	80,6

5.5.2 Input Space Reduction with Feature Ranking

Feature ranking is a function of bioinformatics tool box function of Matlab [130]. The algorithm ranks the each future using an independent evaluation criterion for binary classification [130, 131]. The *rankfeature* function handles *both CCweighting* and *Nweighting*. *CCweighting* uses correlation information to outweigh an output of potential features, according to the average of the absolute values of the cross-correlation coefficient between the candidate feature and all previously selected features [130]. *Nweighting* uses regional information to outweigh the output of the algorithm of potential features the distance between the candidate feature and previously selected features. The algorithm shown in Fig5.12 is used the select most significant features for classification task. Initially, algorithm select most significant feature and perform classification. As shown in Fig. 5.13, the classification task start with only feature 19 whose correlation and regional values is the highest. Totally 24 subset is produced by 24 iteration.



1st step:	*Start with most significant feature*
2nd step :	*Evaluate the classifier(TS-FIS,ANFIS,NN,RBF)*
3rd step:	*is performance is OK ?* *Yes; Accept the feature subset for classification task* *No; Add a new feature with most significant degree*

Figure 5.12 Iterative feature ranking feature selection algorithm

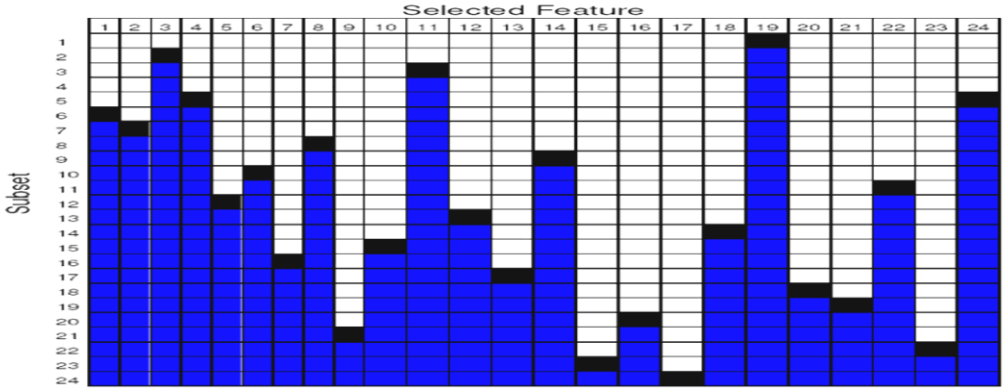


Figure 5.13 Feature selection with ranking

Table 5.6 shows the results of true classifications ratio means for 10-fold cross validation using TSFIS, ANFIS, FF NN and RBF NN classifier. According to the result for subset 1, the results are worst than the results of section 3.3.1 for one input one output classifier for all four classifier. According to results, the best accuracy achieved with 14 features with TS FIS classifier whose performance is a bit lower than in section 5.5.1. The used feature number is 14 while the maximum accuracy achieved with 11 features in counterpart in section 5.5.1. The best accurate result is 96% for ANFIS with feature ranking algorithm with 18 features while it was 95.8% with correlation based feature selection with 12 features. The performance of NN is a little worst with ranking feature than according to feature selection with correlation based. RBF NN with 9 features and more features has a better performance than NN with all combinations of subset.

Table 5.6 The performance of classifiers with selected feature by feature ranking

Subset	TS FIS	ANFIS	FFNN	RBFNN	Used Feature
1	69.6	69.6	68	68.8	*[19]*
2	69.8	68.8	75.6	74.6	*[19;3]*
3	78.2	78	82	79.8	*[19;3;11]*
4	81.6	80.8	84	80.6	*[19;3;11;24]*
5	85	86	87	85.4	*[19;3;11;24;4]*
6	86	86	90.4	91.2	*[19;3;11;24;4;1]*
7	87.4	86.8	85.6	91	*[19;3;11;24;4;1;2]*
8	87.6	87.2	88.8	90.8	*[19;3;11;24;4;1;2;8]*
9	91.8	89.8	89.4	91.8	*[19;3;11;24;4;1;2;8;14]*
10	93.4	92	89	93.8	*[19;3;11;24;4;1;2;8;14;6]*
11	94.8	93.8	91.4	94	*[19 3 11 24 4 1 2 8 14 6 22]*
12	94.4	93.4	88.4	94.4	*[19 3 11 24 4 1 2 8 14 6 22 5]*
13	94.4	93	90.6	94.4	*[19 3 11 24 4 1 2 8 14 6 22 5 12]*
14	**95.2**	94	91.4	94	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18]*
15	93.8	95	88	92.2	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10]*
16	94.6	95.2	89.6	93.8	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7]*
17	94	94.8	89.4	**95.4**	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7 13]*
18	94.2	**96**	90.6	94	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7 13 20]*
19	**95.2**	95	91.6	93.4	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7 13 20 21]*
20	93.4	95.4	91	92.4	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7 13 20 21 16]*
21	94.6	93.8	91.6	93.4	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7 13 20 21 16 9]*
22	92.8	94.6	89.4	93.4	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7 13 20 21 16 9 23]*
23	92.6	94.4	**91.8**	91.8	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7 13 20 21 16 9 23 15]*
24	93.4	93.6	90.6	93	*[19 3 11 24 4 1 2 8 14 6 22 5 12 18 10 7 13 20 21 16 9 23 15 17]*

5.5.3 Input Space Reduction with Principle Component Analysis

One of the effective procedures for reducing size of the input vector this is principal component analysis(PCA) which provides a roadmap for how to reduce a complex data set to a lower dimension[132]. This technique has three effects: it orthogonalize the components of the input vectors (so that they are uncorrelated with each other), it orders the resulting orthogonal components (principal components) so that those with the largest variation come first, and it eliminates those components that contribute the least to the variation in the data set [130]. Algorithm of iterative PCA based feature reduction is given in Fig5.14.

1st step

- *compose new input space with PCA*
- *Subtract the mean for each dimension*
- *Use the singular value decomposition to compute the principal components*
- *Compute the variance of each principal component*
- *Compute total variance and fractional variance*
- *Find the components which contribute more than min_frac of the total variance*
- *project the original data set*

2nd step evaluate the classifier(TS-FIS,ANFIS,NN,RBF)

3th step

- *is performance is OK ?*
- *No; turn the line 1 and decrease the fractional constant*
- *Yes; Accept the feature subset for classification task*

Figure 5.14 Iterative PCA based feature reduction algorithm

The performance of TS FIS, ANFIS and FF NN classifiers are shown in Table5.7, Table 5.8 and Table 5.9 respectively. Unfortunately, we couldn’t perform the RBF classifier with PCA with software that there was a big overlapping output with our experimental code.

Table 5.7 The performance of TSK FIS classifier with reduced with input space by PCA

Size	fold1	fold2	fold3	fold4	fold5	fold6	fold7	fold8	fold9	fold10	Means
22	96	92	94	92	98	98	88	90	92	92	**93.2**
18	90	96	86	94	96	96	98	86	92	94	92.8
16	94	96	94	94	94	90	86	94	90	92	92.4
15	90	96	90	92	88	96	88	94	90	90	91.4
13	94	82	90	92	92	92	88	84	90	94	89.8
12	92	86	96	88	88	100	92	94	88	90	91.4
11	86	82	92	88	90	94	88	82	96	94	89.2
10	90	90	80	82	90	90	94	94	86	86	88.2
9	90	84	78	88	84	86	82	78	92	86	84.8
8	80	82	78	78	76	98	84	86	78	80	82
7	82	88	82	82	84	86	92	72	82	84	83.4
6	86	86	78	90	76	86	78	86	74	90	83
5	84	84	76	64	82	82	90	82	76	86	80.6

Table 5.8 The performance of ANFIS classifier with reduced with input space by PCA

Size	fold1	fold2	fold3	fold4	fold5	fold6	fold7	fold8	fold9	fold10	Means
22	88	92	88	92	94	88	84	94	96	90	90.6
18	96	90	96	90	98	98	98	92	90	96	**94.4**
16	90	92	98	98	96	94	94	90	90	88	93
15	90	84	94	90	96	92	90	92	94	92	91.4
13	92	94	96	96	92	86	94	92	92	90	92.4
12	90	94	94	94	90	92	90	92	90	94	92
11	94	94	86	86	94	92	98	92	96	90	92.2
10	76	90	86	86	92	94	88	84	94	88	87.8
9	86	94	86	88	88	92	86	80	82	84	86.6
8	80	84	90	88	86	86	86	82	82	84	84.8
7	90	80	86	88	80	86	82	86	84	84	84.6
6	94	86	90	86	82	80	80	80	86	84	84.8
5	86	88	86	88	82	96	86	80	84	88	86.4

There is a relation between size of input space and classification accuracy with TS based classifier that maximum accuracy is achieved with the maximum size of the input space. The result is a bit behind previous TS based classifier . The classification performance of ANFIS with PCA based feature reduction is almost similar with previous feature selection based classifier for it. It is noted that the maximum accuracy is taken while projected new input size is 18. The NN classifier's performance with new input space produced by PCA is not as good as other two classifiers. The maximum performance is achieved with 13 input in the classification with NN. It can be seen that, the size of the projected new input hasn't big effects on the classification accuracy that it varies between 83% and 92%, for NN classifier.

Table 5.9 The performance of NN classifier with reduced input space by PCA

size	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10	Mean
22	80	90	96	88	92	76	82	88	84	78	85.4
18	86	84	90	86	88	78	94	86	80	88	86
16	90	86	88	90	86	80	96	82	84	84	86.6
15	84	80	80	90	84	96	86	92	90	92	87.4
13	90	84	92	90	84	86	92	88	92	94	**89.2**
12	90	86	90	84	92	88	92	86	86	84	87.8
11	90	92	84	92	86	92	82	94	90	80	88.2
10	90	92	96	96	76	86	76	90	78	80	86
9	94	82	86	78	86	88	84	76	94	94	86.2
8	96	84	80	92	88	82	76	84	84	92	85.8
7	76	90	88	90	88	94	96	92	86	84	88.4
6	94	90	94	88	90	86	84	84	84	90	88.4
5	74	86	84	84	82	76	82	88	86	90	83.2

5.6 Summary of Results and Discussion

The TSK FIS and ANFIS have similar performance with current classification task and their accuracy is also greater than the other two classifiers for all experiments. It can be said that the accuracy of ANFIS is a little bigger than the TS FIS and TSFIS has reached its maximum accuracy by using more feature than ANFIS. Although, the size of the input effects on both accuracy and computational time, the TS can performed with small time according to same inputs for ANFIS due to the there is no training phase for TS FIS. It can be said that RBF can handle the current problem with more accurate according to the NN based classifier. Because of both structures and training methods of these two classifier are different, the computational time and classification accuracy have different values.

Table 5.10 summarizes the classifier accuracy versus number of used features. It is clear that the best results are taken with correlation based feature selection algorithms for three classifiers. The algorithm doesn't only support accurate results but also decrease the computational time by using minimum input features.

Table 5.10 The best results obtained from four classifiers with three different feature reduction techniques

Classifier	**Correlation Based** **# features-Accuracy**	**Feature Rank Based; # features-Accuracy**	**PCA based ; size of input - Accuracy**
TSK	12---95.8%	14---95.2%	22---93.2%
ANFIS	11---95.8%	18---96%	18---94.4 %
NN	11---94,6%	23---91.8%	13---89.2 %
RBF	*	17---95.4%	*

The computational times versus used features of accuracy of the classifiers are shown in Fig. 5.15. It can be seen that RBF and TS has a similar performance characteristic with a criteria of computational time. Especially the size input hasn't big influence on computational time of both of the classifiers. On the other hand, for NN and ANFIS classifier, it can be seen that the computational time is increased rapidly while size of input increased. The computational time for 24 inputs classifiers are measured in second as; t_{TSKFIS}=0.995 sec**,** t_{ANFIS}=186.399 sec**,** $t_{FF\ NN}$=212,258sec and $t_{RBF\ NN}$ =19.658 sec**.**

The effects of radii, which determines the numbers of clusters and their coordinates of the centers that it is directly changes the structure of Fuzzy system as rule number and membership functions of each inputs, is analyzed and obtained the optimum numbers of rules for each classifier. We didn't use cross validation on data for dividing into training and test, therefore we achieved classification accuracy as 98% (2 cells are classified wrongly) for ANFIS and 95% (5 cells are classified wrongly) for TSK.

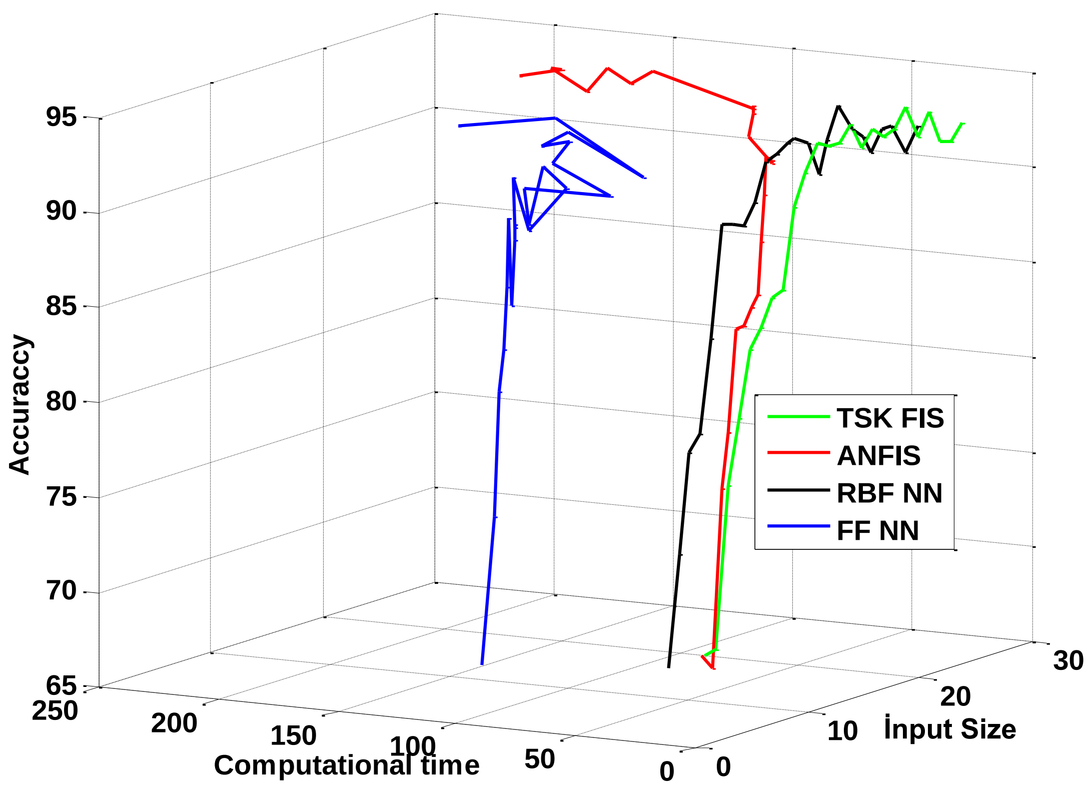


Figure 5.15 The performance of the classifiers according to size of the input versus computational time

The number of hidden layers and neurons in each layer characterize the network dynamic. We tried to find the optimum numbers for neurons by ‘trial and error’ for FF NN. It is found that the network structure which is composed of 36 neurons with combinations 6x6 and 4x9 has the best classifications accuracy as %96. We also analyzed the spread of neurons in RBF to find optimum spread value to represent the input space completely. According to the results the spread value .45 cause the best classifier accuracy

We tried to reduce feature space by three methods; correlation based feature selection, feature selection by ranking and feature reduction with PCA. The proposed simple integrated correlation based for future selection yields more acquired results in comparing to the other feature selection algorithms. Although we couldn’t improve the accuracy of classifier according to the without any feature selection based classifier, the computational time is decreased. Our results for pap-smear classification for cervical cancer diagnosis are a bit front of some works while a bit behind of others recent results in literature with a value %1-%4 .

CHAPTER 6

SIMULATED ANNEALING OPTIMISATION FOR FUZZY CLASSIFIER

In this chapter, it is proposed a new systematic algorithm which achieves not only the optimization of the parameters of fuzzy classifier and its architecture but also feature selection tasks. The proposed algorithm, namely the Simulated Annealing (SA) and Subtractive Clustering (SC) based Fuzzy Classifier (SASCFC) is a cooperation of the SA optimization algorithm and the SC method. The optimization of fuzzy classifier task is performed by optimizing radii parameter of the SC, output threshold value, and input feature subset. In order to demonstrate the effects of these optimizations, it is studied four different SASCFS models namely SASCFC-Type1, Type2, Type3, and Type4. In the former one, the SC radius which determines the number and the center location of clusters which are transformed to input membership function, and rule base are optimized. In Type2, the output threshold value, which states the mapping process of fuzzy output to output classes, and the radii of the SC are optimized. A wrapper type feature selection approach in order to obtain the most proper inputs in addition to the optimization of the radii of SC and the output threshold value is developed in Type3. A hybrid feature selection method combining a simple filter and the SA based wrapper approach is proposed in Type4 which also handles with the optimization of the radii of the SC and the output threshold value. Classification accuracies and execution time of the four proposed classifiers are compared with each other on some well known classification tasks. In these classifiers, the Type2 has the best performance that compared by four best accuracies that achieved within seven data sets in testing phase. The results show that our proposed classifiers have satisfactory performance in comparisons to its counterparts.

6.1 Background of Simulated Annealing

Simulated annealing (SA) is an iteratively search algorithm for solving combinatorial problems. Annealing is the process of heating a solid to a high temperature with subsequent cooling. The cooling process continues up to the solid reaches a state of minimum energy. This process allows obtaining good crystallization in structure of the solid. The SA mimics the physical process of annealing. Although Metropolis proposed the SA in 1953 [66], there wasn't any attention up to the work of Kirkpatrick [67] that brought into open the similarities between some optimization problem and physical process of annealing. The search algorithm for finding optimum solution in SA emulates finding good crystallization in annealing process. Perfect crystallization corresponds to the finding global optima while, poor crystallization corresponds to local optima for a combinatory optimization problem. A combinatorial optimization problem can be defined as the relation between a finite configuration set *S* and *f(S)* the cost associated with each configuration of *S*. The optimization process aims to find the *S* corresponding lowest *f(S)* with searching configuration space. The SA algorithm starts its search by taking an initial configuration, S_i and computes the cost, $f(S_i)$ at initial temperature, T_{init}. Temperature is the controlling parameter of simulated annealing algorithm's acceptance mechanism.

At initial temperature, the SA algorithm generates new combinations of configuration, S_j, and calculates correspondent costs, $f(S_j)$. A candidate configuration is accepted as new solution if the cost of it is lower than that of the current cost as;

$$S_i = S_j \quad if \quad f(S_j) \le f(S_i) \tag{6.1}$$

In case of candidate configuration yields worst cost value, simulated annealing checks an acceptance probability (Metropolis criteria) of a candidate's cost as ;

$$S_i = S_j \quad if \quad f(S_j) > f(S_i) \quad and \quad \exp\left(\frac{f(S_i) - f(S_j)}{kT}\right) > \; rand[0 \quad 1] \tag{6.2}$$

The acceptance of worst configuration with higher cost enables to algorithm to escape local minima. After searching *N* times for configuration in a temperature, the temperature, T is cooled with a cooling schedule and the algorithm searches new configuration sets at new temperature up to the temperature reaches predefined value, T_{final}

Two main features of the SA process are (1) the transition mechanism between states and (2) the cooling schedule [67]. The cooling schedule affects the performance of the optimizer that low cooling can guarantee to find global optima while rapid cooling can cause to trap to local optima. Another approach to SA is the Hide and Seek S.A which was proposed by Romeijn [133]. Hide-and-Seek SA can handle continuous variables, enabling it to make feasible solutions within the constrained or bounded ranges converge to the optimal solution [72]. Conventional SA algorithm searches the solution space in neighboring region which can be obtained adding or subtracting some quantity to the current solution. New solution for *p* dimensional parameter optimization problem with continuous configuration $S_j = [S_{j1}, S_{j2},,,,,,,,,,, S_{jp}]$ can be obtained from *k*'th iteration as;

$$S_j^{k+1} = S_j^k + \Delta S_j^k \tag{6.3}$$

where the change of the solution, ΔS_j^k, is calculated by the multi dimensional Cauchy probability distribution [68]. Hide and Seek SA searches the new configuration in all feasible space instead of the current neighborhood.

$$S_j^{k+1} = S_{jn}^k + u_j\left(ub_j - S_j^k\right) \quad if \quad \operatorname{sgn}(u_j - 0.5) < 0 \tag{6.4.a}$$

$$S_j^{k+1} = S_j^k - u_j\left(S_j^k - lb_j\right) \quad if \quad \operatorname{sgn}(u_j - 0.5) \geq 0 \tag{6.4.b}$$

where u_j is a random variable uniformly distributed between zero and one, ub_j is the upper and lb_j is the lower boundary value of feasible configuration space.

Another major difference is annealing process that conventional S.A uses constant annealing while hide and seek uses adaptive annealing process which decreases the total time with converging global [72]. In this work we use hybrid simulated annealing algorithm which uses both Hide and Seek and conventional SA searching algorithms. In order to guarantee in finding global optima , our modified algorithm searches the new configuration in all feasible regions with a transition function, which is shown in Eqn. 6.5., in high temperature value in which acceptance probability is high with taking a worst solution in order to escape global minima. In low temperature value algorithm searches a broader range according to high temperature with searching neighborhood of current states, as described in Eqn. 6.3, in order to find exact solution.

$$S_j^{k+1} = S_{jn}^k + u_j\left(ub_j - S_j^k\right)T \quad if \qquad \mathrm{sgn}(u_j - 0.5) < 0 \tag{6.5.a}$$

$$S_j^{k+1} = S_j^k - u_j\left(S_j^k - lb_j\right)T \quad if \qquad \mathrm{sgn}(u_j - 0.5) \geq 0 \tag{6.5.b}$$

6.2 Simulated Annealing Subtractive Clustering Based Fuzzy Classifier

Designing an optimal fuzzy classifier is a multivariable optimization problem. Under the lights of recent literature, construction process of a fuzzy classifier can be mainly achieved by performing following steps;

i. Selection optimum input feature subspace via removing redundant and noisy feature from input feature space

ii. Obtaining the number of the membership functions for each input.

iii. Obtaining the parameters of the membership functions

iv. Constructing rule base

v. Adjustment of the output threshold value which describes the boundaries of each class.

In order to construct an optimal fuzzy based classifier, the parameters mentioned above must be chosen properly for obtaining good classification performance. We have developed four different SA and SC based fuzzy classifiers (SASCFC) to overcome construction of proper fuzzy classifier system.

As shown in Fig. 6. 1, the SASCFC aims to optimize fuzzy classifier by optimizing r_a in the SC input subspace and output thresholds. Objective function is the classification accuracy while parameters are subtractive clustering radius, r_a, output threshold value, th, and input feature subspace, $F = \left[f_{1,} f_2 ....f_n \right]$ where n is the number of attributes. Table 6. 1 shows which parameters are optimized and which steps are achieved by the SASCFC algorithms, where $S^*$is the optimized parameters set for the configuration *S*.

Table 6.1 Optimization space and corresponding construction steps for the proposed classifiers

Classifier	*Optimization Space*	Construction steps
SASCFC Type1	$S^* = (r_a)$	*ii-iii-iv*
SASCFC Type2	$S^* = (r_a, th)$	*ii-iii-iv-v*
SASCFC Type3	$S^* = (r_a, th, F)$	*i-ii-iii-iv-v*
SASCFC Type4	$S^* = (r_a, th, F')$	*i-ii-iii-iv-v*

The SASCFC algorithm searches for global maxima as maximum classification accuracy in classification problem as;

$$f\left(S^*\right) >> f\left(S\right) \tag{6.5}$$

The algorithm starts with an initial solution at initial temperature. At next step, new parameters are generated as explained in the previous section. The classifier is trained and tested with generated parameters. If the test accuracy for new parameters is higher than the previous one, or acceptance probability is higher than a random then the new parameters are chosen as current parameters, otherwise the new parameters are refused. After acceptance for the first iteration, the algorithm searches for new parameters up to reaching a predefined maximum iteration count. When the maximum iteration number is reached, temperature is cooled with cooling mechanism in order to decrease the effect of acceptance probability.

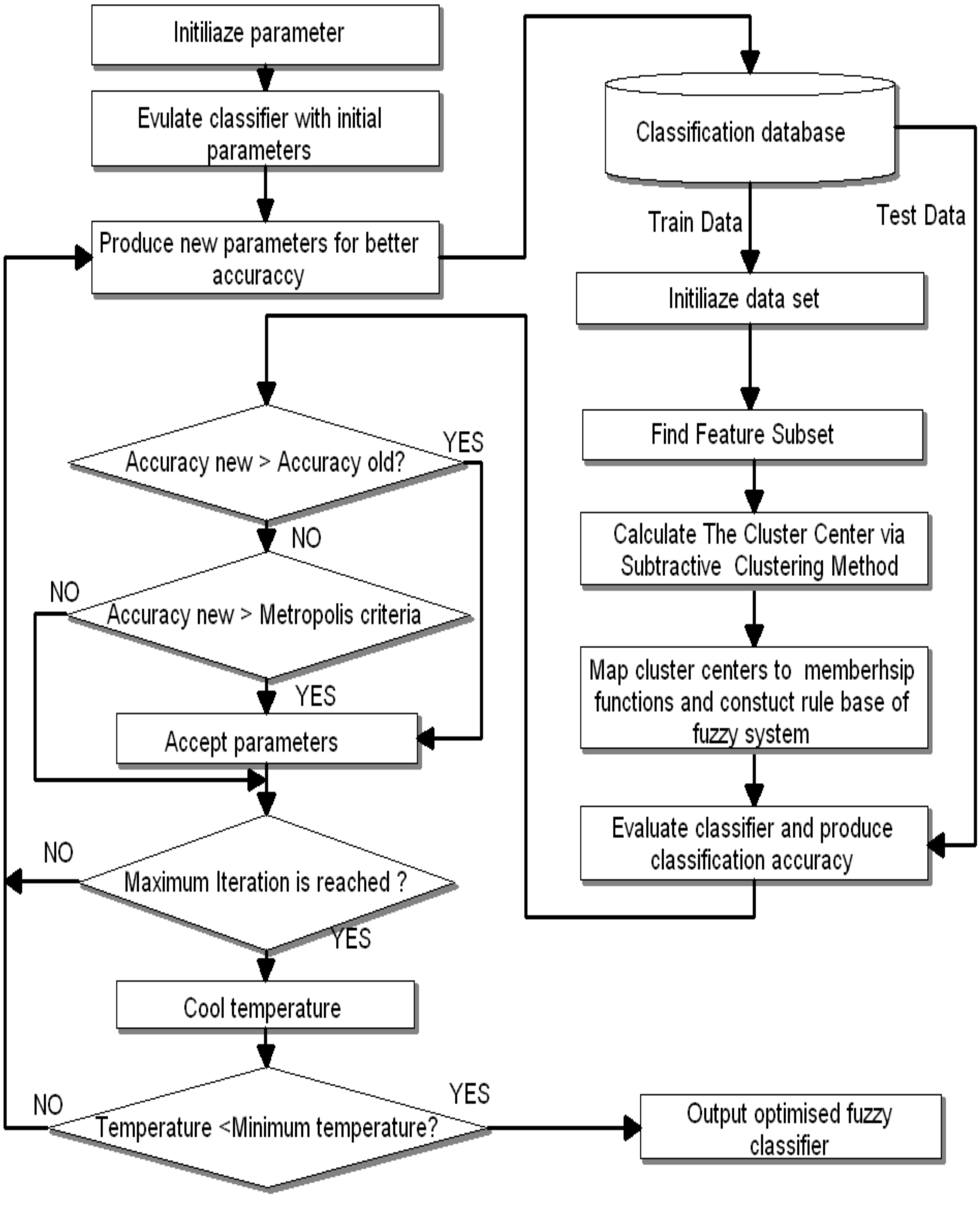

Initiliaze parameter
Evulate classifier with initial parameters
Produce new parameters for better accuraccy
Classification database
Train Data
Test Data
Initiliaze data set
Find Feature Subset
Calculate The Cluster Center via Subtractive Clustering Method
Map cluster centers to memberhsip functions and constuct rule base of fuzzy system
Evaluate classifier and produce classification accuracy
Accuracy new > Accuracy old?
YES
NO
Accuracy new > Metropolis criteria
NO
YES
Accept parameters
Maximum Iteration is reached ?
NO
YES
Cool temperature
Temperature <Minimum temperature?
NO
YES
Output optimised fuzzy classifier

Figure 6.1 Flow chart of SASCFC

The probability of acceptance of a worst solution is high at high temperature while it converges to zero when temperature goes down. The algorithm searches for new parameters until the temperature reaches the predefined minimum temperature. After the minimum temperature is reached or maximum accuracy is obtained, the algorithm stops its search and outputs the optimized fuzzy classifier with optimized input feature subspace.

6.2.1 SASCFC-Type1

Basic problems with constructing fuzzy system are obtaining membership functions, finding locations of them and constructing fuzzy if –then rule base. The SASCFC-Type1 classifier uses SA algorithm and SC method to realize the steps of ***ii, iii***, and ***iv*** which are stated in Section 4. In order to achieve these steps, algorithm tries to optimize the SC neighborhood radius, r_a, which provides finding optimum clusters. The number of clusters specifies both the number of membership functions for each input and the number of rules. Fig. 6. 2. demonstrates how cluster centers are transformed to membership functions.

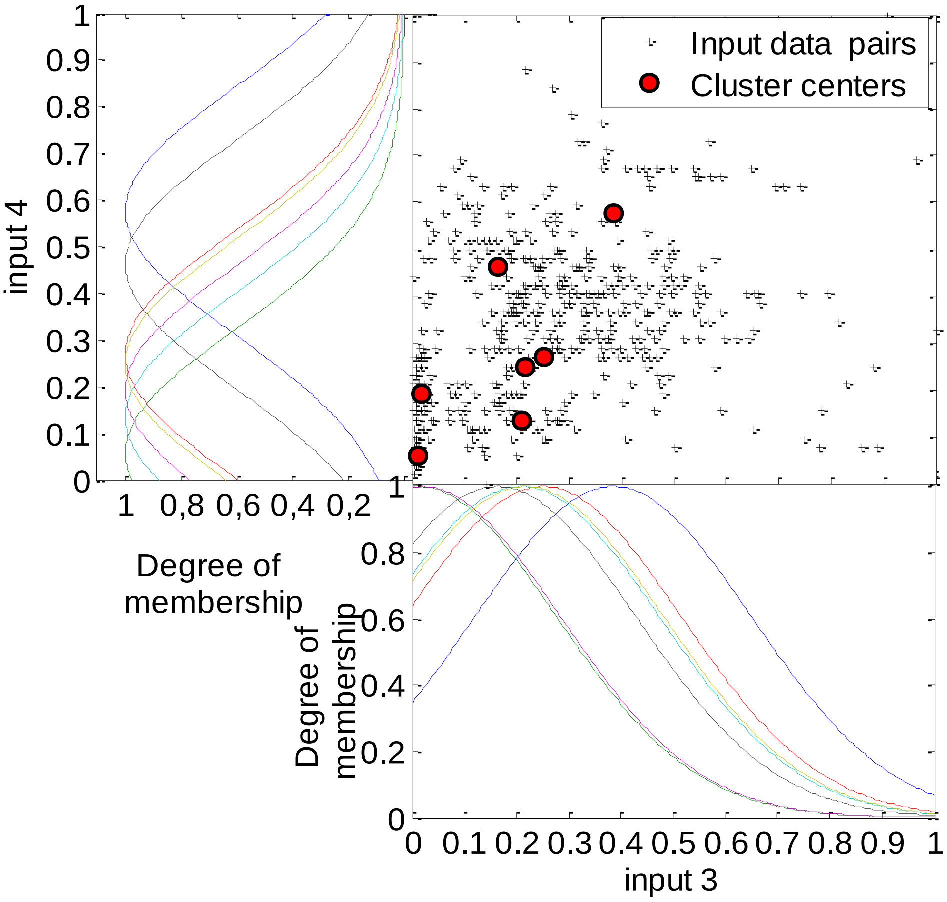
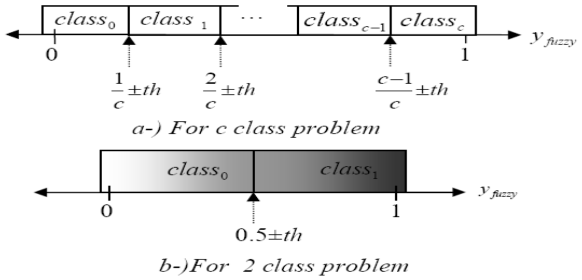


Figure 6.2 Membership Functions generation using clusters

6.2.2 SASCFC-Type2

A multi input single output fuzzy classifier produces crisp output, y_{fuzzy}. In order to test the classifier, the produced outputs must be compared with the actual output which is generally integers that represent to output classes. For this reason, the output of the fuzzy classifier must be mapped to the output class by using some threshold value as illustrated in Fig6.3.



a-) For c class problem

b-)For 2 class problem

Figure 6.3 Mapping fuzzy output to the classes

We developed a novel mapping function with adjustable parameter *th* for *c* class problem with output normalized to unit hypercube;

$$output_{class} = \left\{ \begin{array}{lll} class_1 & if & y_{fuzzy} \le \frac{1}{c} - th \\ class_2 & if & y_{fuzzy} \le \frac{2}{c} - th \\ \ldots & . & \\ class_{c-1} & if & y_{fuzzy} \le \frac{c-1}{c} - th \\ class_c & & else \end{array} \right\} \tag{6.6}$$

,where *th* can be chosen in the interval as ;

$$th = \left[-\frac{1}{2c}, \quad \frac{1}{2c} \right] \tag{6.7}$$

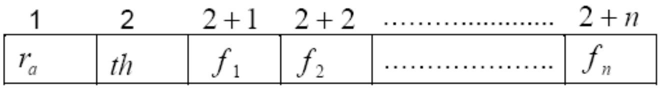
Besides the obtaining optimized membership functions and rules as in Type1, Type 2 algorithms produces optimized output threshold value *th* within a bounded interval.

6.2.3 SASCFC-Type3

It is important to know the effect of each input to the output. Modeling task deals with finding relations between inputs and outputs. But in many cases, the input dimension that describes the object which will be modeled may be very large and include irrelevant, redundant and noisy information. In these cases, the classifier performance gets worse and computational time increases. Feature selection or feature reduction is a technique of selecting a subset of relevant features for building robust learning models. According to the work of Abe and Kudo in 2006 [73], the main benefits of feature selection are as follows;

- Reducing computational cost and storage requirements
- Dealing with the degradation of classification efficiency due to the finiteness of training sample sets
- Reducing training and prediction time
- Facilitating data understanding and visualization

The SASCFC-Type3 algorithm has features of Type2 algorithm that searches optimum fuzzy rule base, membership functions and output threshold value. In addition to Type2, Type3 algorithm also searches the most proper input attributes feasible region. The algorithm outputs the fuzzy classifier with optimum membership functions, rule base and input features. Fig. 6. 4 shows the configuration space, *S*, in which r_a and th are continuous parameters while f_1, f_2 ,..., f_n are integers that either 1 or 0, which defines the corresponded feature will be selected or removed.



1	2	$2+1$	$2+2$	…………………	$2+n$
r_a	th	f_1	f_2	…………………	f_n

Figure 6.4 Configuration space

6.2.4 SASCFC-Type4

Wrapper types feature selection algorithms generally achieve better performance while consumption task is larger and filter type algorithms achieve lower performance within a lower time [83]. In case of too many features for a classification problem, iteration number of the SA at a temperature state must be big enough in order to find optimum configuration. It is also possible that Type3 algorithm couldn’t find the optimum feature subspace in case of too many input features with reasonable iteration number. For example, in Sonar Data Classification problem [134], there are 1.1529e+018 alternative feature subsets for 60 input, and configuration space has 62 parameters. It causes to increase the consumption time dramatically. In order to overcome this drawback, a statistical analyze is applied at data initialization phase to remove redundant and noisy features.

Parameter optimization of SASCFC-Type4 algorithm is similar with Type3 algorithm except feature selection procedures. In order to overcome the problem of finding relevant features within a reasonable consumption time, SASCFC-Type4 algorithm uses a hybrid feature selection algorithm which is a combination of filter type feature selection method with wrapper type method. Filter methods use statistical analyze on data for irrelevancy, redundancy and noisy. Relevancy is usually characterized in terms of correlation between two variables.The correlation coefficient $R(X,Y)$between two random variables X and Y with expected values μX and μY, and standard deviations σ_X and σ_y as follows

$$R(X,Y) = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_y} = \frac{E((X-\mu X)(Y-\mu Y))}{\sigma_X \sigma_y} \qquad (6.8)$$

where E is the expected value operator and *cov* means covariance. The value of $R(X,Y)$ lies between -1 and +1. If X and Y are completely correlated, $R(X,Y)$ takes the value of 1 or -1 and $R(X,Y)$is 0 when X and Y are independent. In SASCFC, linear correlation coefficients of each input versus output are calculated according to the Eqn. 6.8 with the Matlab function *corrcoef* . If the current input is relevant with output, then the correlation coefficient will be close to 1 while it is zero when current input is irrelevant with output. The inputs whose correlation coefficients greater than 0.05 are chosen as elements of $Subset_1$. The variances or standard deviations of inputs give information about noise. If the variance of input is too low, the positive effect of its on classification accuracy will be low. The inputs, whose variance is greater than half of the mean of corresponded inputs, are chosen as the elements of $Subset_2$. If the some inputs have similarity between each other, although they don’t decrease the accuracy, they increase total computational time. Variances, standard deviations and correlation coefficients of inputs of sonar data are demonstrated in Fig. 6. 5. The features, $f_{16}, f_{17}, f_{18}, f_{25}, f_{26}, f_{29}, f_{30}$, $f_{38}, f_{40}, f_{41}, f_{57}$ whose correlation coefficients are less than .05 are removed to obtain $Subset_1$.

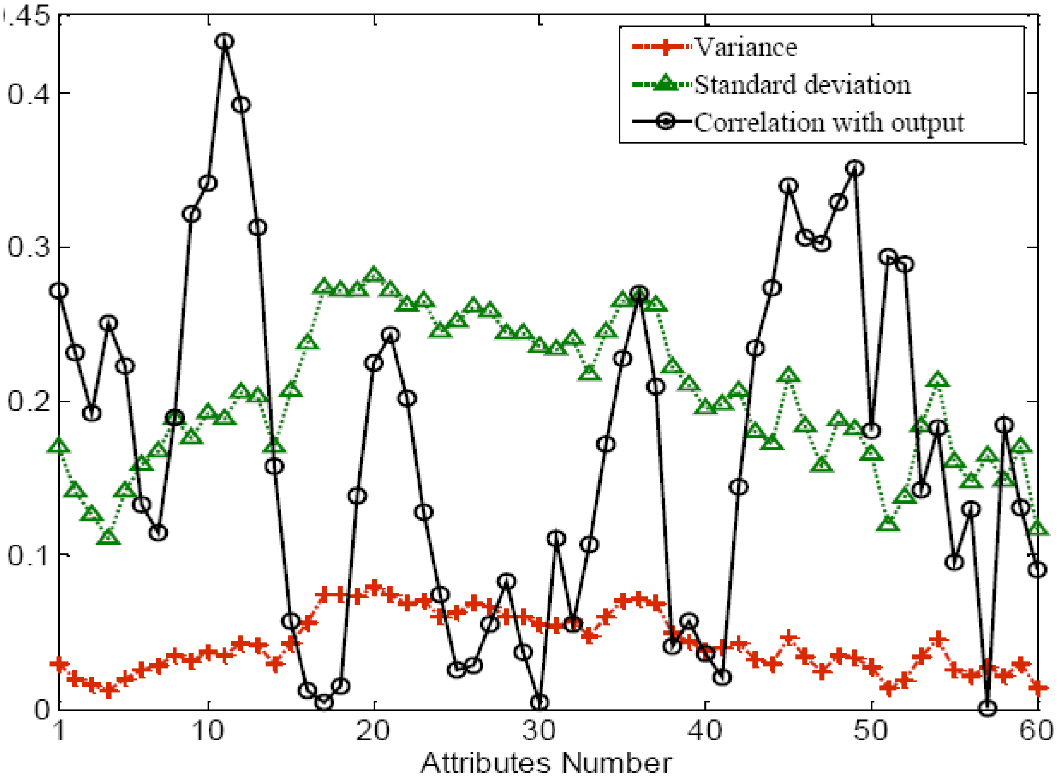


Figure 6.5 Variances, standard deviations and linear correlation coefficients of the sonar data set input features [134]

Redundancy analyze is performed by calculating the cross correlation coefficient of inputs with each others. If any two inputs are very similar, their correlation coefficients tend to 1. If cross correlations of an input with other inputs are higher than 0.95, one of them is leaved while others are pruned. $Subset_3$ is obtained by removing redundant feature from all input feature space. Fig. 6. 6 shows the correlation counter plot of input variables for sonar data set. Highly correlated input features with each others, $f_{15} - f_{16}, f_{17} - f_{18}, f_{20} - f_{21}$ are obtained and one element in each pair is removed such as f_{15}, f_{17}, f_{21} to form $Subset_3$.

A core subset $Subset_{core}$ is produced by union of the $Subset_1$ and $Subset_2$ at initialization phase of the algorithm. After data initialization, SASCFC-Type4 generates *a* $Subset_4$ by adding new feature to the core set, $Subset_{core}$, in new configuration generation phase of SA. The generated $Subset_4$ still may have redundant feature. Intersection operator is applied to remove redundant feature between $Subset_3$ and $Subset_4$ to obtain final subspace. SASCFC-Type4 uses same configuration space as Type3, that aims to optimize, r_a, th and feature subspace F as shown in Fig. 6.4.

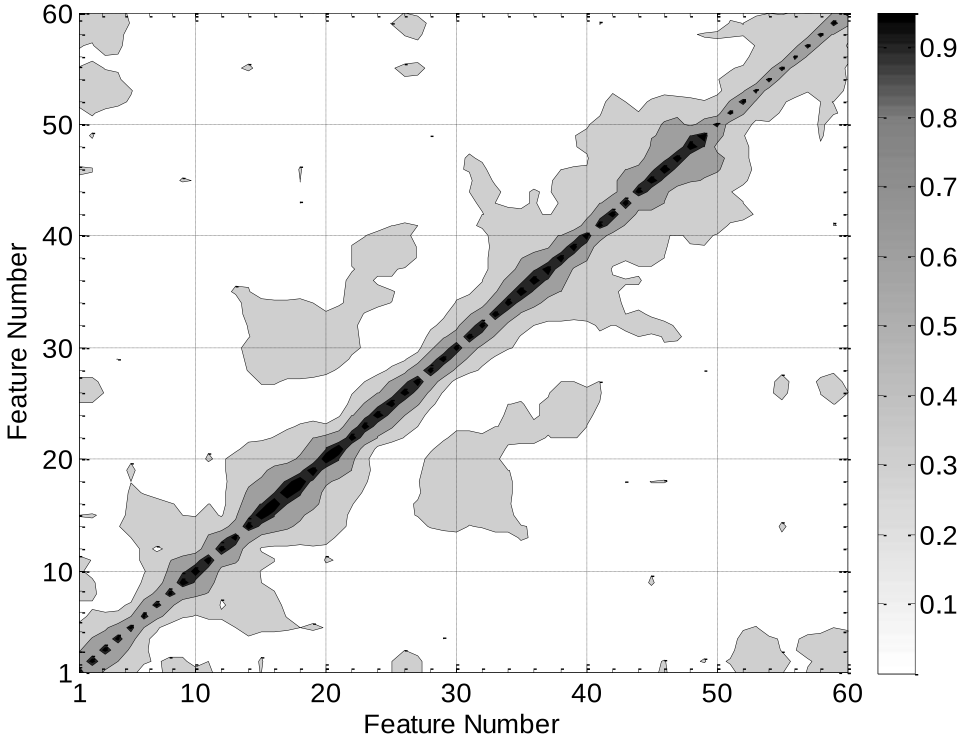


Figure 6.6 Counter graph representation of linear cross correlation coefficients of inputs with each others for sonar data set [134]

6.3 Experimental Results and Discussions

Proposed algorithm’s codes are written in Matlab software with m-file format which is given in Appendix chapter and evaluated on a personal computer with Pentium IV 2.6 GHz CPU and 1 GB of RAM. In order to prove the robustness of proposed classifiers, twelve classification problems [117, 134] whose specifications are shown in Table 6.2, are chosen.

In the initializing phase, all attributes and class outputs are normalized into the unit interval [0 1]. Classifier validations are obtained with k-fold cross validation procedure [135]. The k-fold cross validation relies on a random portioning of the data set into k parts. Then, one part is used for testing while the remainder parts are used for training. This procedure is repeated *k-1* times in order to use every part. Finally k classification estimates are averaged to yield a robust overall classification value. Because the SASCFC uses power of SA, and k-fold partitions the data randomly, the outputs of the SASCFC may not be same for each run despite using same data set. For this reason, each SASCFC is executed 10 times for each data set, and average values are taken. The classification performances are measured with classification accuracy both in training and testing phase with standard deviations. The developed programming codes are given in Appendix. As described in Section3, SASCFC algorithm uses different SA parameters at low and high temperature region. The used SA parameters are given in Table 6. 3.

Table 6.2 Used data sets and their specifications

Data Set	#Instance	#Attribute	# Class	Dev.Cla (%)	Maj.Cla (%)	Min.Cla (%)
Pap Smear (smr,[117])	500	24	2	28.28	70	30
Breast Cancer (brst,[134])	683	9	2	21,22	65,01	34,99
Pima Indians(pima,[134])	768	8	2	21,36	65,1	34,9
Sonar(snr,[134])	208	60	2	4,76	53,37	46,63
Ionosphere	351	33	2	19,94	64,1	35,9
Heart Disease(hrt,[134])	270	13	2	7,86	55,56	44,44
Bupa Liver(bupa,[134])	345	6	2	11,27	57,97	42,03
Australian (cra,[134])	690	14	2	7,79	55,51	44,49
Fisher Iris(iris,[134])	150	4	3	0	33,33	33,33
Balance[bswd,[134]]	625	4	3	22,08	46,08	7,84
Wine(wine,[134])	178	13	3	6,46	39,89	26,97
Waveform(wave,[134])	5000	21	3	0,52	33,92	32,94

Dev. Cla is the standard deviation of class distribution, *Maj. Cla* and *Min.Cla* percentage of majority and minority class instances respectively.

In experiments, firstly pap-smear classification performed by the SASCFC-Type1 and the performance of the classifier is given in Table 6. 4. The performance measurements are done by calculating, Correct Classification Number (CCN), Incorrect Classification Number (IC), Incorrect Positive Classification (IPC), Incorrect Negative Classification (INC) and Classification Accuracy (CA) of the classifier for each fold. According to the results, a 98.8 % accuracy rate is achieved by the SASCFC-Type1 algorithm in testing phase. This accuracy is a reasonable accuracy rate according to the recent works about pap-smear classification problem in literature [127].

Table 6.3 Parameters of SASCFC

Parameters	*T>1e-3*	*T < 1e-3*
Initial Temperature, T_{init}	1	1
Cooling procedure	*T*=0.8 x*T*	T=0.8x*T*
Final Temperature, T_{final}	1,00e-05	1,00e-05
Number of iteration at a temperature state, *N*	300	100
Number of acceptance at a temperature state	150	50
Number of rejection at a temperature state	150	50
New parameter generation mechanism	Eqn.6.5	Eqn.6.3

In the work of [72], the k-fold cross validation is used with a different process in training and test phase of the classifier. The authors used SA for optimization of back propagation neural network based classifier (SABPN). Instead of validating an optimization configuration for all k folds, in each fold they optimized the network and tested within the current fold. In order to compare SABPN classifier with SASCFC-Type1 classifier, we run the SASCFC-Type1 with the same optimization strategy in k-fold cross validation as in [72]. According to the each fold, the new radii r_a and rules are optimized with optimized rule number (RN). As shown in Table 6.5, the accuracy of SASCFC-Type1 classifier has better performance than SABPN [72] for breast cancer diagnosis (brst).

Table 6.4 Results for classification of pap-smear data set with the SASCFC-Type1 with optimized $r_a = 0.9214$ which yields three cluster centers

Folds	CCN	IC	IPC	INC	CA(%)
1	49	1	1	0	98
2	50	0	0	0	100
3	50	0	0	0	100
4	48	2	1	1	96
5	48	2	1	1	96
6	50	0	0	0	100
7	50	0	0	0	100
8	49	1	0	1	98
9	50	0	0	0	100
10	50	0	0	0	100
Mean	49.4	0.6	0.3	0.30	98.8

Table 6.5 Breast Cancer Classification with SASCFC-Type1 and SABPN [72]

	SASCFC-Type1		SABPN[72]		
Folds	Accuracy(%)	r_a	RN	NHN	Accuracy(%)
1	98.53	0.547	1	1	98.55
2	95.65	0.382	2	2	97.10
3	100.00	0.099	17	3	98.55
4	100.00	0.099	17	3	95.65
5	95.59	0.205	5	1	98.53
6	100.00	0.284	2	1	100.00
7	100.00	0.284	2	1	100.00
8	98.55	0.543	1	1	98.53
9	100.00	0.216	3	2	97.06
10	100.00	0.216	3	2	98.59
Mean	**98.83**				98.26

The configuration space is different for each type SASCFC as shown in Table 6. 1. Optimized parameters and architectures of the classifiers are shown for smr and Ion data set in Table 6.6. and Table 6.7 respectively. Output threshold values are close to zero while r_a values are very different from the suggested in [49]. Improvement is done on execution time, classification accuracy and classifier complexity for smr data set with feature selection, while classification accuracy is a bit behind from without feature selection for snr data set. Improvement on execution time is eventful although classification accuracy of Type3 algorithm less with percentage %.059 than Type2 algorithm. This also provides the classifier complexity and architecture which show the input output relations in more readable form with fewer rules.

Table 6.6 Basic parameters of Optimized classifiers for Pap Smear data(smr)

Optimized Parameter	SASCFC Type1	SASCFC Type2	SASCFC Type3	SASCFC Type4
th	0	0.1040	0.0237	0.0849
r_a	0.9214	0.9213	0.8597	0.9847
# Feature	24	24	9	13
# Rule	3	3	1	2
Execution Time	3.11sec	2.86sec	1.516sec	2sec
Accuracy(%)	98.8±1.69	98.8±1.4	98.8±1.69	99.00±1.05

Table 6.7 Basic parameters of Optimized classifiers for Sonar data(snr)

Optimized Parameter	SASCFC Type1	SASCFC Type2	SASCFC Type3	SASCFC Type4
th	0	0.1438	0.1727	0.145
r_a	0.9546	0.9022	0.9366	0.9554
# Feature	60	60	26	54
# Rule	144	150	12	128
Execution Time	44.062sec	43.746sec	2.922 sec	32.812 sec
Accuracy(%)	90.35 ± 3.31	91.99 ± 7.39	**92.75 ± 5.39**	91.86 ± 5.88

Time consumptions of each type of SASCFC are demonstrated in Fig. 6.7 with mean execution time for all data set. Because there is no feature selection process in Type1 and Type2, the execution time is significantly higher than other two types. Despite of both Type3 and Type4 have feature selection characteristic , the execution time of Type4 is higher than Type3 because hybrid feature selection methodology of Type4 searches new feature subspace by adding new features to core feature subset which generally causes more rules.

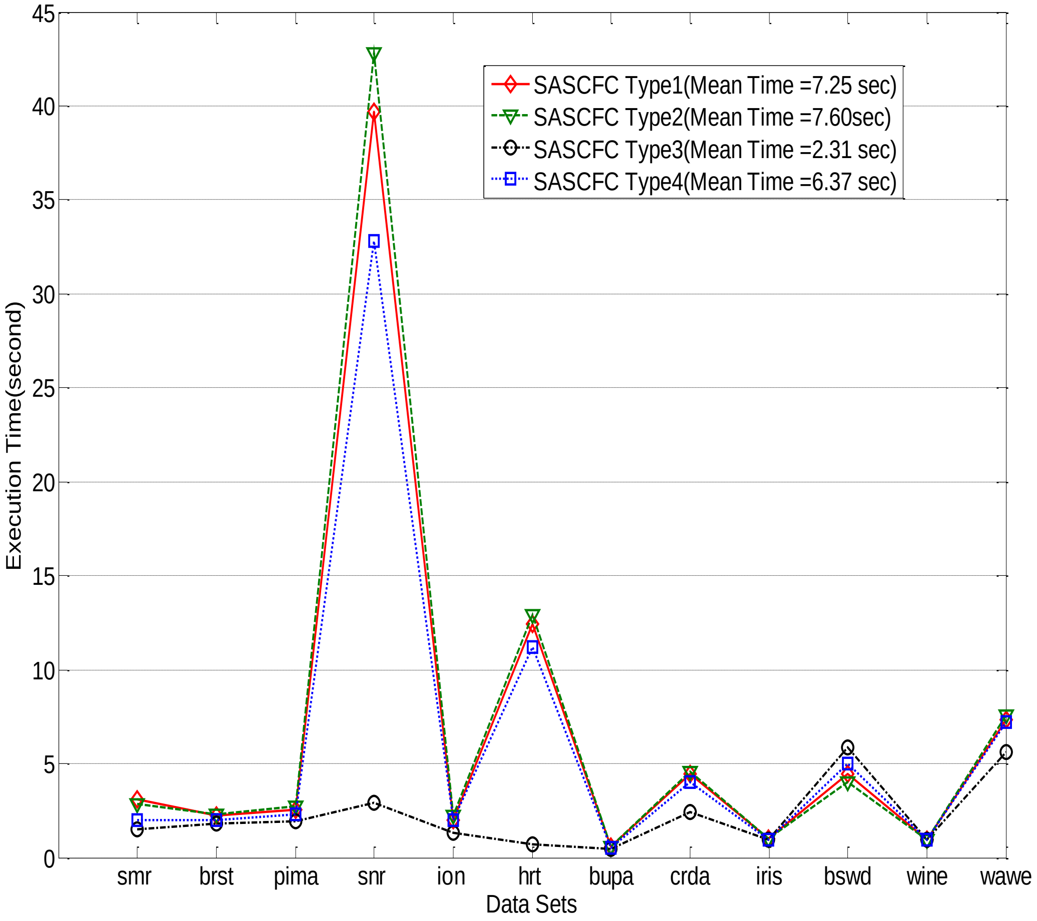


Figure 6.7 Consumptions times of each types of SASCFC

Performance comparisons of SASCFC-Type1, Type2, Type3 and Type4 algorithms in training and testing phase are given in Table 6. 8 and Fig. 6. 8. According to Table 6.8, it can be concluded that feature selection improve both classification accuracy and standard deviation in both training and testing phase, for high feature size problems as snr, ion and smr classification problem. However, for low feature size problem as bswd and iris, feature selection decreases the classification accuracy. For medium feature size problem as *brst*, *pima* and *bupa*, although performances of classifier which have feature selection algorithm are higher than the classifiers which have no feature selection algorithm, standard deviation gets worst. In generally, it can be seen that Type4 algorithm reaches the best average accuracy rate as expected for all data set in testing phase while Type2 achieves the best average value in training phase. In fact, classifier complexity, readability of input output relations and consumption time, in addition to the classification accuracy, are significant criteria in comparing the classifier. For this reason, it can be concluded that, the best successful classifier is SASCFC-Type4.

Table 6.8 Classification results of SASCFC-Type1, Type2, Type3 and Type4

Data Set		SASCFC-Type1	SASCFC-Type2	SASCFC-Type 3	SASCFC-Type 4
Pap Smear	Test.%	98.8 ± 1.69	98.8 ± 1.4	98.8 ± 1.69	**99.00 ± 1.05**
(smr,[117])	Train %	99.62 ± 0.26	99.64 ± 0.11	99.58 ± 0.22	99.51 ± 0.18
Breast Cancer	Test %	96.48 ± 2.42	97.67 ± 2.19	97.67 ± 1.96	**97.67 ± 2.84**
(brst,[134])	Train %	96.31 ± 0.49	97.61 ± 0.29	97.62 ± 0.28	97.67 ± 0.31
Pima Indians	Test %	78.64 ± 2.24	78.52 ± 3.22	78.52 ± 4.23	**78.78 ± 3.79**
(pima,[134])	Train %	78.66 ± 0.32	78.4 ± 0.69	78.47 ± 0.73	78.5 ± 0.86
Sonar	Test %	90.35 ± 3.31	91.99 ± 7.39	**92.75 ± 5.39**	91.86 ± 5.88
(snr,[134])	Train %	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Ionosphere	Test %	89.74 ± 5.99	89.76 ± 6.97	**90.91 ± 3.37**	89.68 ± 4.65
(ion,[134])	Train %	97.34 ± 0.94	95.82 ± 1.13	97.18 ± 0.6	96.68 ± 0.55
Heart Disease	Test %	80.37 ± 9.48	81.11 ± 8.27	80.74 ± 8.69	**81.11 ± 5.91**
(hrt,[134])	Train %	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Bupa Liver	Test %	73.6 ± 5.13	73.9 ± 5.01	73.93 ± 9.95	**74.13 ± 12.7**
(bupa,[134])	Train %	73.49 ± 1.11	73.01 ± 1.24	73.2 ± 1.31	73.14 ± 1.15
Australian	Test %	85.37 ± 4.77	**85.64 ± 4.31**	85.53 ± 3.33	85.52 ± 2.85
(cra,[134])	Train %	91.95 ± 0.66	92.16 ± 0.8	91.95 ± 0.36	91.92 ± 0.63
Fisher Iris	Test %	**98.00 ± 3.22**	98.00± 3.22	**98.00 ± 3.22**	97.33 ± 4.66
(iris,[134])	Train %	97.19 ± 0.58	97.70 ± 0.62	97.41 ± 0.52	96.07 ± 0.61
Balance&scale	Test %	**91.84 ± 3.59**	91.51 ± 2.02	86.88 ± 4.5	91.21 ± 2.73
(bswd,[134])	Train %	94.95 ± 0.25	94.38 ± 0.75	94.13 ± 1.06	**95.06 ± 0.87**
Wine	Test %	97.22 ± 4.72	99.41 ± 1.86	98.92 ± 2.28	**99.44 ± 1.76**
(wine,[134])	Train %	98.5 ± 1.24	100.0 ± 0.00	100.0 ± 0.00	100.0 ± 0.00
Waveform	Test %	77.70 ± 0.72.	77.90 ± 0.7	78.69 ± 5.05	79.8 ± 4.34
(wave,[134])	Train %	83.91 ± 1.42	84.41 ± 1.21	82.63 ± 1.01	81.66 ± 0.59
Mean	Test %	88.18 ± 3.96	88.63 ± 3.94	88.33 ± 4.55	**88.79 ± 4.43**
	Train %	92.66 ± 0.49	**92.7 ± 0.45**	92.68 ± 0.54	92.52 ± 0.48

Another recent work with optimization of fuzzy classifier [20] which is used simulated annealing algorithm to find optimum if-then rule base of a fuzzy classifier.

The proposed classifier are compared by well known classifier [136] as; decision tree induction algorithm (C4.5), the nearest neighbor classifier technique (IBk), simple Bayesian network based classifier (N.B), support vector machines based classifier (SVM), Pittsburgh genetic-based machine learning system (GAs). Detailed descriptions of each tool can be found in corresponding references. The performances of C4.5, IBk, N.B, SVM and GAs type classifiers are taken from [136] and SAFC from [20]. Comparisons with SASCFC-Type2 and other classifiers for seven classification data sets are shown in Table 7. Although, classification performance of Type2 isn't as good as Type4, Type2 is chosen for comparison because there is no feature selection algorithm in [20, 136].

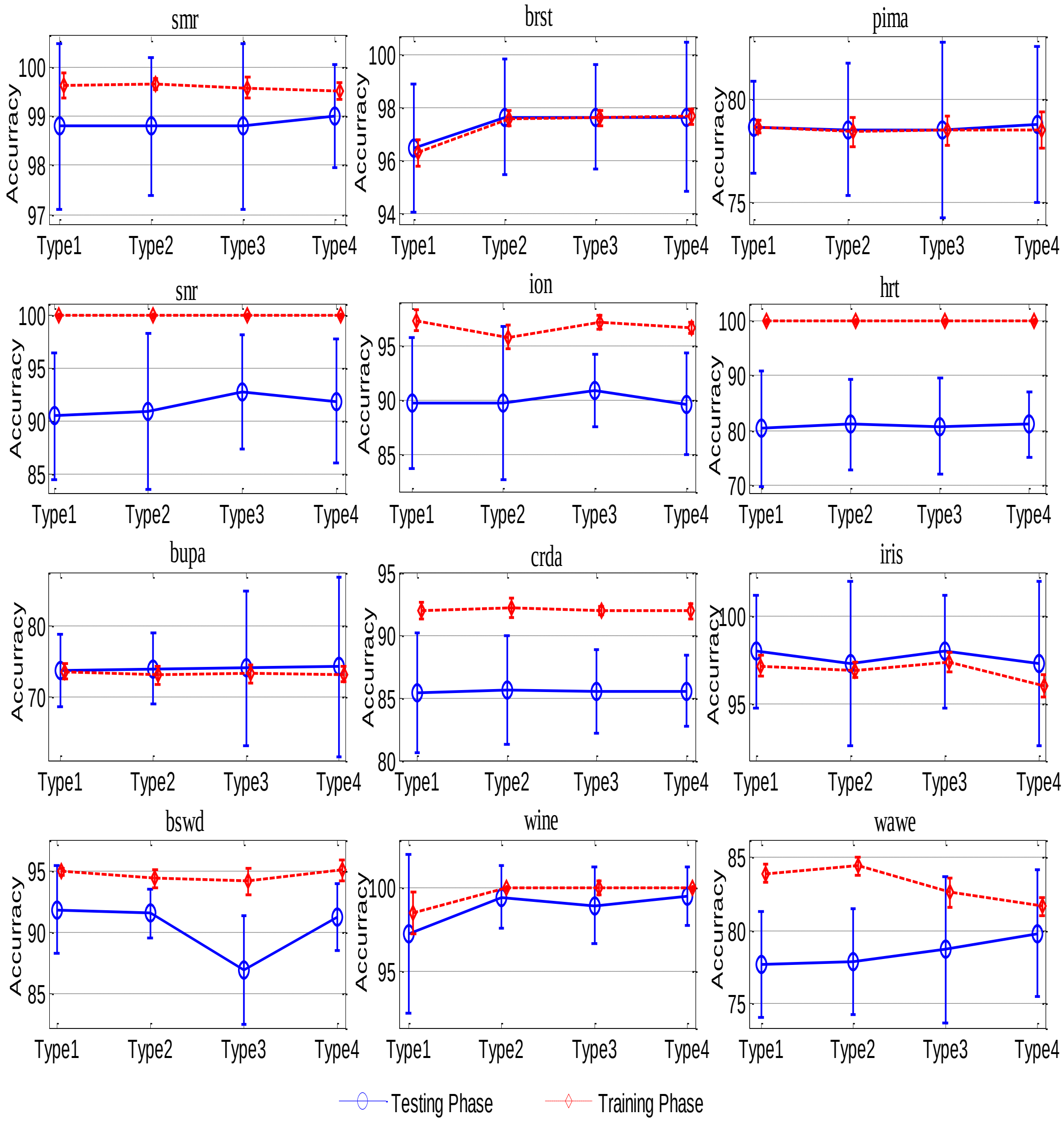


Figure 6.8 Classification performance of proposed classifiers

Table 6.9 Classification performance of some well known classifiers

Accuraccy		C4.5	IBk	N.B	SVM	GAs	SAFC	SASCFC-Type2
Bswd	Train %	89.93±0.68	90.53±0.54	91.92±0.25	91.01±0.19	92.14±0.28	**94.63±0.46**	94.38±0.75
	Test %	77.66±2.91	86.09±2.72	91.43±1.25	90.9±1.43	89.62±2.22	90.47±1.36	**91.51±2.02**
Cra	Train %	90.31±0.86	91.05±0.52	82.58±0.82	55.51±0.08	91.07±0.73	**94.25±0.54**	92.16±0.8
	Test %	85.55±3.45	84.73±4.04	81.07±5.32	55.51±0.7	85.62±4	**85.77±3.27**	85.64±4.31
Ion.	Train %	98.68±0.54	90.94±0.59	93±0.42	94.19±0.64	96.9±0.74	**99.66±0.34**	95.82±1.13
	Test %	88.97±5.91	85.66±4.66	91.5±4.7	92.14±4.62	**92.71±5.01**	91.89±4.65	89.76±6.97
Iris	Train %	98±0.61	96.59±0.49	96.67±0.53	97.11±0.64	98.33±0.79	**99.85±0.19**	97.80 ±0.62
	Test %	94.22±5.37	94.89±6.37	96.22±5.36	96.22±4.77	95.2±5.87	96.66±3.09	**98.00±3.22**
Pima	Train %	84.43±2.41	85.67±0.65	77.07±0.61	78.27±0.53	83.11±0.82	**87.55±0.59**	78.40±0.69
	Test %	75.44±4.79	74.52±3.91	75.3±4.45	77.32±4.7	74.46±5.19	75.71±4.41	**78.52±3.22**
Wine	Train %	98.86±0.54	97.27±0.53	98.67±0.45	99.33±0.32	100.0±0.00	99.98±0.04	**100.0±00**
	Test %	94.24±6.44	96.61±4.02	97.2±3.43	98.1±3.4	96.33±4.13	97.63±3.02	**99.41±1.86**
Wave	Train %	**97.29±0.61**	0±0	81.59±0.21	0±0	78.28±0.6	85.02±0.18	84.41±0
	Test %	75.93±2.1	0±0	79.89±1.4	0±0	76.01±1.97	**80±1.16**	77.9±0

For bswd classification, the performance of Type2 is better than all counterparts in testing while it is a bit lower than only SAFC in training phase. Comparison on crda classification is not clear because all classifiers have almost same performance except SVM. In ion data set, Type2 performance isn’t as good as the performance of N.B, SVM, GAs and SAFC while a bit better than performance of C4.5 and IBk. Type2 achieves the best performance in iris data classification in testing phase with acceptable standard deviation while it is lower than the performances of C4.5, GAs and SAFC in training phase. A similar score is in pima in which Type2 achieves the best accuracy rate and standard deviation in testing phase, while in training phase, its performance isn’t better than its counterparts. For wawe data, that SVM and IBk performance are absent in [136], performance of Type2 is betwixt that although it is better than performances of C4.5 and Gas, it is worse than the performance of SAFC and N.B. The experimental results for wine data set shows that Type2 reaches best performance accuracy and standard deviation in both training and testing phase.

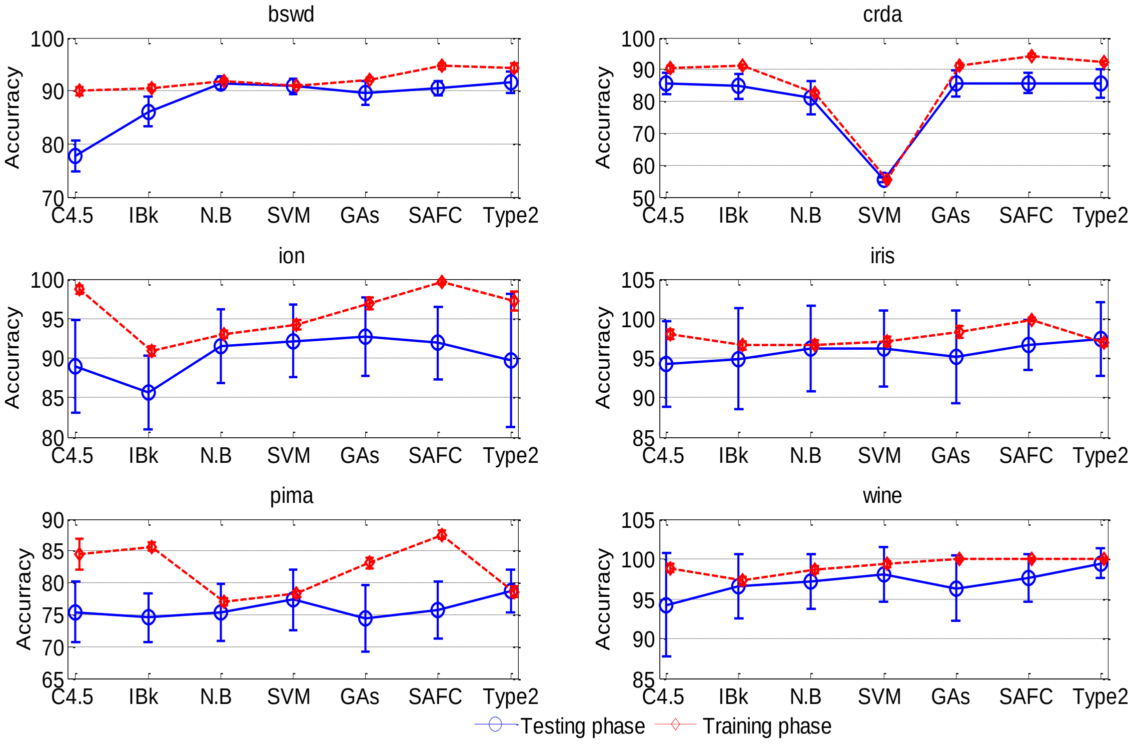


Figure 6.9 Classification performances of some well known classifiers

6.4 Conclusions

In this chapter, a novel optimization algorithm is given in order to obtain proper fuzzy classifier. The optimization starts with clusters centers which are found by SC . Then cluster centers are transformed to membership functions of input variables and rules of fuzzy classifier. In the next step, output threshold value is optimized which yields an improvement in classification accuracy. Finally a wrapper type feature selection and a simple hybrid feature selection approaches are introduced. Experimental results show that, although satisfactory improvement on the accuracy performance is not obtained by wrapper approach, it enables to reduce the classifier complexity which directly influences on total rule size and consumption time of the classifier. Hybrid feature selection approach improves both performance and complexity of classifier and experimental results show the best accuracies are achieved by this approach. The comparisons of SASCFC-Type2 classifier with well known classifiers lend countenance that proposed classifier is grateful approach since four best accuracies are achieved within seven data sets in testing phase.

CHAPTER 7

COMPETITIVE LEARNING BASED NEUROFUZZY CLASSIFIER

Clustering methods are widely used in structure learning phase of both neural network and fuzzy inference based systems as fuzzy c-means, k-means clustering, mountain clustering, subtractive clustering, and agglomerative clustering. Detailed reviews of clustering algorithms are addressed in Chapter 3 and in [48]. Rival Penalized Competitive Learning Clustering (RPCL) was proposed by Xu et al. in 1992 [63]. Some studies on improving performance of RPCL are found in [60, 64] and its application in construction of Radial Basis Function network is given [65]. RPCL has advantage that it doesn't need to know numbers of clusters center, therefore it can be regarded as an unsupervised clustering algorithm. However there is no work in literature that uses RPCL at initial structure of NeuroFuzzy Systems, and although NeuroFuzzy architectures are convenient for RPCL type back propagation there is also no study which aims to hybridization of RPCL and back propagation training.

In this chapter two new NeuroFuzzy Classifiers are proposed as NFC1 and NFC2. Initial structures of both classifiers are set up via rival penalized competitive learning based classifier. Parameter tuning of NFC1 is performed by gradient descent base back propagation batch training algorithm. Rule adaptation mechanism is embedded into training of the NFC2 that both parameters and structural optimization performed twice. It enables to change the structure of classifiers by adding new rules and deleting unnecessary rules, and improve the classifier performance. After initial structure obtained by rival penalized competitive learning, parameters of the NFC2 are tuned by incremental learning type back propagation algorithm. In fine tuning phase of structure, according to error criteria and rule firing counts criteria's structure of the NFC2 are reconstruct and final structure is retuned by back propagation algorithm. Mathematical derivations for proposed classifiers training algorithms are given. the NFC1 and the NFC2 also tested on two real world problems that performances are valuable. The developed programming codes are given in Appendix

7.1 Rival Penalized Competitive Learning Based Clustering

Competitive Learning has been developed for unsupervised learning in artificial neural network, clustering, and pattern recognition. In fact, it can be regarded as adaptive version of classical K means and fuzzy C-means clustering algorithm. Previous two algorithms need to know how many clusters will be obtained. Although some validity indexes have been proposed in the literature to solve for finding compact cluster numbers, it is hard task to decide in real world problem to decide the correct cluster number according to finding index.

In the literature one of the most popular index for hard clustering is the Davies Bouldin (DB) Index [137] which is the ratio of the sum of within-cluster distance to between-cluster separation, and is computed as follows:

$$DB(k) = \frac{1}{k}\sum_{i=1}^{k} max_{j,j\neq i}\left\{\frac{S_i+S_j}{d_{ij}}\right\}, \tag{7.1}$$

where within-cluster scatter for cluster *i* denoted S_{i} and the between-cluster separation for clusters *i* and *j*, denoted *dij*, are calculated by

$$d_{ij} = \left\|\mu_i - \mu_j\right\| \tag{7.2}$$

$$S_i = \frac{1}{|C_i|}\sum_{X\in C_i}\|x - \mu_i\| \tag{7.3}$$

Smaller values of DB represent better clustering, and hence the value that *minimizes* DB is the optimal number of clusters.

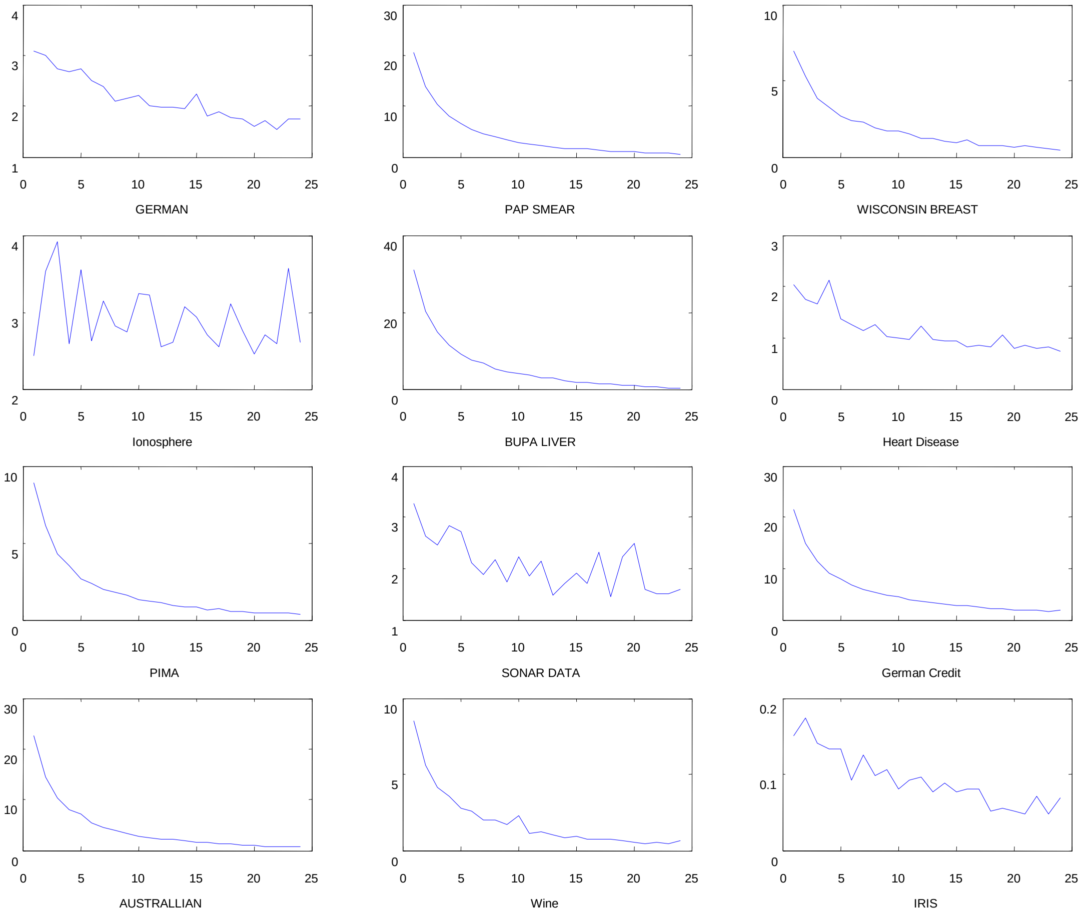


Figure 7.1 DB index for K-means clustering algorithm for some real world problems

Another Clustering index in recent literature in order to find compact cluster number in hard clustering tasks is known as Chou-Su or CS index [138].

A distance metric between any two data points x_i and x_j is denoted by,

$$d_{ij} = \|x_i - x_j\| \tag{7.4}$$

The CS measure can be defined as,

$$CS(k) = \frac{\frac{1}{k}\sum_{i=1}^{k}\left[\frac{1}{N_i}\sum_{x_q \in C_i} max\{d_{ij}\}\right]}{\frac{1}{k}\sum_{i=1}^{k}\left[min_{j \in K, j \neq i}\{d_{ij}\}\right]} \tag{7.5}$$

Where k is the number of cluster , N_i is the number of data in cluster i.

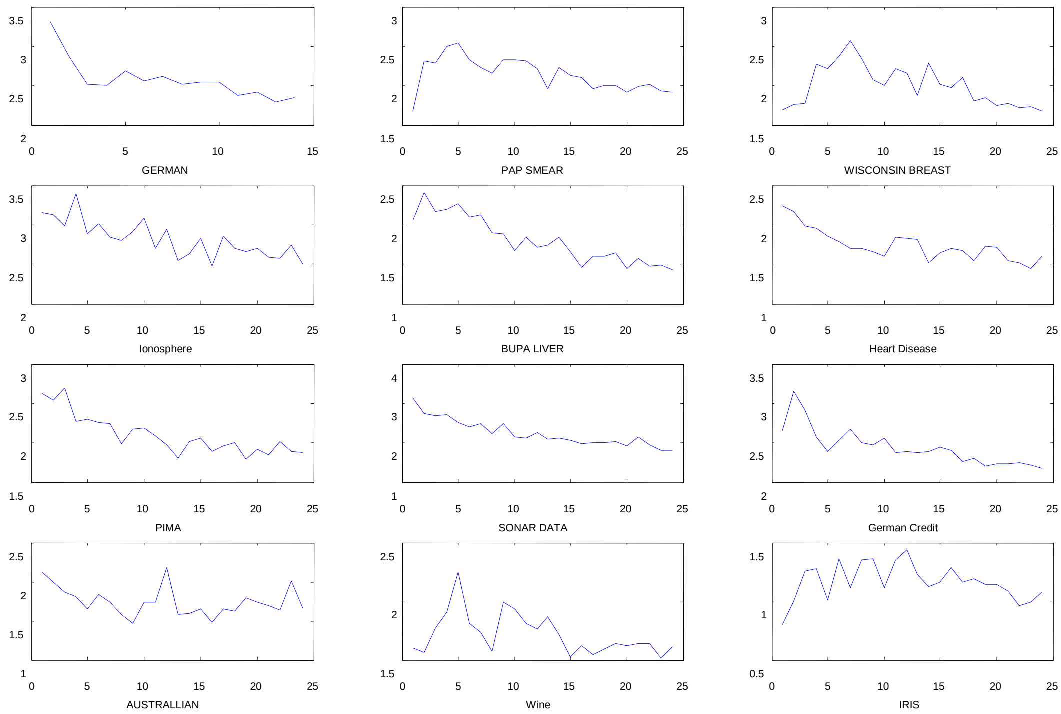


Figure 7.2 CS index for K-means clustering algorithm for some real world problems

Although the above mentioned indices give good results in artificial problem, obtaining the optimal number of clusters is still an open problem. Clustering methods use a fixed parameter, *k*, as the number of clusters. Such parameter is usually determined by a trial-and-error procedure in order to obtain a value that yields the best clustering results. In particular, for large data sets, there is no evidence that the value of *k* obtained is optimal (unless one knows the correct number of clusters based on the nature of the data set). As shown in Fig. 7.1 and 7.2, it is hard task to decide numbers of the optimal clusters for real world problems.

Rival penalized competitive learning (RPCL) based clustering algorithm which doesn’t need to know number of clusters is proposed by Xu [62] is used in order to obtain initial structure of NeuroFuzzy classifier proposed here. In RPCL, the number of clusters is adjusted automatically during learning, resulting in a flexible partitioning of input–output space, as well as the optimal number of fuzzy rules. An initial structure of the FNN is first constructed, after that, some nodes and links are deleted to form the final structure of the network as the number of clusters is adjusted. Assume that the availability of a data set composed by P input –output pairs,

$$S = \left\{z_p = \left\{x_p, y_p\right\}\right\} = \begin{bmatrix} x_{11} & x_{12} & \cdots.. & x_{1m} & y_1 \\ x_{11} & x_{12} & \cdots.. & x_{1m} & y_1 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots.. \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_{p1} & x_{p2} & \cdots.. & x_{pm} & y_p \end{bmatrix} \tag{7.6}$$

Where p is the number of patterns, and m is the number of attributes. The steps of RPCL algorithm as follows;

Step1; Take randomly some samples as initial cluster centers$\{c_i\}_i^k$, where k is the initial number of clusters

Step2; Randomly take an input-output pattern z_p from data set *S* and for *i=1,2,…k,* let

$$\mu_i = \begin{bmatrix} 1 & if\ i = winner \\ -1 & if\ i = rival \\ 0 & otherwise \end{bmatrix} \tag{7.7}$$

with

$$\gamma_{winner}||z - c_{winner}||^2 = min_i\gamma_i||z - c_i||^2 \tag{7.8}$$

$$\gamma_{rival}||z - c_{rival}||^2 = min_{i\neq winner}\gamma_i||z - c_i||^2 \tag{7.9}$$

where

$$\gamma_i = \frac{n_i}{\sum_{i=1}^{k} n_i} \tag{7.10}$$

where c_{winner} stands for winning cluster centers, c_{rival} is the rival clustering centers. n_i is the cumulative number of winning.

Step3;Uptade winning and rival centers,

$$c_{winner} = c_{winner} + \alpha_{winner}(t)(z - c_{winner}) \tag{7.11}$$

$$c_{rival} = c_{rival} - \alpha_{rival}(t)(z - c_{rival}) \tag{7.12}$$

Where $1>\alpha_{winner}>>\alpha_{rival}>0$ are the learning and de-learning rate.

*Step4;*if all pattern is proceeded finish algorithm, else go to Step2*;*

Suppose an artificial problem that contains 150 patterns with 5 distinct classes. By using half of data as training of RPCL, initial cluster centers and their behaviors in training phase are illustrated in Fig.7. 3. Initial number of clusters is set to square root of number of pattern times number of attributes. So for current problem, initial cluster size is 18. During the training phase winner cluster goes closest to the data while rival cluster is penalized so it goes far from the current data sample. After all sample is seeded, cluster centers which contains less data points than a threshold is deleted. In Fig 7.4, final obtained cluster centers are shown.

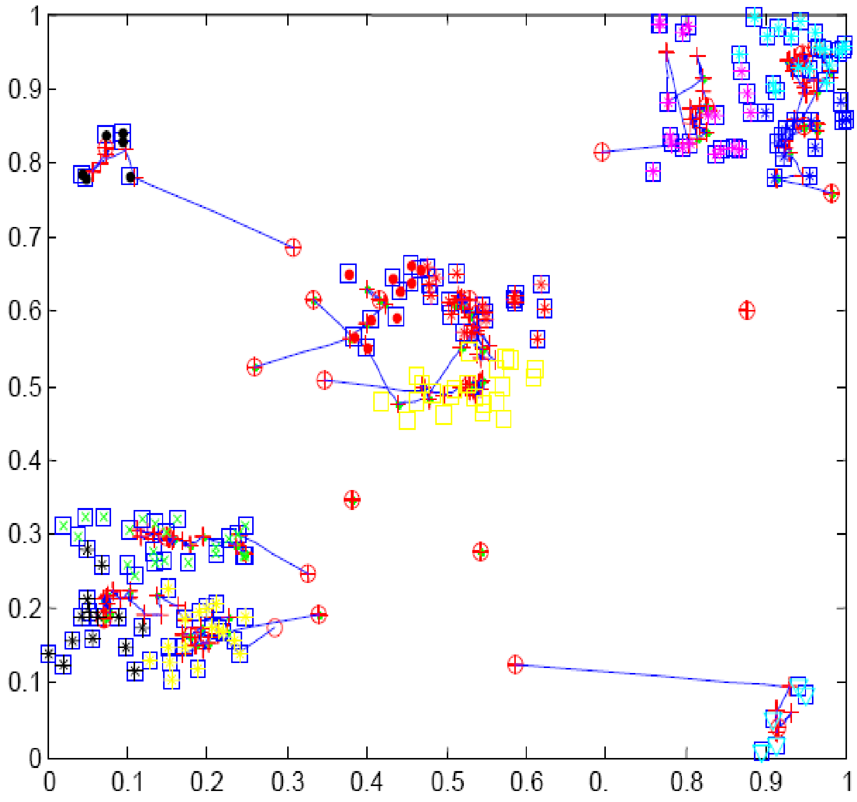


Figure 7.3 Cluster behaviors in training phase of RPCL for artificial data

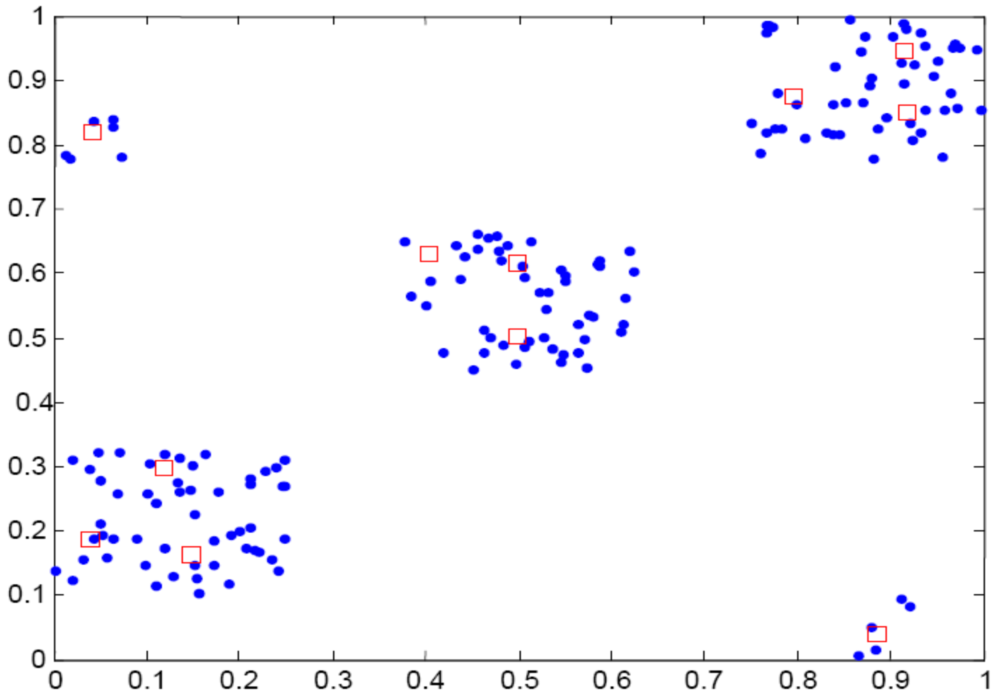


Figure 7.4 Final Cluster centers by obtained RPCL for artificial data

7.2 NeuroFuzzy Classifier1 (NFC1)

The proposed NeuroFuzzy classifier1 (NFC1) 6 layer network as shown in Fig 7.5. Without loss of generally, we consider a multi input multi class fuzzy classifier which consist of *L* rules for *C* class problem

$$R_k \quad : IF \;\; x_1 \; is \; A_{k1} \;\; AND \; .... AND \; x_m \; is \; A_{km} \quad THEN \;\; y = y_k \qquad (7.13)$$

Where R_k is the *k*th rule, A_{ij} is the fuzzy sets defined as Gaussian function. Each Layer operation in NFC1 is defined as;

Layer1 : Input layer. Each node represents an input. No operation is performed in this layer.

Layer2 : Fuzzification Layer. Each node arranged into L groups, each group representing the if-part of a fuzzy rule.

$$O_{ij}^{l2} = A_{ij} = e^{\frac{-\left(x_{j-}c_{ij}\right)^2}{\sigma_{ij}{}^2}} \qquad (7.14)$$

Where c_{ij} and σ_{ij} are centers and spread of the membership functions for input x_j.

Layer3 : Fuzzy Inference Layer. Fuzzy AND operation is performed in this layer. AND operator generally represented by a t-norm that is usually expressed as a product operator.

$$O_s^{l3} = \prod_{j \in Class_c}^{m} O_{sj}^{l2}, \qquad s=1,2,\ldots\ldots\ldots,L, \qquad (7.15)$$

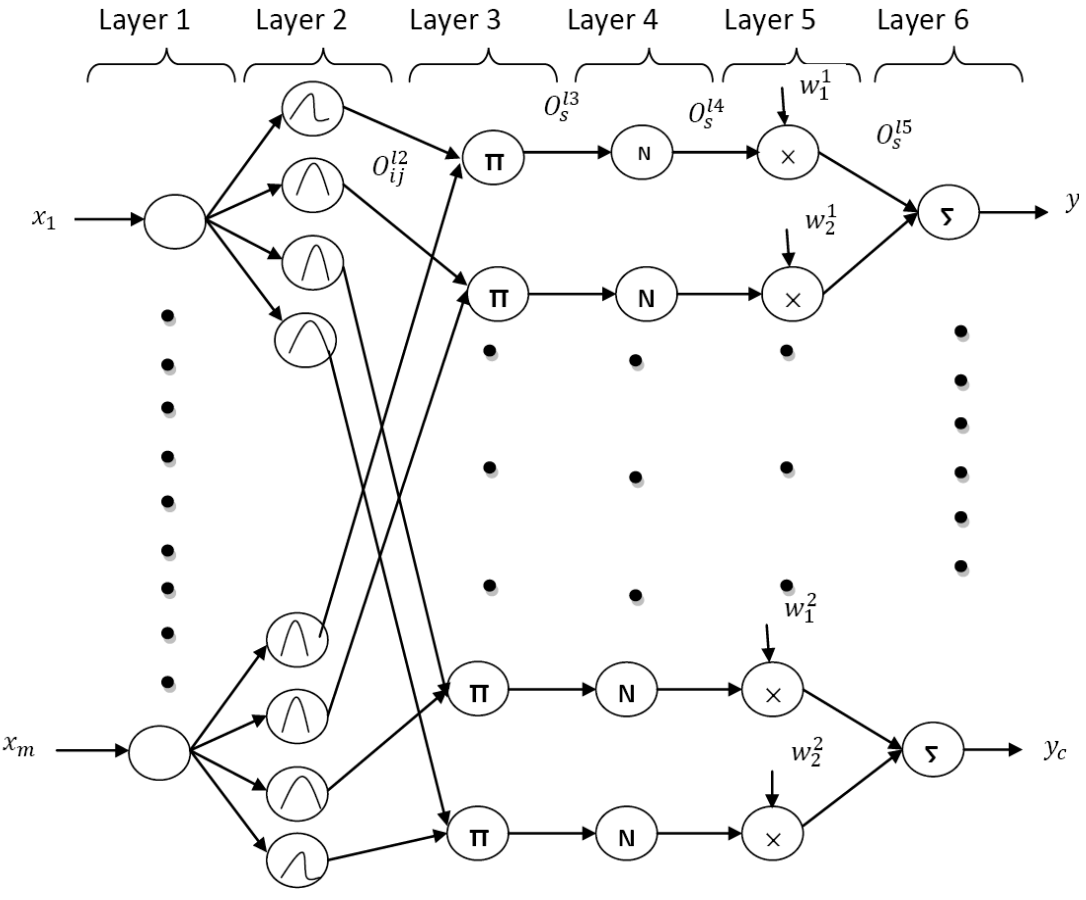


Figure 7.5 Proposed NFC1 structure

Layer4 : Normalization Layer.

$$O_s^{l4} = \frac{O_s^{l3}}{\sum_{k \in Class_c}^{n_c} O_k^{l3}} \qquad s=1,2,\ldots\ldots\ldots,L, \qquad (7.16)$$

Layer5:Rule Weighting Layer. Winning occurrences in Rival Penalized Competitive Learning bases structural identification phase are used as rule weighting. Weighted rule outputs are obtained as;

$$O_s^{l5} = w_s \times O_s^{l3} \qquad s=1,2,\ldots\ldots\ldots,L, \tag{7.17}$$

Layer6: Rule Consequent Layer. Rule outputs for Class c are summed and computed the output value $\overline{y_c}$;

$$\overline{y_c} = \sum_{k\in Class_c}^{n_c} O_k^{l5} \qquad c=1,2,\ldots\ldots C \tag{7.18}$$

Error function can be defined as;

$$E_c = \frac{1}{2}\sum_{p=1}^{P}(y_c - \overline{y_c})^2 \tag{7.19}$$

Where *p* is the current pattern, y_c is the actual output for class *c*, $\overline{y_c}$ is the computed output for class *c*.

For minimizing the error,

$$\frac{\partial E_c}{\partial \overline{y_c}} = -(y_c - \overline{y_c})^2 = error \tag{7.20}$$

Change of $\overline{y_c}$ with respect to the O_s^{l5}, propagation Layer6 to Layer5;

$$\frac{\partial \overline{y_c}}{\partial O_s^{l5}} = 1 \tag{7.21}$$

Change of $O_s^{l5}$with respect to the O_s^{l4}, propagation Layer5 to Layer4;

$$\frac{\partial O_s^{l5}}{\partial O_s^{l4}} = w_s \tag{7.22}$$

Change of $O_s^{l4}$with respect to the O_s^{l3}, propagation Layer4 to Layer3;

$$\frac{\partial O_s^{l4}}{\partial O_s^{l3}} = \frac{\sum \ O_k^{l3} - O_s^{l3}}{(\sum \ O_k^{l3})^2} = \frac{1}{\sum \ O_k^{l3}}\left(1 - \frac{O_s^{l3}}{\sum \ O_k^{l3}}\right) = \frac{1 - O_s^{l4}}{\sum \ O_k^{l3}} \tag{7.23}$$

Change of $O_s^{l2}$with respect to the O_{ij}^{l2}, propagation Layer3 to Layer2;

$$\frac{\partial O_s^{l3}}{\partial O_{ij}^{l2}} = \prod_{k \neq j} O_{ik}^{l2} \tag{7.24}$$

Change of $O_s^{l2}$with respect to the c_{ij}, propagation Layer2 to Layer1;

$$\frac{\partial O_{ij}^{l2}}{\partial c_{ij}} = 2\frac{(x_j - c_{ij})}{{\sigma_{ij}}^2} e^{\frac{-\left(x_{j-} c_{ij}\right)^2}{{\sigma_{ij}}^2}} = 2\frac{(x_j - c_{ij})}{{\sigma_{ij}}^2} O_{ij}^{l2} \tag{7.25}$$

Change of $O_s^{l2}$with respect to the σ_{ij}, propagation Layer2 to Layer1;

$$\frac{\partial O_{ij}^{l2}}{\partial \sigma_{ij}} = 2\frac{(x_j - c_{ij})^2}{{\sigma_{ij}}^3} e^{\frac{-\left(x_{j-} c_{ij}\right)^2}{{\sigma_{ij}}^2}} = 2\frac{(x_j - c_{ij})^2}{{\sigma_{ij}}^3} O_{ij}^{l2} \tag{7.26}$$

Gradient vector for rule weights;

$$\nabla w_s = \frac{\partial E_c}{\partial w} = \frac{\partial E_c}{\partial \overline{y_c}} \times \frac{\partial \overline{y_c}}{\partial O_s^{l5}} \times \frac{\partial O_s^{l5}}{\partial w_s} = error \times w_s \tag{7.27}$$

Gradient vector for membership function centers;

$$\nabla c_{ij} = \frac{\partial E_c}{\partial c_{ij}} = \frac{\partial E_c}{\partial \overline{y_c}} \times \frac{\partial \overline{y_c}}{\partial O_s^{l5}} \times \frac{\partial O_s^{l5}}{\partial O_s^{l4}} \times \frac{\partial O_s^{l4}}{\partial O_s^{l3}} \times \frac{\partial O_s^{l3}}{\partial O_{ij}^{l2}} \times \frac{\partial O_{ij}^{l2}}{\partial c_{ij}} \tag{7.28}$$

$$= error \times w_s \times \frac{1 - O_s^{l4}}{\sum \ O_k^{l3}} \times \prod_{k \neq j} O_{ik}^{l2} \times 2\frac{(x_j - c_{ij})}{{\sigma_{ij}}^2} O_{ij}^{l2} \tag{7.29}$$

Gradient vector for membership function spread ;

$$\nabla \sigma_{ij} = \frac{\partial E_c}{\partial \sigma_{ij}} = \frac{\partial E_c}{\partial \overline{y_c}} \times \frac{\partial \overline{y_c}}{\partial O_s^{l5}} \times \frac{\partial O_s^{l5}}{\partial O_s^{l4}} \times \frac{\partial O_s^{l4}}{\partial O_s^{l3}} \times \frac{\partial O_s^{l3}}{\partial O_{ij}^{l2}} \times \frac{\partial O_{ij}^{l2}}{\partial \sigma_{ij}} \tag{7.30}$$

$$= error \times w_s \times \frac{1-O_s^{l4}}{\sum \; O_k^{l3}} \times \prod_{k \neq j} O_{ik}^{l2} \times 2\,\frac{(x_j-c_{ij})^2}{{\sigma_{ij}}^3}\,O_{ij}^{l2} \qquad (7.31)$$

Finally new value of rule weights, membership function centers and membership function spreads are updated as;

$$w_s(t+1) = w(t) + \varphi_w \nabla w_s \qquad (7.32)$$

$$c_{ij}(t+1) = c_{ij}(t) + \varphi_c \nabla c_{ij} \qquad (7.33)$$

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \varphi_\sigma \nabla \sigma_{ij} \qquad (7.34)$$

,where,φ_w, φ_c and φ_σ are learning coefficients for of rule weights, membership function centers and membership function spread respectively.

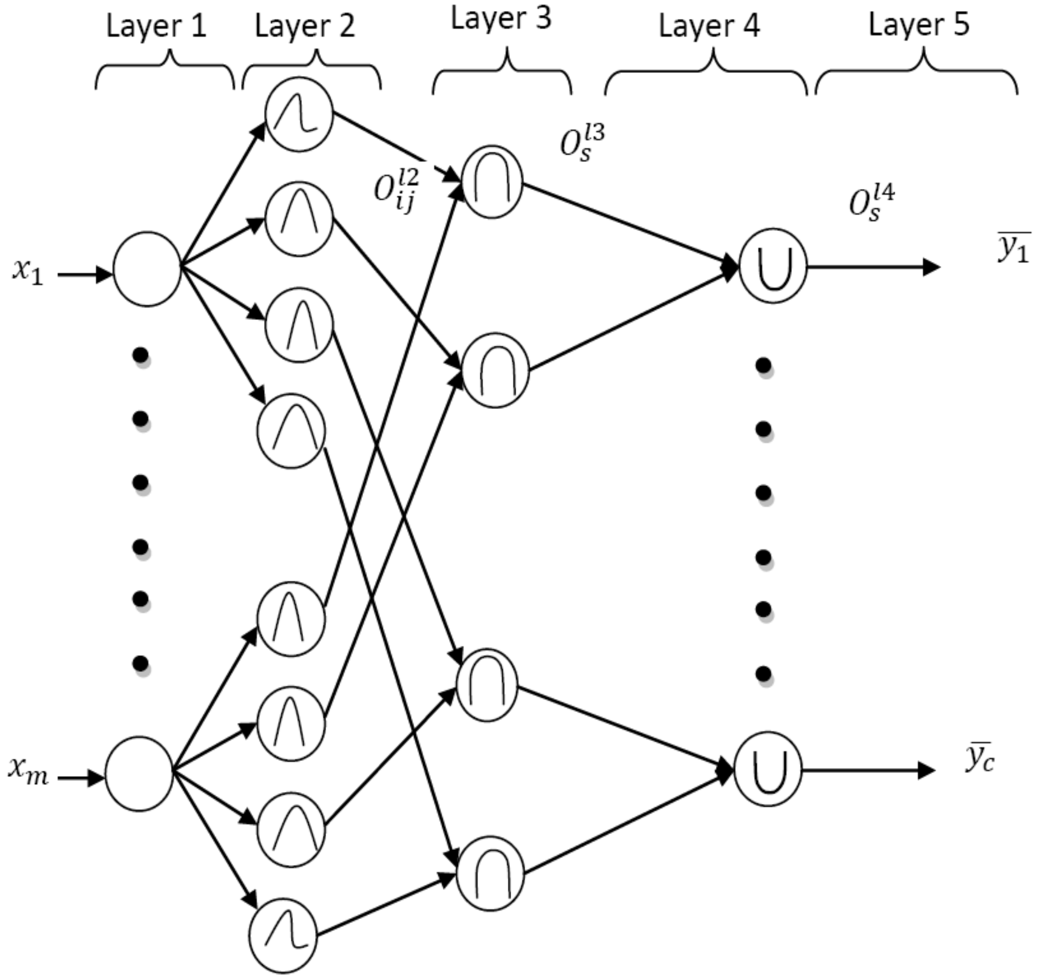


Figure 7.6 Proposed NFC2 structure

7.3 NeuroFuzzy Classifier2 (NFC2)

The proposed NeuroFuzzy Classifier2 (NFC2) is 5 layers network as shown in Fig. 7.6. In fact for *C* class problem, the NFC2 consists of C distinct classifiers. As described in Equation 7.13, rule base of the NFC2 is similar to the NFC1. Structure of the NFC2 is a simple version of the NFC1 in order to develop a new back propagation algorithm in parameter tuning phase. A dynamic learning strategy is also adopted in order to delete unnecessary nodes and add new nodes for improving classifier performance.

The training of the NFC2 are four steps as not its counterparts that generally training of NeuroFuzzy System consist of two steps as structural learning and parameters learning. Construction steps of the NFC2 as follows;

Step1 ; Construct initial NeuroFuzzy Classifiers with founded cluster centers by RPCL

Step2 ; Use back propagation algorithm in order to improve the classifier performance

Step3 ; Analyze the classifier by performing;

- Check the errors; if error is higher than a threshold add new neurons which centers is current input sample.
- Check the rules; delete rules whose firing counts are below some threshold.
- Check the membership function; delete nodes whose Euclidian distances are lower than a threshold.

Step4 ; Use back propagation to get final classifiers

Each Layer operation in the NFC2 is defined as;

Layer1: Input layer. Each node represents an input. No operation is performed in this layer.

Layer2: Fuzzification Layer. Each node arranged into L groups, each group representing the if-part of a fuzzy rule.

$$O_{ij}^{l2} = A_{ij} = e^{\frac{-\left(x_{j-}c_{ij}\right)^2}{\sigma_{ij}{}^2}} \tag{7.35}$$

Where c_{ij} and σ_{ij} are centers and spread of the membership functions for input x_j.

Layer3: Fuzzy Inference Layer. Fuzzy AND operation is performed in this layer.MIN operator is used as AND in the NFC2. Rule firing strength of each rule is computed as follows ;

$$O_s^{l3} = \min_{j \in Class_c} O_{sj}^{l2}, \qquad s=1,2,\dots\dots\dots,L, \tag{7.36}$$

Layer4: Rule Consequent Layer.

$$O_c^{l4} = \max_{j \in Class_c} O_j^{l3} \qquad c=1,2,\dots\dots\dots,C, \tag{7.37}$$

Layer5: Final Output Layer.

$$\overline{y_c} = O_c^{l4} \qquad c=1,2,\dots\dots C \tag{7.38}$$

In parameter tuning phase of proposed the NFC2, rival penalization mechanism is adopted in to the gradient descent algorithm. In order to achieve this following cost function defined as;

$$E_c = \frac{1}{2}\sum_{p=1}^{P} y_c(1 - \overline{y_c} + \overline{y_{rival}})^2 \tag{7.39}$$

For incremental learning case Equation 40 can be defined as;

$$E_c^p = \frac{1}{2} y_c^p (1 - \overline{y_c^p} + \overline{y_{rival}^p})^2 \tag{7.40}$$

Change of Error with respect to the classifier C, $\overline{y_c^p}$ is computed as;

$$\frac{\partial E_c}{\partial \overline{y_c}} = -y_c^p\left(1 - \overline{y_c^p} + \overline{y_{rival}^p}\right) = error_c \tag{7.41}$$

Rival error change is defined as negative of winner error, $error_c$;

$$error_{rival} = -error_c \tag{7.42}$$

Change of $\overline{y_c}$ with respect to the O_c^{l4}, propagation Layer5 to Layer4;

$$\frac{\partial \overline{y_c}}{\partial O_c^{l4}} = 1 \tag{7.43}$$

Change of $O_c^{l4}$with respect to the O_s^{l3}, propagation Layer4 to Layer3;

$$\frac{\partial O_c^{l4}}{\partial O_s^{l3}} = \begin{cases} 1, & O_c^{l4} = O_s^{l3} \\ 0, & otherwise \end{cases} = \delta_{max} \tag{7.44}$$

Change of $O_s^{l4}$with respect to the O_s^{l3}, propagation Layer3 to Layer2;

$$\frac{\partial O_s^{l3}}{\partial O_{ij}^{l2}} = \begin{cases} 1, & O_s^{l3} = O_{ij}^{l2} \\ 0, & otherwise \end{cases} = \delta_{min} \tag{7.45}$$

Change of $O_{ij}^{l2}$with respect to the c_{ij}, propagation Layer2 to Layer1;

$$\frac{\partial O_{ij}^{l2}}{\partial c_{ij}} = 2\frac{(x_j - c_{ij})}{{\sigma_{ij}}^2} e^{\frac{-\left(x_j - c_{ij}\right)^2}{{\sigma_{ij}}^2}} = 2\frac{(x_j - c_{ij})}{{\sigma_{ij}}^2} O_{ij}^{l2} \tag{7.46}$$

Change of $O_s^{l2}$with respect to the σ_{ij}, propagation Layer2 to Layer1;

$$\frac{\partial O_{ij}^{l2}}{\partial \sigma_{ij}} = 2\frac{(x_j - c_{ij})^2}{{\sigma_{ij}}^3} e^{\frac{-\left(x_j - c_{ij}\right)^2}{{\sigma_{ij}}^2}} = 2\frac{(x_j - c_{ij})^2}{{\sigma_{ij}}^3} O_{ij}^{l2} \tag{7.47}$$

Gradient vector for winner membership function centers;

$$\nabla {c_{ij}}^{\text{winner}} = \frac{\partial E_c}{\partial c_{ij}} = \frac{\partial E_c}{\partial \overline{y_c}} \times \frac{\partial \overline{y_c}}{\partial O_c^{l4}} \times \frac{\partial O_c^{l4}}{\partial O_s^{l3}} \times \frac{\partial O_s^{l3}}{\partial O_{ij}^{l2}} \times \frac{\partial O_{ij}^{l2}}{\partial c_{ij}} \tag{7.48}$$

$$= error_{winner} \times \delta_{max} \times \delta_{min} \times 2\,\frac{(x_j - c_{ij})}{{\sigma_{ij}}^2}\,O_{ij}^{l2} \tag{7.49}$$

Gradient vector for winner membership function spread ;

$$\nabla{\sigma_{ij}}^{\text{winner}} = \frac{\partial E_c}{\partial \sigma_{ij}} = \frac{\partial E_c}{\partial \overline{y_c}} \times \frac{\partial \overline{y_c}}{\partial O_c^{l4}} \times \frac{\partial O_c^{l4}}{\partial O_s^{l3}} \times \frac{\partial O_s^{l3}}{\partial O_{ij}^{l2}} \times \frac{\partial O_{ij}^{l2}}{\partial \sigma_{ij}} \tag{7.50}$$

$$= error_{winner} \times \delta_{max} \times \delta_{min} \times 2\,\frac{(x_j - c_{ij})^2}{{\sigma_{ij}}^3}\,O_{ij}^{l2} \tag{7.51}$$

Gradient vector for rival membership function centers;

$$\nabla{c_{ij}}^{rival} = \frac{\partial E_c}{\partial c_{ij}} = \frac{\partial E_c}{\partial \overline{y_c}} \times \frac{\partial \overline{y_c}}{\partial O_c^{l4}} \times \frac{\partial O_c^{l4}}{\partial O_s^{l3}} \times \frac{\partial O_s^{l3}}{\partial O_{ij}^{l2}} \times \frac{\partial O_{ij}^{l2}}{\partial c_{ij}} \tag{7.52}$$

$$= error_{rival} \times \delta_{max} \times \delta_{min} \times 2\,\frac{(x_j - c_{ij})}{{\sigma_{ij}}^2}\,O_{ij}^{l2} \tag{7.53}$$

Gradient vector for winner membership function spread ;

$$\nabla{\sigma_{ij}}^{rival} = \frac{\partial E_c}{\partial \sigma_{ij}} = \frac{\partial E_c}{\partial \overline{y_c}} \times \frac{\partial \overline{y_c}}{\partial O_c^{l4}} \times \frac{\partial O_c^{l4}}{\partial O_s^{l3}} \times \frac{\partial O_s^{l3}}{\partial O_{ij}^{l2}} \times \frac{\partial O_{ij}^{l2}}{\partial \sigma_{ij}} \tag{7.54}$$

$$= error_{rival} \times \delta_{max} \times \delta_{min} \times 2\,\frac{(x_j - c_{ij})^2}{{\sigma_{ij}}^3}\,O_{ij}^{l2} \tag{7.55}$$

Finally new value of membership function centers and membership function spread are updated as ;

$${c_{ij}}^{winner}(t+1) = {c_{ij}}^{winner}(t) + {\varphi_c}^{winner}\nabla{c_{ij}}^{\text{winner}} \tag{7.56}$$

$${\sigma_{ij}}^{winner}(t+1) = {\sigma_{ij}}^{winner}(t) + {\varphi_\sigma}^{winner}\nabla{\sigma_{ij}}^{\text{winner}} \tag{7.57}$$

$${c_{ij}}^{rival}(t+1) = {c_{ij}}^{rival}(t) + {\varphi_c}^{rival}\nabla{c_{ij}}^{rival} \tag{7.58}$$

$${\sigma_{ij}}^{rival}(t+1) = {\sigma_{ij}}^{rival}(t) + {\varphi_\sigma}^{rival}\nabla{\sigma_{ij}}^{rival} \tag{7.59}$$

7.4 Case Studies

The proposed Classifiers the NFC1 and the NFC2 are performed on two real world problems. At initializing phase, all attributes and class outputs are normalized into the unit interval [0 1]. Classifier validations are obtained by k-fold cross validation procedure [135]. Classifier performance is measured on both testing and training phase as;

$$Accurracy = 100 \times \frac{\#\, correctly\ classified\ sample}{\#\, sample} \tag{7.60}$$

Root mean square error measurement is performed as;

$$RMSE = \sqrt{\ \sum_{m=1}^{M}(y_c^m - \overline{y_c^m} + \overline{y_{rival}^m})^2\ } \tag{7.61}$$

where *m* is the current pattern, *M* is the number of total patterns, y_c^m is actual output for *c*th classifier with *m*th pattern, $\overline{y_c^m}$ is predicted output for *c*th classifier with *m*th pattern and $\overline{y_{rival}^m}$ is the second winner classifier predicted output with *m*th pattern

Iris Dataset [134]

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other. The attributes are as follows:x_1 -sepal length in cm, x_2 -sepal width in cm, x_3 - petal length in cm, x_4 -petal width in cm. Three classes: *Iris Setosa, Iris Versicolou*r and *Iris Virginica.*

In the experiment, firstly iris classification task is performed by the NFC1. Figure 7.7 shows the dynamic behaviors of the NFC1 classifiers during the learning phase. According to 2 fold cross validation test, 75 patterns are used in training while remainder 75 for testing. As shown in Figure 7.7, RMSE gets lower while epoch numbers increases up to 30. However after epoch 30, no more improvement can be accomplished by gradient descent algorithm. When Classifier Error graph is analyzed, it can be easily observed that there is a problem with the NFC1 Classifier with pattern 63, 70. According to iris classification accuracy results for the NFC1 at Table7.1, it can be concluded that the NFC1 is successful in comparing to other soft computing tools whose performance were given in Chapter6 and Section 6.3.

Table 7.1 Classifiers Performance for iris classification problems

	Training Accuracy	Training std	Testing Accuracy	Training std	# rules
NFC1	96.67	0.94	96.67	0.94	5
NFC2	100	0	97.33	0.94	9

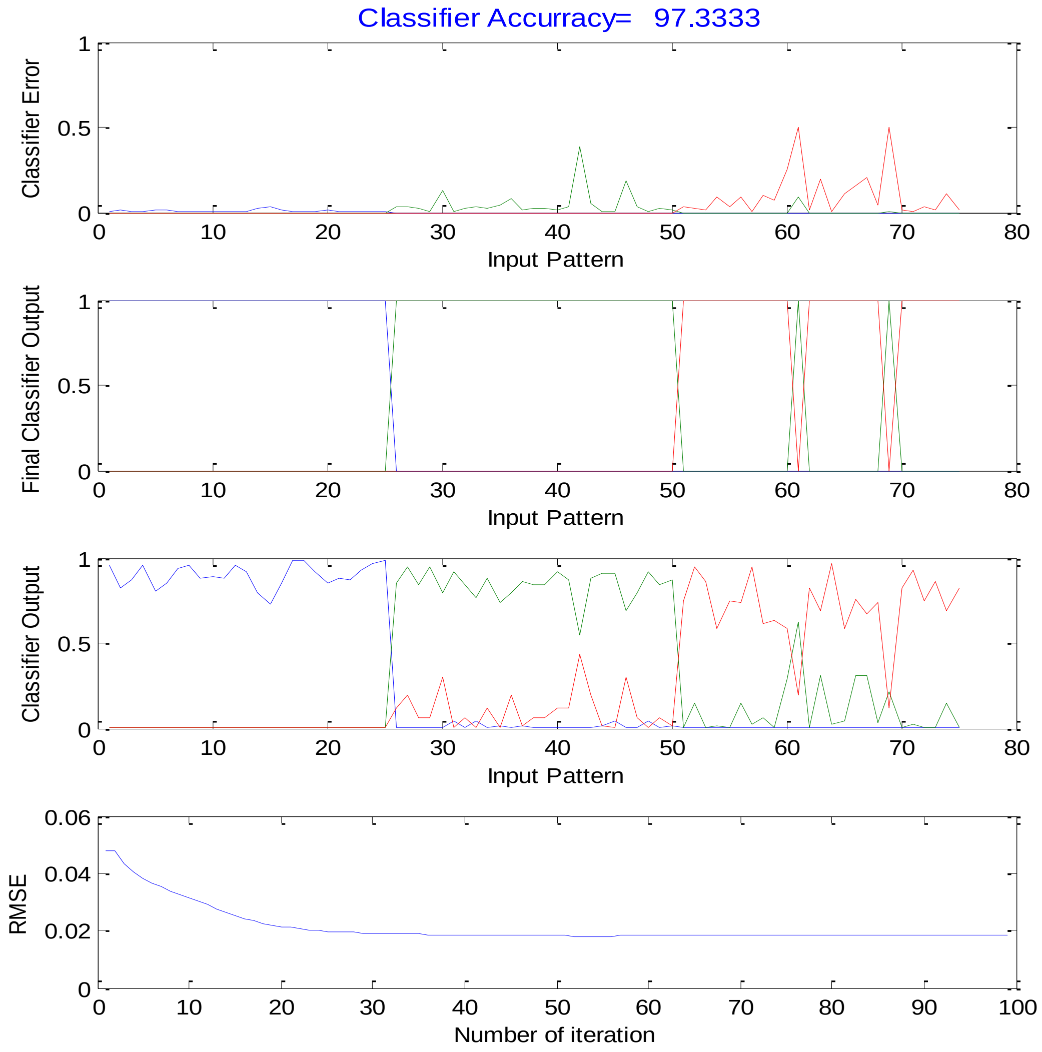


Figure 7.7 Classifiers behaviors at the end of Step2 for Iris Classification tasks with the NFC2

The NFC2 has different features as mentioned before that it has rule adaptation mechanism and uses a RPCL type GD training algorithm. As shown in Table7.1 classification accuracy for training phase is 100%, and 97.3 % in testing phase. These outcomes are proof of the NFC2 robustness. Fig.7.8 demonstrates the dynamic behavior of the NFC2 while training by RPCL type GD and rule adaptation mechanism. At the end of epoch 100, new neurons or rules are added to system according to error criteria, therefore it provide the NFC2 capable to learn all patterns during the learning phase.

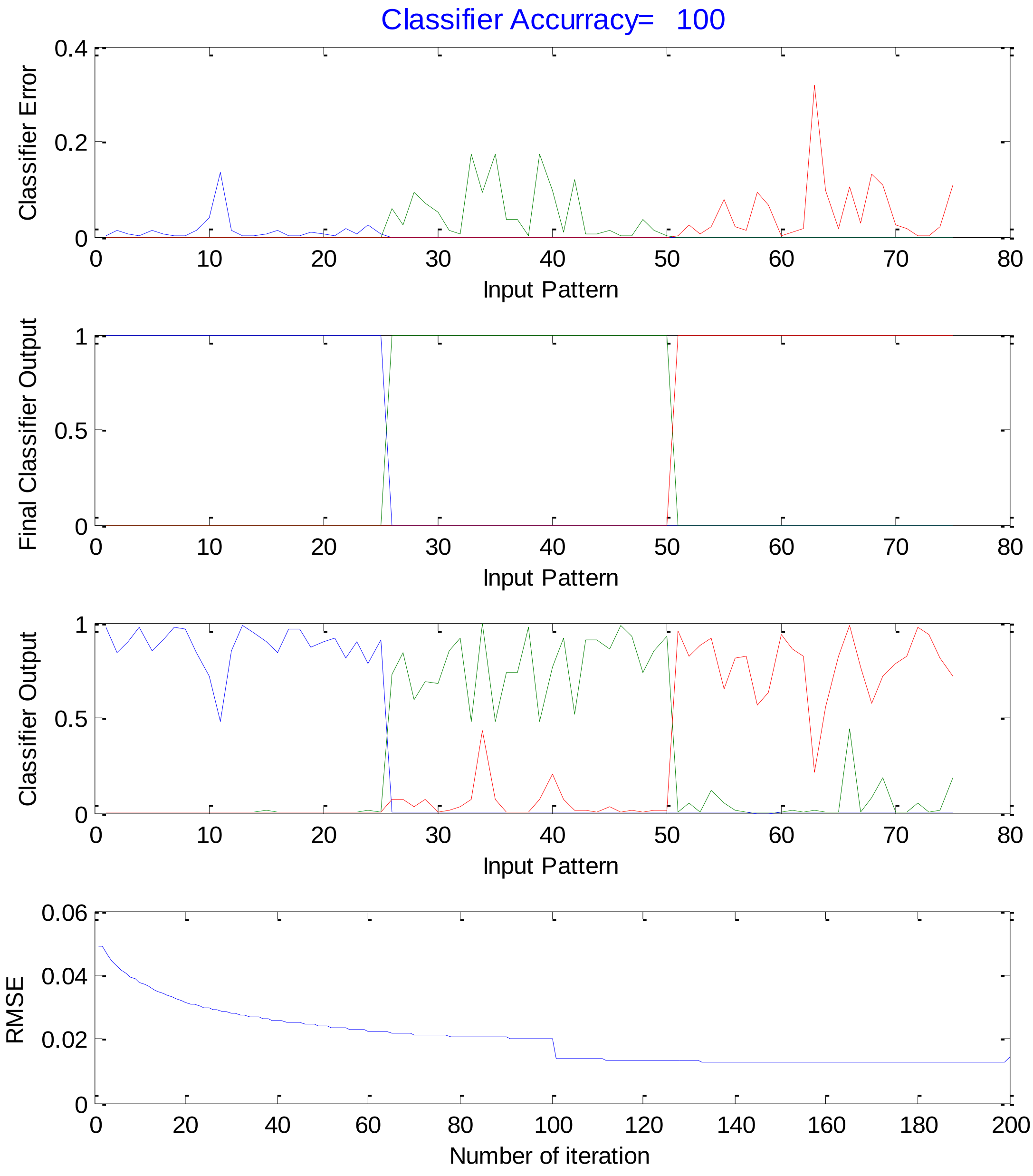


Figure 7.8 Classifiers behaviors at the end of Step4 for Iris Classification tasks with the NFC2

In Fig.7.9, obtained membership functions at the end of step2 for Iris Setosa the NFC2 classifiers are demonstrated. At the end of step3, rule adaption mechanism is performed that yields to delete one rule from the NFC2 structure. After re tuning by RPCL type GD training, final obtained membership functions for Iris Setosa the NFC2 are shown in Fig.7.10.

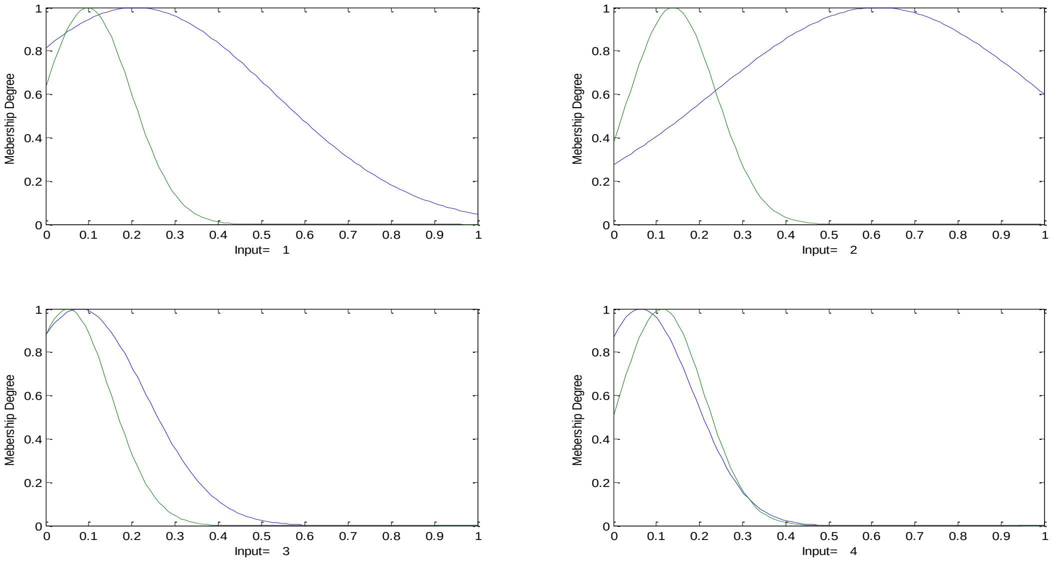


Figure 7.9 Tuned membership functions for *Iris Setosa* at the end of step2

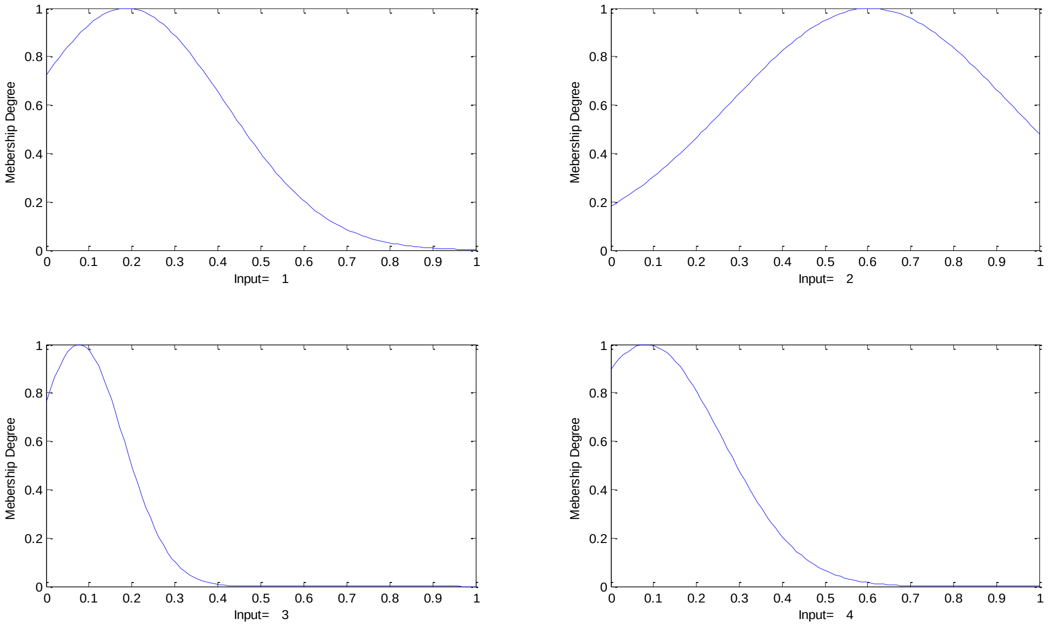


Figure 7.10 Tuned membership functions for *Iris Setosa* at the end of step4

Classification for Iris *Versicolour* results are shown in Fig.7.11, and 7.12 at the end of step2 and step4, respectively. At the end of 100 epoch that corresponds the end of step2, rule adaptation mechanism is performed which causes to add one extra rule to the NFC2.

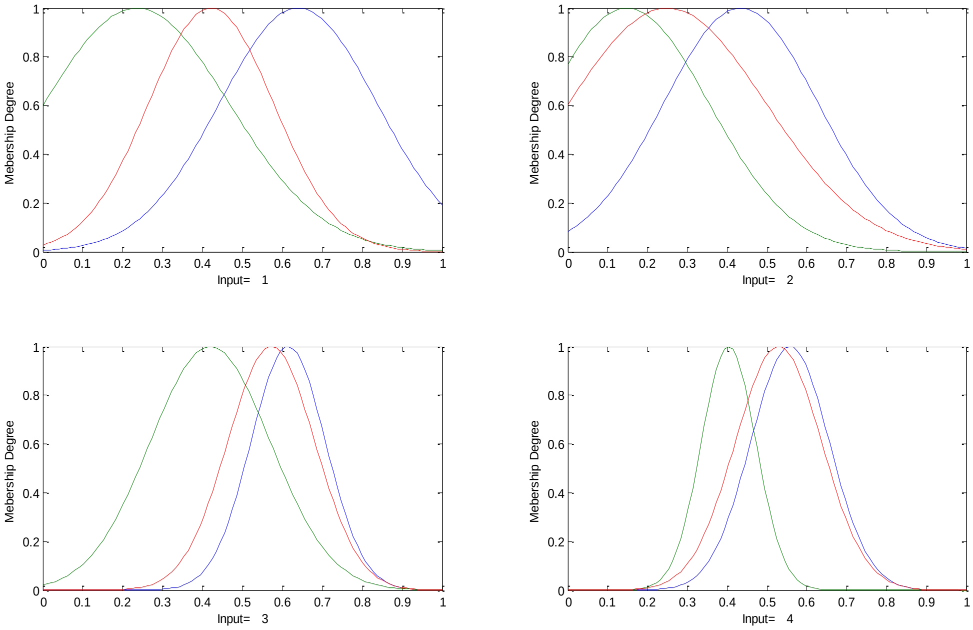


Figure 7.11 Tuned membership functions for *Iris Versicolour* at the end of step2

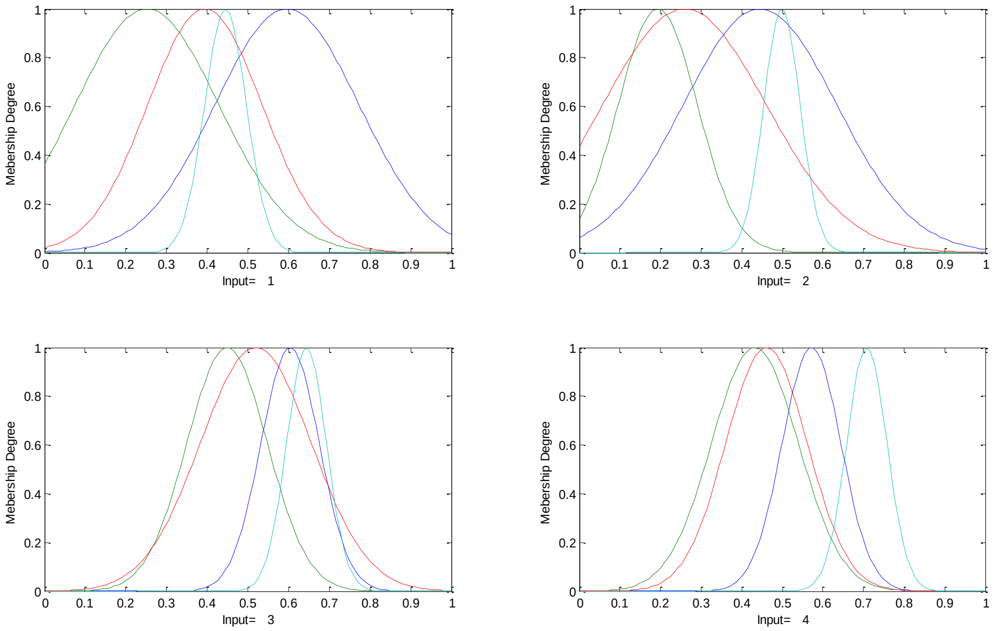


Figure 7.12 Tuned membership functions for *Iris Versicolour* at the end of step4

In *Iris Virginica* classification with the NFC2, the obtained membership functions at the end of step2 and step4 are shown in Fig.7.13 and 7.14, respectively. As in the Iris *Versicolour* classification one rule is added by rule adaptation mechanism at the end of step3.

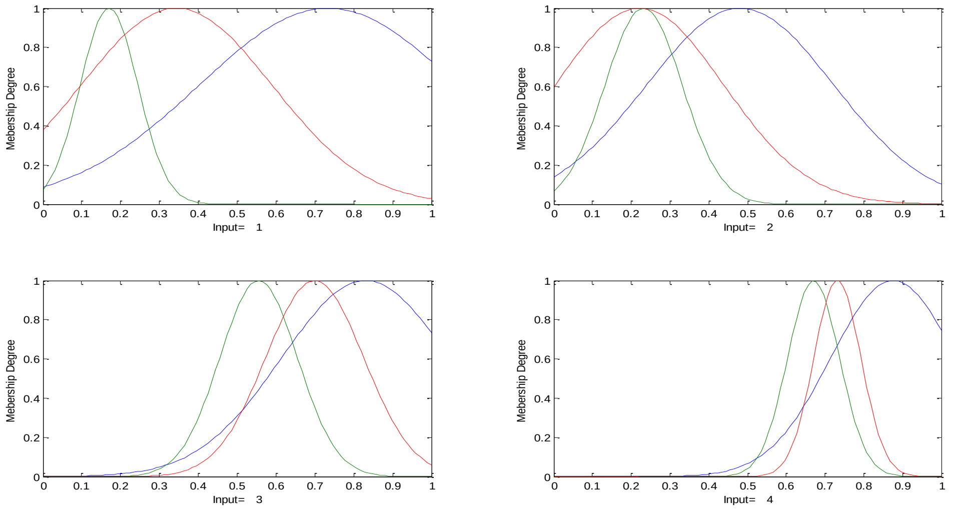


Figure 7.13 Tuned membership functions for *Iris Virginica* at the end of step2

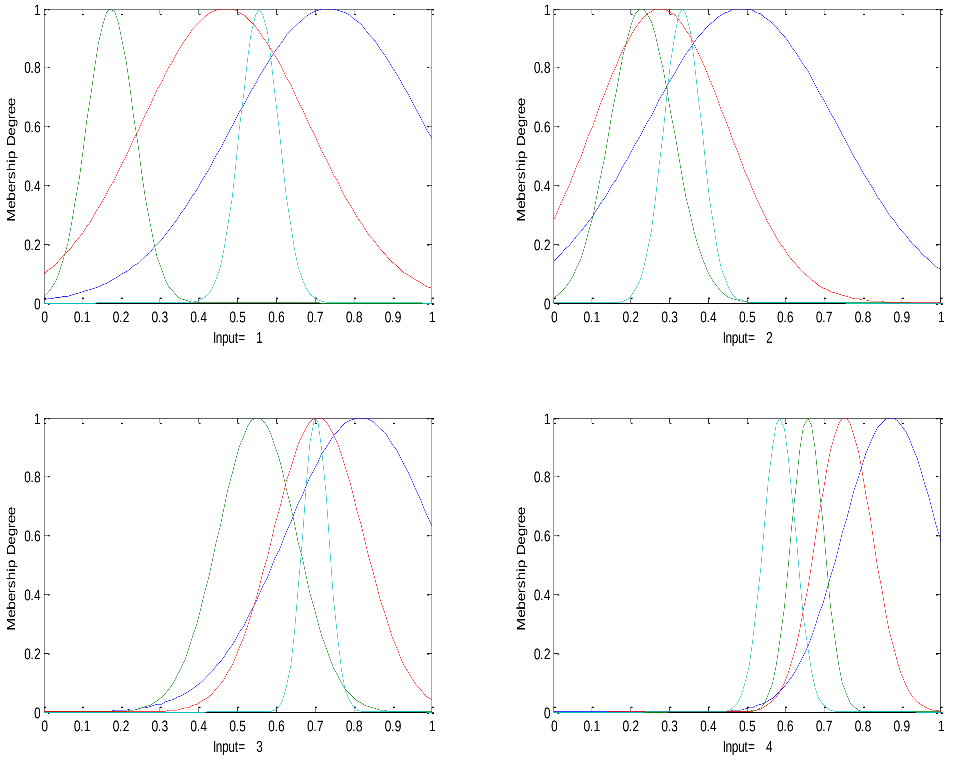


Figure 7.14 Tuned membership functions for *Iris Virginica* at the end of step4

Wisconsin Breast Cancer Dataset [134]

The Wisconsin Breast Cancer Dataset was compiled by Medical College of Wisconsin and has been widely used to test the functionality of many classification and rule extraction methods. It is composed by nine numerical attributes, describing nine different blood ingredients and two different output classes, describing the nature of the cancer, either malign or benign. The original dataset is composed of 699 observations of which 16 were deliberately excluded due to incomplete descriptions of all nine dimensions. From the remaining 683 patterns, 444 belong to the benign class, 239 to the malign class while 252 belong to the intersection of the two classes. It is consequently a fairly dimensional dataset where the two classes coincide considerably. Nine features of Wisconsin Breast Cancer data set contains ;Clump Thickness (UC), Uniformity of Cell Size (UC), Uniformity of Cell Shape (UC), Marginal Adhesion (MA), Single Epithelial Cell Size (SE), Bare Nuclei (BN), Bland Chromatin (BC), Normal Nucleoli (NN), Mitoses (Mit.) ranging from 1 to 10. Output of data set either 0 for benign or 1 for malignant. Therefore it is a two class problems.

The NFC1 and the NFC2 performance are shown in Table 7.2. It is clear that the NFC2 has better performance than the NFC1 in both training and testing phase. However the NFC2 needs more rule to perform classification task because rule adaptation mechanism produces extra rule while RMSE couldn’t be improved by GD training. Although, training accuracy of the NFC2 beats the each type of SASCFC which shown in Table 6.8 in Chapter 6, testing accuracy of the NFC2 is a bit behind its counterparts.

Table 7.2 Classifiers performance for Wisconsin Breast Cancers classification

	Training Accuracy	Training std	Testing Accuracy	Training std	# rules
NFC1	96.67	0.94	94.2	2.3	5
NFC2	98.81285	2.89859	95.30239	2.49026	41

7.5 Summary of Results and Conclusion

Rival penalized competitive based clustering for construction initial structure of NeuroFuzzy classifiers namely NFC1 and NFC2, are proposed in this chapter. In the NFC1, the found cluster centers and spread are used in construction of rule base and winning occurrence numbers are used as rule weighting coefficients. After initial structure obtained, a gradient descent base optimization is performed in parameter tuning phase. The NFC2 structure is obtained a similar way with the NFC1. Main difference of the NFC2 that it consists of fewer layers than the NFC1 because the NFC2 has a winner takes all mechanism. This mechanism is also very similar with Rival penalized competitive learning principle. So in parameter tuning phase, rival penalization type gradient descent is performed for the NFC2. The other difference of the NFC2 is that it is a dynamic network that during training phase its structure changes according to system output error and similarity measurements.

CHAPTER 8

CONCLUSION

8.1 Summary and Conclusions

The main steps for construction fuzzy or a NeuroFuzzy based classifiers, controllers, models and decision making systems contain followings;

i. Selection optimum input feature subset via removing redundant and noisy features from input feature space

ii. Obtaining the number of the membership functions for each input.

iii. Obtaining the parameters of the membership functions

iv. Constructing rule base

v. Adjustment of the output threshold value which describes the boundaries of each class.

In this thesis, systematic approaches are developed to accomplish above mentioned steps. Firstly, common clustering algorithms as SC, K-means and FCM which are widely used in constructing initial structure of fuzzy and NeuroFuzzy systems are introduced and performed as a standalone classifiers for cervical cancer diagnosis. The number of rules and the rule structure in a NeuroFuzzy system plays an important role both system performance and training time. It is reviewed most common Artificial Intelligence tools such as TS-FIS, ANFIS, RBF-NN and FFN and their application to pap-smear classification task for diagnosis of cervical cancer. Effects of the variations of number of cluster on true classification ratio are demonstrated for TS-FIS, ANFIS based classifiers. The effects of the number of neurons in hidden layers on true classification ratio for FFN and effect of spreads of radial basis neurons for RBF are figured out. Feature selection also plays important role on system performance because of irrelevant features and redundant features cause misclassification and also cause increasing of the total computing time. By using correlation based feature selection, feature ranking algorithm and projecting of input space to a new space by using principle component analysis, size of the input space are reduced. According to new input space which derived by integration of features selection algorithms to the classifier, the classification results of four classifiers; TS-FIS, ANFIS, RBF-NN and FFN, are shown and compared. The proposed simple integrated correlation based future selection yields more acquired results in comparing to the other feature selection algorithms. Although the classification accuracy of classifiers with feature selection is not being improved but the computational time is decreased. The computational time versus classification accuracies for TS-FIS, ANFIS, RBF-NN and FFN classifiers with various features space sizes are demonstrated. Main contribution of this thesis is that we developed mainly two systematic ways in order to obtain fuzzy and NeuroFuzzy based classifier.

8.1.1 Simulated Annealing Subtractive Clustering Based Fuzzy Classifier

A modified version of simulated annealing optimization algorithm is developed in order to obtain proper fuzzy classifier. The modification on classical simulated annealing provides to find solution with lower iteration. The optimization deals with constructing fuzzy if-then rule base, finding output threshold value and selecting most proper inputs for classification task. A hybrid feature selection that combines filter and wrapper types feature selection techniques is also developed.

In order to compare the effects of feature selection, rule base and output threshold optimization four SASCFC are proposed and test with 12 benchmark classification problems. Experimental results show that, although satisfactory improvement on the accuracy performance is not obtained by wrapper approach, it enables to reduce the classifier complexity which directly influences on total rule size and consumption time of the classifier. Hybrid feature selection approach improves both performance and complexity of classifier and experimental results show the best accuracies are achieved by this approach. SASCFC-Type2 is also compared with 6 well known classification tools for 7 classification problems that it has best testing accuracy for 4 of 7 classification tasks.

8.1.2 Rival Penalized Competitive Learning Based NeuroFuzzy Classifier

Rival Penalized Competitive Learning strategy is derived for obtain neuro fuzzy based classifiers namely NFC1 and NFC2. Initial structures of both the NFC1 and the NFC2 are constructed by RPCL type clustering technique. Although gradient descent learning is preformed in parameter learning phase for both two classifiers, a modified version of gradient descent algorithm is derived for the NFC2 which has a winner takes all mechanism structure. This mechanism is also very similar with Rival penalized competitive learning principle. So in parameter tuning phase, rival penalization type gradient descent is performed for the NFC2. The other difference of the NFC2 is that it is a dynamic network that during training phase its structure changes according to system output error and similarity measurements. Both Classifiers are tested and compared with two benchmark problems with classification accuracy, standard deviation and classifier complexity. Although the NFC2 needs more if-then rule, it has better performance that it is more accurate in testing and training phase.

8.2 Future Works and Recommendations

Following topics will be our future studies;

i. Although Simulated Annealing is a powerful optimization tool for combinatory optimization tasks, it needs too much training time. Especially temperature cooling mechanism in conventional Simulated Annealing algorithm is either a linear or a logarithmic function. A fuzzy control system can be used for controlling the cooling schedule that probably causes to decrease iteration number.

ii. Other optimization methods such as Genetic Algorithm, Hill Climbing, and Particle Swarm can be used to obtain optimum Fuzzy and NeuroFuzzy classifier.

REFERENCES

[1] Efe, O. M., Kaynak, O., Wilamowski, B. M. (2000). Stable Training of Computationally Intelligent System by Using Variable Structure System Techniques, *IEEE Transaction on Ind. Elec*., **47 (2)**, 487-496.

[2] Ranjan, R. , Awasthi, A. , Aggarawal, N. , Gulati, J. (2006). Applications of Fuzzy and Neuro-Fuzzy in Biomedical Health Sciences, *International Conference Electro information Technology*, *IEEE*., 60-65.

[3] Shitong, W., Duan, F., Min, X., Dewen , H. (2007). Advanced fuzzy cellular neural network: Application to CT liver images, *Artificial Intelligence in Medicine*, **39**, 65-77.

[4] Othman, M. F., Shan Yau, T. (2007). Neuro Fuzzy Classification and Detection Technique for Bioinformatics Problems, *IEEE Proceedings of the First Asia International Conference on Modelling & Simulation* (AMS'07), 375-380.

[5] Arasu, G. T. (2007). NeuroFuzzy Agent Programming for Processing of EEG/ECG/EMG, *IEEE International Conference on Information Technology* (ITNG'07), 937-938.

[6] Barisci, N., Hardalac, F. (2007). Application of an adaptive neuro-fuzzy inference system for classification of Behcet disease using the fast Fourier transform method. *Expert Systems*, **24**, 2, 123-130.

[7] Mahfouf, M., Abbod, M.F., Linkens D.A. (2001).A survey of fuzzy logic monitoring and control utilisation in medicine, *Artificial Intelligence in Medicine*, **21**, 27-42.

[8] Aqil, M., Kita, I., Yano, B., Nishiyama, A. (2007). A comparative study of artificial neural Networks and neuro-fuzzy in continuous modeling of the daily and hourly behaviour of runoff, *Journal of Hydrology*, **337**, 22–34.

[9] Gholipour, A., Araabi, B., Lucas, C. (2006). Predicting Chaotic Time Series Using Neural and Neurofuzzy Models: A Comparative Study, *Neural Processing Letters*, **24 (3)**, 217-239.

[10] Eker, İ.,Torun, Y. (2006). Fuzzy logic control to be as a conventional method, *Energy Conversion and Management*, **47 (4)**, 377-394.

[11] Babuska, R., Verbruggen, H. (2003). Neuro-Fuzzy methods for nonlinear system identification, *Annual Reviews in Control*, 73-85.

[12] Jang, J. S. R. and Sun, C. T. (1993). Functional equivalence between radial basis function networks and fuzzy inference systems, *IEEE Trans. NeuralNetworks*, **4**, 156–159.

[13] Wu, S., Chiang, H., Lin, H., Lee, T. (2005). Neural-network-based optimal fuzzy controller design for nonlinear systems, *Fuzzy Sets and Systems*, **154**, 182–207.

[14] Gao, Y., Meng, J. R. (2003). Modelling, Control, and stability analysis of non-linear system using generalized fuzzy neural networks, *International Journal of System Science*, **34 (6)**, 427-438.

[15] Vieira, J., Morgado, D. F., Alexandre, M. (2004). Artificial neural networks and neuro-fuzzy systems for modeling and controlling real systems: a comparative study, *Engineering Applications of Artificial Intelligence*, **17 (3)**, 265-273.

[16] Isermann, R. (2005). *Fault Diagnosis Systems. An Introduction from Fault Detection to Fault Tolerance*. Springer, Berlin.

[17] Korbicz, J., Kowal, M. (2007). Neuro-fuzzy networks and their application to fault detection of dynamical systems. *Engineering Applications of Artificial Intelligence*, **20 (5)**, 609-617.

[18] Zadeh, L. A. (1965). Fuzzy Sets. Information and Control, 8 , 338-353

[19] Sean, N. G., Thunshun, W. L. (2008). Medical data mining by fuzzy modeling with selected features, *Artificial Intelligence in Medicine*, **43**, 195-206.

[20] Mohamadi, H., Habibi, J., Abadeh, M. S., Sadi, H. (2008). Data mining with a simulated annealing based fuzzy classification system, *Pattern Recognition*, **41**, 1824 – 1833.

[21] Bezine, H., Derbel, N., Alimi, A. M. (2002). Fuzzy control of robot manipulators: some issues on design and rule base size reduction. *Eng Appl Artif. Intell.*, **15,** 401–416.

[22] Jang, R. J. (1993). ANFIS: Adaptive-Network-Based Fuzzy Inference System, *IEEE Transactions on Man, and Cybernetics*, **23 (3)**, 665-683.

[23] Jang, R. J., and Sun, C.T. (1995). Neuro-fuzzy modelling and control,*The Proc. of the IEEE*, 378-406.

[24] Hayashi, Y. and Buckley, J. J. (1994). Approximations between fuzzy expert systems and neural networks, *Int. J. Approx. Reas.*, **10**, 63–73.

[25] Zhang, Y., Kandel, A. (1998). Compensatory Neurofuzzy Systems, *IEEE Transactions on Neural Networks*, **9 (1)**, 83-105.

[26] Chakraborty, D., Pal, R. N. (2001). Integrated Feature Analysis And Fuzzy Rule-Based System Identification In A Neuro-Fuzzy Paradigm, *IEEE*

Transactıons On Systems, Man, And Cybernetıcs—Part B: Cybernetıcs, **31 (3),** 391-400.

[27] Azeem, M. F., Hanmandlu, M., Ahmad, N. (2000). Generalization of Adaptive Neuro-Fuzzy Inference Systems, *IEEE Transaction on Neural Networks,* **11 (6)**, 1332-1346.

[28] Azeem, M. F., Hanmandlu, M., Ahmad, N. (2003). Structure identification of Generalized Neuro-Fuzzy Inference Systems, *IEEE Transaction on Fuzzy Systems,* **11 (5)**, 666-681.

[29] Azeem, M. F., Hanmandlu, M., Ahmad N. (2005). Parameter Determination for a Generalized Fuzzy Model, *Soft Computing,* **9**, 211-221.

[30] Lee George, C. S., Wang, J. S. (2005). Self Adaptive Neuro Fuzzy Systems ;Structure and Learning, *Proc. Of the Int. Conf. On Intelligent Robots and Systems*, 3861-3866.

[31] Javonovic, B. B., Relijin, S. I. (2004). Modified ANFIS Architecture-Improving Efficienty of ANFIS Technique, *Proc. On Neural Network In Electrical Eng., NEUREL*, 215-220.

[32] Figueiredo, M., Ballini, R., Andrade, M., Gomide, F. (2004). Learning Algorithms For A Class Of Neurofuzzy Network And Application, *IEEE Transactions On Systems, Man, And Cybernetics—Part C: Applications And Reviews,* **34 (3)**, 293 -301.

[33] Treesatayapun, C., Uatrongit, S. (2005). Adaptive controller with fuzzy rules emulated structure and identification, *Engineering Applications of Artificial Intelligence,* **18**, 603-615.

[34] Ouyang, C., Lee, W., Lee, S. (2005). A TSK-Type Neurofuzzy Network Approach to System Modeling Problems, *IEEE Transactions On Systems, Man, And Cybernetics—Part B: Cybernetics,* **35 (4)**, 751-767.

[35] Yu, W., Marco, A. (2005). System identification using hierarchical fuzzy neural networks with stable learning algorithms, *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005 Seville, Spain*, 171-183.

[36] Zanchettin, C., Minku, F. L., Ludermir T. (2005). Design of Experiments in Neuro-Fuzzy Systems, In *Proceeding of 5th Int. Conf. On Hybrid Intelligent System*, 218-226.

[37] Wangm, L.X., Mendel, J.M. (1992). Generating fuzzy rules by learning from examples, *IEEE Trans. Syst. Man Cybern.,* **22**, 1414–1427.

[38] Ishibuchi, H., Nozaki, K., Tanaka, H. (1992). Distributed representation of fuzzy rules and its application to pattern classification, *Fuzzy Sets and Systems,* **52**, 21–32.

[39] Abe, S., Lan, M.S. (1992). A method for fuzzy rules extraction directly from numerical data and its application to pattern classification, *IEEE Trans. Fuzzy Syst.*, **3**, 18–28.

[40] Mitra, S., Pal, S.K. (1994). Self-organizing neural network as a fuzzy classifier, *IEEE Trans. Syst. Man Cybern.*, **24**, 385–399.

[41] Huang, Y. P. and Wang, S. F. (2007). Designing a fuzzy model by adaptive macroevolution genetic algorithms, *Fuzzy Sets and Systems*, **113**, 367–379.

[42] Ishibuchi, H. M. and Murata, T. (1997). Linguistic rule extraction from neural networks and genetic-algorithm-based rule selection, *in Proc. IEEE Int. Conf. Neural Networks, Houston, TX*, 2390–2395.

[43] Denna, M., Mauri, G., and Zanaboni, A. M. (1999). Learning fuzzy rules with tabu search-an application to control, *IEEE Trans. Fuzzy Syst.*, **7 (2)**, 295–318.

[44] Bagis, A. (2003). Determining fuzzy membership functions with tabu search-an application to control, *Fuzzy Sets and Systems*, **139**, 209-225.

[45] Markos, G. et al. (2008). A methodology for automated fuzzy model generation, *Fuzzy Sets and Systems*, **159**, 3201 – 3220.

[46] Chen, Y., Yang, B., Abraham, A., Peng, L. (2007). Automatic Design of Hierarchical Takagi–Sugeno Type Fuzzy Systems Using Evolutionary Algorithms, *IEEE Trans. Fuzzy Syst.*, **15 (3)**, 385-397.

[47] Kang, S. J. et al. (2000). Evolutionary design of fuzzy rule base for nonlinear system modeling and control, *IEEE Trans. Fuzzy Syst*., **8 (1)**, 37–45.

[48] Du, K.-L. (2009). Clustering: A neural network approach, *Neural Networks*, doi:10.1016/j.neunet.2009.08.007

[49] Chiu, S.L. (1994). Fuzzy Model Identification Based on Cluster Estimation, *Journal of Intelligent and Fuzzy System*, **2**, 267-278.

[50] Yager, R. and Filev, D. (1994). *Essentials of Fuzzy Modeling and Control*. New York: Wiley.

[51] Guillaume, S. (2001). Designing Fuzzy Inference Systems from Data: An Interpretability-Oriented Review, *IEEE Trans. Fuzzy Syst.*, **9 (3)**, 426-442.

[52] Han, M., Sun, Y., Fan, Y. (2008). An improved fuzzy neural network based on T–S model, *Expert Systems with Applications*, **34**, 2905–2920.

[53] Demirli, K. and Khoshnejad, M. (2009). Autonomous parallel parking of a car-like mobile robot by a neuro-fuzzy sensor-based controller, *Fuzzy Sets and Systems*, doi:10.1016/j.fss.2009.01.019

[54] Paiva , S.C and Dourado, A. (2004). Interpretability and learning in neuro-fuzzy systems, *Fuzzy Sets and Systems,* **147**, 17–38.

[55] Zhao, L., et al.. (2008). Eliciting compact T–S fuzzy models using subtractive clustering and co evolutionary particle swarm optimization, *Neurocomputing*, doi:10.1016/j.neucom.2008.11.001

[56] Eftekhari, M., et al.. (2008). Eliciting transparent fuzzy model using differential evolution, *Applied Soft Computing*, **8**, 466–476.

[57] Eftekhari, M. and Katebi, S. D. (2008). Extracting compact fuzzy rules for nonlinear system modelling using subtractive clustering, GA and unscented filter, *Applied Mathematical Modeling*, **32**, 2634-2651.

[58] Rantala, J., Koivisto, H. (2002). Optimized Subtractive Clustering for Neuro-Fuzzy Models, *International Conference on Fuzzy Sets & Fuzzy Systems, FSFS'02, Switzerland.*

[59] Demirli, K., Cheng, S. X., Muthukumaran, P. (2003). Subtractive clustering based modeling of job sequencing with parametric search, *Fuzzy Sets and Systems*, **137**, 235–270.

[60] Ma, J., Wang, T. (2006). A cost function Approach to Rival Penalized Competitive Learning, *IEEE Trans. On System Man and Cybernetics, Part B*, **36 (4)**, 722-736.

[61] Nielsen, R. H. (1987). Counterpropagation Networks, *Applied Opt*., **26**, 4979-4984.

[62] Ahalı, S. C. et al. (1990). Competitive Learning Algorithm for Vector Quantization, *Neural Networks*, **3 (3)**, 277-291.

[63] Xu, L., et al. (1992). Unsupervised and Supervised Classification by Rival Penalized Competitive Learning, *11 Int. Conf. Pattern Recognition, The Netherlands, Aug.30, Sep 3*, **1**, 672-675.

[64] Cheung, Y. M. (2005). On Rival Penalization Controlled Competitive Learning for Clustering with Automatic Cluster Number Selection, *IEEE Transaction On Knowledge and Data Engineering*, **17 (11)**, 1583-1588.

[65] Zhao, Z-S. et al. (2009). An evolutionary RBF Networks Based on Rival Penalized Competitive Learning and Its Application in Fault Diagnosis, *Proc. of 8th Inter. Conference on Machine Learning and Cybernetics, Boading, 12-15 Jully*, 1005-1009.

[66] Metropolis, N. et al. (1953). Equation of state calculation by fast computing machines, *J. Chem. Phys.*, **21**, 1087–1092.

[67] Kirkpatrick, S., Gelatt, C.D., Vecchi, M. P. (1983). Optimization by simulated annealing, *Science*, **220**, 671–680.

[68] Lee, Y., Lee, J. S., Lee, S. Y., Park, C. H. (2007) Improving Generalizationg capability of Neural Networks based on Simulated Annealing, *IEEE Congress on Evolutionary Computation (CEC2007)*, 3447-3453.

[69] Wang, X.Y., Garibaldi, J. (2005). Simulated annealing fuzzy clustering in cancer diagnosis, *Eur. J. Inf.,* 61–70.

[70] Haber, R. E. et al. (2009). An optimal fuzzy control system in a network environment based on simulated annealing. An application to a drilling process, *Appl. Soft Computing*, **9 (3)**, 889-895.

[71] Alizadeh, S., Ghazanfari, M. (2008). Learning FCM by chaotic simulated annealing, Chaos, *Solutions & Fractals (2008),* doi:10.1016/j.chaos.2008.04.058.

[72] Lin, S., Tseng, T., Chou, S. and Chen, S. (2008). A simulated annealing based approach for simultaneous parameter optimization and feature selection of backpropagation networks, *Expert System with Application*, **34**, 1491-1499.

[73] Abe, N., Kudo, M. (2006). Non Parametric Classifier –Independent Feature Selection, *Pattern Recognition,* **39**, 737-746.

[74] Chow, T. W. S., Wang, P. and Ma, E. W. (2008). A New Feature Selection Scheme Using a Data Distribution Factor for Unsupervised Nominal Data, *IEEE Transactions On Systems, Man, And Cybernetıcs—Part B: Cybernetıcs*, **38 (2)**, 499-502.

[75] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection, *J. Mach. Learn. Res*, **3**, 1157–1182.

[76] Liu, H., Motoda, H. (1998). Feature *Selection for Knowledge Discovery and Data Mining,* Kluwer Academic Publishers.

[77] Kohavi, R., John, G. H. (1998). Wrappers for Feature Subset Selection, *Artificial Intelligence,* **97 (2)**, 273-324.

[78] Sean, N. G., Thunshun, W. L. (2008). Medical data mining by fuzzy modeling with selected features, *Artificial Intelligence in Medicine*, **43**, 195-206.

[79] Pizzi, N. J., Pedrycz, W. (2008). Effective Classification using feature selection and fuzzy integration, *Fuzzy Sets and Systems*, **159**, 2859-2872.

[80] Chiang, J. H., Ho, S. H. (2008). A Combination of Rough based Feature Selection and RBF Neural Network for classification using Gene Expression, *IEEE Transaction on Nanobioscience*, **7 (1)**, 91-100.

[81] Atay, M. F. (2008). Support Vector Machines Combined with Feature Selection for Breast Cancer Diagnosis, *Expert System with Applications*, doi.10.1016 /j.eswa.2008.1.09.

[82] Karabatak, M., Ince, C. M. (2009). An expert system for detection of breast cancer based on association rules and neural network, *Expert Systems with Applications,* **36**, 3465–3469.

[83] Chou, C., et al. (2008). Network Intrusion Detection Design Using Feature Selection of Soft Computing Pradigms, *Int . Journal Com. Intelligence,* **4 (3)**, 196-208.

[84] Huang C. J., et al. (2008). Applications of wrapper Approach and Composite Classifier to the Stock Trend Prediction, *Expert System with applications*, **34**, 2870-2878.

[85] Mamdani, E. H., Assilian, S. (1975). An Experiment in Linguistic Synthesis with Fuzzy Logic Controller, *International Journal of Man-Machine Studies*, **7 (1)**, 1-13.

[86] Takagi, T., Sugeno, M. (1985). Fuzzy identification of a system and its application to modeling and control, *IEEE Trans. On Systems, Man and Cybernetics,* **15**, 135-156.

[87] Sugeno, M, Kang, G. T. (1988). Structure Identification of Fuzzy Models, *Fuzzy Sets and Systems,* **28**, 18-35.

[88] Abraham, A. (2005). *Handbook of Measuring System Design, edited by Peter H. Sydenham and Richard Thorn*. John Wiley & Sons, Ltd. ISBN: 0-470-02143-8

[89] Kosko, B. (1991). *Neural Networks and Fuzzy Systems.* Englewood Cliffs, NJ: Prentice-Hall.

[90] Abraham A. (2005).Adaptation of Fuzzy Inference System Using Neural Learning, *Studies in Fuzziness and Soft Computing*, Springer Berlin , Heidelberg, **181**, 53-80.

[91] Czogala, E., Leski, J. (2000). *Fuzzy and Neuro-Fuzzy Intelligent Systems*, Physica- Verlag Heidelberg, New York.

[92] Takagi, H. (1990). Fusion technology of fuzzy theory and neural network—Survey and future directions, *in Proc. Int. Conf. Fuzzy Logic and Neural Networks*, Iizuka, Japan, 13–26.

[93] Lin, C. T. and Lee, C. S. G. (1991). Neural Network based Fuzzy Logic Control and Decision System, *IEEE Transactions on Comput.* , **40 (12)**, 1320-1336.

[94] Berenji, H. R. and Khedkar, P. (1992). Learning and Tuning Fuzzy Logic Controllers through Reinforcements, *IEEE Transactions on Neural Networks,* **3**, 724-740.

[95] Nauck, D. and Kruse, R. (1994). NEFCON-I: An X-Window Based Simulator for Neural Fuzzy Controllers. *In Proceedings of the IEEE International Conference on Neural Networks,* Orlando, 1638-1643.

[96] Nauck, D. and Kruse, R. (1995). NEFCLASS: A Neuro-Fuzzy Approach for the Classiffication of Data, *In Proceedings of ACM Symposium on Applied Computing*, George K et al (Eds.), Nashville, ACM Press, 461-465.

[97] Nauck, D. and Kruse, R. (1999). Neuro-Fuzzy Systems for Function Approximation, *Fuzzy Sets and Systems*, **101**, 261-271.

[98] Sulzberger, S. M., Tschicholg-Gurman, N. N., Vestli, S. J. (1993). FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks, *In Proceedings of IEEE Conference on Neural Networks*, San Francisco, 312-316.

[99] Feng, J. C. and Teng, L.C. (1998). An Online Self Constructing Neural Fuzzy Inference Network and its Applications, *IEEE Transactions on Fuzzy Systems*, **6 (1)**, 12-32.

[100]Tano, S., Oyama, S. and Arnould, S. (1996). Deep combination of Fuzzy Inference and Neural Network in Fuzzy Inference, *Fuzzy Sets and Systems*, **82 (2)**, 151-160.

[101]Kasabov, N. and Qun, S. (1999). Dynamic Evolving Fuzzy Neural Networks with m-out- of-n Activation Nodes for On-line Adaptive Systems, *Technical Report TR99/04, Department of information science*, University of Otago, New Zealand, 1999.

[102] Kasabov, N. (1998). Evolving Fuzzy Neural Networks - Algorithms, Applications and Biological Motivation, In Yamakawa T and Matsumoto G (Eds), *Methodologies for the Conception, Design and Application of Soft Computing, World Scientific*, 271-274.

[103]Cordon, O. et al. (2001). *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*, World Scientific Publishing Company, Singapore.

[104] Mathworks (2001) *Fuzzy logic toolbox for use with Matlab. User's guide, version 2*. Natick, MA: The Mathworks, Inc.

[105]Rumelhart, D. E., et al. (1986). *Learning internal representations by error propagation*. In D. E. Rumelhart & J. L. McClelland (Eds.), Parallel distributed processing. Cambridge, MA: MIT Pres.

[106] Demuth, H. et al. (2010). *Neural Network toolbox user's guide*. The Mathworks Inc.

[107]Nauck, N., Klawonn, F. and Kruse, F. (1997). *Foundations of Neuro–Fuzzy Systems*. Chichester, U.K. Wiley.

[108]Taha, I. A. and Ghosh, I. A. (1999). Symbolic interpretation of artificial neural networks, *IEEE Trans. Knowl. Data Eng.*, **11**, 448–463.

[109]Mitra, S., Hayachi, Y. (2000) Neurofuzzy Rule Generation: Survey in Soft Computing Framework, *IEEE Tran. On Neural Networks,* **11 (3)**, 748-768.

[110]Yupu, Y., Xiaoming, X. and Wengyuan, Z. (1988). Real-time stable self-learning FNN controller using genetic algorithm, *Fuzzy Sets Syst.*, **100**, 173–178.

[111]Farag, W. A., Quintana, V. H. and Lambert-Torres, G. (1988). A genetic-based neuro–fuzzy approach for modeling and control of dynamical systems, *IEEE Trans. Neural Networks*, **9**, 756–767.

[112]Chopra S. et al. (2006). A Neuro Fuzzy Learning and Its Applications to Control System , *International Journal of Comp. Intelligent*, **3 (1)**, 72-78.

[113]Jang, J.-S. R., Sun, C.-T., Mizutani, E. (1997). *Neuro- Fuzzy and Soft Computing - A Computational Approach to Learning and Machine Intelligence*, Prentice Hall.

[114]Hartigan, J. A. and Wong, M. A. (1979). A k-means clustering algorithm, *Applied Statistics*, **28**, 100-108.

[115]Ling, J., Wiederkehr, U., Cabiness, S. (2008). Application of flow cytometry for biomarker-based cervical cancer cells detection, *Wiley InterScience, Diagnostic Cytopathology*, **36 (2)**, 76-84.

[116]Byriel, J. (1999). Neuro-*Fuzzy Classification of Cells in Cervical Smears*, MSc Thesis, Technical University of Denmark, Dept. of Automation.

[117] http://fuzzy.iau.dtu.dk/smear/

[118]Matthews, B.W., (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme, *in Biochemica et Biophysica Acta*, **405**, 442-551.

[119]Koss, L. (2000). The Application of PAPNET to Diagnostic Cytology", in P.Lisboa, E.Ifeachor,P.Szczepaniak (Eds.), *Artificial Neural Networks in Biomedicine*, 51-68.

[120]Pedrycz, W., Oliveira, J. V. (2007). *Advances in Fuzzy Clustering and its Applications*, John Wiley & Sons Inc., ISBN 978-0-470-02760-8.

[121] Ricketts, I. W. (1992). Cervical cell image inspection- a task for artificial neural networks, *Network*, **3**, 15 – 18.

[122] Palcic, B., et al. (1992). Comparison of three different methods for automated classification of cervical cells, *Analytical Cellular Pathology*, **4**, 429- 441.

[123]Mango, L. J. (1994). Computer-assisted cervical cancer screening using neural networks, *Cancer Letters*, **77**, 155 - 162.

[124]Zhong, L., Najarian, K. (2001). Automated classification of Pap smear tests using neural networks Neural Networks, *Proceedings. IJCNN '01*. (0-7803 7044-9).

[125] Ampazis, N., et al.(2004). Pap-smear classification using efficient second order neural network training algorithms. *Lecture Notes in Artificial Intelligence Subseries of Lecture Notes in Computer Science*, **3025**, 230-245.

[126] Dounias, G., et al. (2006). Automated identification of canceroussmears using various competitive intelligent techniques, *Oncology Reports*, **15**, 1001–1006.

[127]Marinakis, Y. et al. (2007). Particle swarm optimization for pap-smear diagnosis, *Expert Systems with Applications* (2007), doi:10.1016/j.eswa.

[128]Torun, Y., Tohumoğlu, G. (2008). Veri Kümeleme Yöntemlerininin Rahim Ağzı Kanserinde Kullanımı., *Akıllı Sistemlerde Yeni Uygulamalar Semp., ASYU, INISTA.*

[129]Hall, M. A. (1998). *Correlation-based feature selection for machine learning*. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand.

[130] Mathworks (2010) *Bioinformatic Tool Box, User's guide,* The Mathworks, Inc.

[131]Xiuju, F. and Wang, L. (2003). Data Dimensionality Reduction With Application to Simplifying RBF Network Structure and Improving Classification Performance *IEEE Trans. On Systems, Man and Cybernetics, Part B, Cybernetics*, **33 (3)**, 399-410.

[132]Yijuan, L., et al. (2007). Feature Selection Using Principal Feature Analysis, *ACM Multimedia*, Augsburg, Germany, September, 23-29.

[133]Romeijn, H. E., et al. (1999). New reflection generator for simulated annealing in mixed-integer/continuous global optimization, *Journal of Optimization Theory and Applications*, **101**, 403–427.

[134]Hettich, S., Blake, C. L., and Merz, C. J. (1998). UCI repository of machinelearning databases. Irvine, CA: Department of Information andComputer Sciences, University of California. Available from :http://www.ics.uci.edu/~mlearn/MLRepository.html.

[135] Weiss, S. M., Kulikowski, C. A. (1991). *Computer Systems that Learn*, Morgan Kaufmann Publishers, San Mateo.

[136]Bacardit, J. (2004), *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representation, Generalization, and Run-time*, Ramon Llull University, Barcelona, Catalonia, Spain.

[137] Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence, **1 (4)**, 224-227.

[138]Chou, C. H., Su, M. C., and Lai, E. (2004). A new cluster validity measure and its application to image compression Sergios theodoridis, *Pattern Anal. Applic.,* **7**. 205-220.

APPENDIX

A1. Programme Codes (Matlab m-file Programming)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
anneal_for_multiminima.m
%modified simulated annealing for searching global optima
function [minimum,fval,hata,energy,T_,itry,parent,x] =
anneal_for_multiminima(loss, parent, options)
clf
load rastring_data;
def = struct(...
    'CoolSched',@(T) (.8*T),...
    'Generator', @(x) (x+(randperm(length(x))==length(x))*randn),...
    'InitTemp',1,...
    'MaxConsRej',1000,...
    'MaxSuccess',1,...
    'MaxTries',20,...
    'StopTemp',1e-6,...
    'StopVal',-Inf,...
    'Verbosity',1);
options=def;
% main settings
newsol = options.Generator;       % neighborhood space function
Tinit = options.InitTemp;         % initial temp
minT = options.StopTemp;          % stopping temp
cool = options.CoolSched;         % annealing schedule
minF = options.StopVal;
max_consec_rejections = options.MaxConsRej;
max_try = options.MaxTries;
max_success = options.MaxSuccess;
report = options.Verbosity;
k=1  % boltzmann constant expp=0;itry = 0;success = 0;finished = 0;consec =
0;T = Tinit;prop=0;
total = 0;ul=5;ll=-5;limits=[-5 -5 ; 5 5];tt=1;ttt=1;pt=1;t=1;b=0;
parent=[-5 -5];
loss=@(p)rastriginsfcn(p);
initenergy = loss(parent);
oldenergy = initenergy;
while ~finished;
    itry = itry+1;
    current = parent;
    if T<1e-3
        funct=1;
        threshold=1e-6;
        max_try=50;
        max_success=40;
        max_consec_rejections=40;
        newsol=@(x)(x+(randperm(length(x))==length(x))*randn*T*5);
    else
        funct=2;
        max_try=50;
        max_success=40;
        threshold=1e-6
        max_consec_rejections=40;
        newsol=@(x)(x+(randperm(length(x))==length(x))*randn);
    end
    % % Stop / decrement T criteria
    if itry >= max_try || success >= max_success ||consec >=
max_consec_rejections;
        if T < minT ;
            finished = 1;
            total = total + itry;
            break;
        else
            x(tt,:)=[parent];
            b=b+1;
            figure(3);
line(parent(1),parent(2),'Color','r','LineStyle','.','LineWidth',2,'Marker',
'+');
            text(parent(1),parent(2),['T =
',num2str(b)],'Color','b','FontSize',8)
            T = cool(T);  % decrease T according to cooling schedule
            T_(tt)=oldenergy;
            fprintf(1,'  T = %5.10f, loss = %5.10f,x1 = %5.5f,x2 = %5.5f
\n',T...
            ,oldenergy,parent(1),parent(2));
            energy(t)=oldenergy;
            t=t+1;
            total = total + itry;
            itry = 1;
            success = 1;
            tt=tt+1;
        end
    end
    if funct==0
        newparam=[(ul-ll)*rand+ll (ul-ll)*rand+ll];
    else
        newparam = newsol(current);
    end
        while  ~all(newparam<=limits(2,:)) |  ~all(newparam>=limits(1,:))
        newparam = newsol(current);
        end
    newenergy = loss(newparam
    fprintf(1,'  newenergy = %7.5f,\',newenergy);
    if (newenergy < minF
       parent = newparam
       oldenergy = newenergy
       fprintf(1,'  newenergy < minF  = %7.5f,\n',newenergy)
      break
    end
    if (oldenergy-newenergy > 1e-6)
       parent = newparam;
       oldenergy = newenergy;
       success = success+1;
       consec = 0;                                         fprintf(1,'
       ACCEPTED\n');
     fprintf(1,' oldenergy-newenergy>1e-6 newparam = %7.5f,\n',newparam);
    else
if (rand < exp( (oldenergy-newenergy)/(k*T) ));
            parent = newparam;
            oldenergy = newenergy;
            success = success+1;
            prop=prop+1;
            fprintf(1,'  ACCEPTED PROP\n');
            ttt=ttt+1 ;
            pt =pt+1;
        else
           consec = consec+1;
           fprintf(1,'  REJECTED\n');
        end
    end
    p(tt)=pt;
    pt=0;
end
minimum = parent;
fval = oldenergy;
hata=minn-fval;
if report;
    fprintf(1, ' \nInitial temperature:     \t%g\n', Tinit);
    fprintf(1, '  Final temperature:       \t%g\n', T);
    fprintf(1, '  Consecutive rejections:  \t%i\n', consec);
    fprintf(1, '  Number of function calls:\t%i\n', total);
    fprintf(1, '  Total final loss:        \t%g\n', fval);
    fprintf(1, '  Probablity of Acceptance:        \t%g\n', prop);
end
xx=10:.1:20;
n =length(xx);
figure(1);meshc(x1,x2,A);
figure(2);plot(T_);
figure(3);line(x(:,1),x(:,2),'Color','r','LineWidth',2,'Marker','+');
hold on
figure(3);contour(x1,x2,A);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
anneal_tsk.m
%SIMULATED ANNEALING SUBTRACTIVE CLUSTERING BASED FUZZY
CLASSIFIER
%%
function [minimum,fval] = annealtsk(func,parent,options)
diary annealtskSI2009
c = fix(clock);
diary on
    initdata;
    clear all;
    load initdata
    [m n]=size(in_all)
    %%
    def = struct(...
        'CoolSched',@(T)  (.8*T),...
        'Generator',@(x)  (x+(randperm(length(x))==length(x))*randn/100),...
         'InitTemp',1,...
        'MaxConsRej',1000,...
        'MaxSuccess',5,...
        'MaxTries',20,...
        'StopTemp',5e-5,...
        'StopVal',-inf,...
        'Verbosity',1);
    options=def;
    %% MAIN SETTING
    newsol = options.Generator;      % neighborhood space function
    Tinit = options.InitTemp;        % initial temp
    minT = options.StopTemp;         % stopping temp
    cool = options.CoolSched;        % annealing schedule
    minF = options.StopVal;
    max_consec_rejections = options.MaxConsRej;
    max_try = options.MaxTries;
    max_success = options.MaxSuccess;
    report = options.Verbosity;
    k = 1;                           % boltzmann constant
    %% CONSTANT
    itry = 0;success = 0;finished = 0;consec = 0;
    T = Tinit; parent=.7; defuzz=.5;it=1; ul=1;uld=.7; lld=.2; ll=.5;
    k=1;    md=1;tt=1; ttt=1;tttt=1;
    %% INITILIASE
    [initenergy,performance] =   tsk(parent,index,in_all,out_all,defuzz);
    oldenergy = initenergy;
    total = 0;
    prop=0;
    fprintf(1,'\n  T = %7.5f, loss = %10.5f,performance = %10.5f
\n',T,oldenergy,performance);
    energy(1)=0;
    energy(2)=oldenergy;
    %% SA ALGORITHM
    while ~finished;
        itry = itry+1; % just an iteration counter
        current = parent;
        newindex=index;
        currentdefuzz=defuzz;
        if T<1e-4  % ıf temprature is low ;new options
            funct=1;
            threshold=1e-6;
            max_try=150;
            max_success=75;
            max_consec_rejections=50;
        else    %if temprature is hıgh ;new options
            funct=0;
            max_try=200;
            max_success=100;
            threshold=1e-6;
            max_consec_rejections=75;
        end
        % % Stop / decrement T criteria
        if itry >= max_try || success >= max_success ||consec >=
max_consec_rejections;
            if T < minT % %  || oldenergy==0;
                finished = 1;
                total = total + itry;
                break;
            else
                T = cool(T);  % decrease T according to cooling schedule
                fprintf(1,'  T = %7.5f, loss = %10.5f , parent= %10.5f,
defuzz=%10.5f, Testperf = %10.5f,Trainperf = %10.5f\',T,
oldenergy,parent,defuzz,perform,newperformancetrain);
                fprintf(1,'  %3.3g\',index);
                fprintf(1,'  # of Features = %5.0f\',length(newindex))
                fprintf(1,'  \n');
                itry = 1;
                success = 1;
                consec=0;
                tt=tt+1;
                ttt=ttt+1;
                energy(tt)=oldenergy;
            end
        end

        if funct==0 % if temprature is high then use random vector for
obtaining new sol'n
            newparam=(ul-ll)*rand+ll;
            newdefuzz=(uld-lld)*rand+lld;
        else % if temprature is low then use  neighborhood search for new
sol'n
            newparam = newsol(current);
            newdefuzz=newsol(currentdefuzz);
        end
        % produce new defuzz sol'n
         if newdefuzz<ll ||newdefuzz>ul % if new sol'n out of limit
            newdefuzz=(uld-lld)*rand+lld;
            %fprintf(1,' Limit asimi var= %10.0f\n',newparam);
         end
        if newparam<ll ||newparam>ul   % if new sol'n out of limit
            newparam=(ul-ll)*rand+ll;
            %fprintf(1,' Limit asimi var= %10.0f\n',newparam);
        end
           newindex=randint(n,1,[0,1])';
           newindex=find(newindex>0);
           newindex=union(subindex,newindex);
           newindex=intersect(index3,newindex);

[newenergy,performance,hatax,ctr,pztf,ngtf,tpztf,tngtf,performancetrain,test
std,trainstd] = tsk(newparam,index,in_all,out_all,newdefuzz) ; % calculate
the output for  current values
        perftr(it)=performancetrain;
        perfts(it)=performance;
        if mean(energy)<1e-2
            fnished=1;
            fprintf(1,'No more optimisation will be done\n' );
        end
        rmsee(itry)=newenergy;
        if (newenergy < minF)     % if minimisation has done
            parent = newparam;
            defuzz=newdefuzz;
            index=newindex;
            fprintf(1,' ACC  newenergy < minF parent= %10.0f\n',parent);
            oldenergy = newenergy;
            newhata=hatax;
            newctr=ctr;
            newteststd=teststd;
            newtrainstd=trainstd;
            break
        end
        if (oldenergy-newenergy > threshold) % if new parameter cause
improvement
            parent = newparam;          % take current radii as radii
            defuzz=newdefuzz;          %% take current defuzz as defuzz
            index=newindex;             %% take current Features as Features
            oldenergy = newenergy;      % take current energy as energy
            perform=performance;
            newhata=hatax;
            newctr=ctr;
            newperformancetrain=performancetrain;
            newteststd=teststd;
            newtrainstd=trainstd;
            newngtf=ngtf;
            newpztf=pztf;
            newtngtf=tngtf;
            newtpztf=tpztf;
            success = success+1;        %
            consec = 0;
            fprintf(1,' SOLN ACC LOWER \n');
        else
            if (rand < exp( (oldenergy-newenergy)/(k*T) ));% if the new sol
is not better but it is a candicate for next iteration
                parent = newparam;    % take current radii as radii
                defuzz=newdefuzz;     %take current defuzz as defuzz
                index=newindex;       %take current Features as Features
                oldenergy = newenergy;% take current energy as energy;
                newhata=hatax;
                newctr=ctr;
                perform=performance;
                newperformancetrain=performancetrain;
                newteststd=teststd;
                newtrainstd=trainstd;
                newngtf=ngtf;
                newpztf=pztf;
                newtngtf=tngtf;
                newtpztf=tpztf;
                success = success+1;
                prop=prop+1;
                p(ttt,tttt)=prop;
                tttt=tttt+1;
                fprintf(1,'SOLN ACC PROP \n');
            else   % if current sol is not better then reject it
                consec = consec+1;
                fprintf(1,'SOLN REJECTED \n');
            end
           fprintf(1, '  Number of succeess :  \t%i\n', success);
        end
    end
    %%
    %% ALGORTIHM OUTPUTS
    newindex=index;
    fval = oldenergy;

save('annealtsk','rmsee','newindex','parent','fval','energy','defuzz','newha
ta','newctr','newperformancetrain','newngtf','newpztf','newtngtf','newtpztf'
,'newteststd','newtrainstd','perftr','perfts');
    if report;
        fprintf(1, '\n  Initial temperature:     \t%g\n', Tinit);
        fprintf(1, '  Final temperature:       \t%g\n', T);
        fprintf(1, '  Consecutive rejections:  \t%i\n', consec);
        fprintf(1, '  Acceptence probablity  :  \t%i\n', prop);
        fprintf(1, '  Number of function calls:\t%i\n', total);
        fprintf(1, '  Number of iteration :  \t%i\n', itry);
        fprintf(1, '  final tsk_y2sa value :        \t%g\n', fval);
        fprintf(1, '  Final Substractive Cl Radii:        \t%g\n', parent);
        fprintf(1, '   :        \t%g\', newindex);
    end
    d = fix(clock)
    ni=length(nfis.input)
    for i=1:ni
    [x,mf] = plotmf(nfis,'input',ni);
    xx(ni)={x};
    mff(ni)={mf};
    subplot(6,1,ni), plot(xx(ni),mff(ni));
    end
    u=[newctr ;newngtf];
    u=[newctr;newngtf;newpztf];
    u=[newctr;newngtf;newpztf];
    a=newctr + newhata;
    u=[a;newctr;newngtf;newpztf;newtngtf;newtpztf]
    sensivity=mean(100*newtpztf./(newtpztf+newngtf));
    specificity=mean(100*newtngtf./(newtngtf+newpztf));
    Positive_predictive_value=mean(100*newtpztf./(newtpztf +newtngtf));
    Negative_predictive_value=mean(100*newtngtf./(newtpztf +newtngtf));
    performanceoftesting=mean(100*newctr./(newctr+newhata));
    fprintf(1, '  sensivity                     :\t%g\n',sensivity);
    fprintf(1, '  specificity                   :\t%g\n',specificity);
    fprintf(1, '  Positive_predictive_value
:\t%g\n',Positive_predictive_value);
    fprintf(1, '  Negative_predictive_value
:\t%g\n',Negative_predictive_value);
    fprintf(1, '  Performance of Testing
:\t%g\n',performanceoftesting);
    fprintf(1, '  Standard Deviation of Testing :\t%g\n',newteststd);
    fprintf(1, '  Performance of Training
:\t%g\n',newperformancetrain);
    fprintf(1, '  Standard Deviation of Training:\t%g\n',newtrainstd);
% end
diary off
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tsk.m
%Takagi Sugeno type fuzzy classifier with SC for 2 class
problems


function
[hata,performance,hatax,ctr,pztf,ngtf,tpztf,tngtf,performancetrain,teststd,t
rainstd,nfis] = tsk(parent,newindex,in_all,out_all,defuzz)
clear hata;
clear ctr;
in_allx=in_all;
t=1;
tt=1;
N=10;
indices = crossvalind('Kfold',out_all,N);
for k=1:N
    testx = (indices == k); trainx = ~testx;
    Ev_i=in_allx(testx,:);
    Ev_o=out_all(testx,:);
    Tr_i=in_allx(trainx,:);
    Tr_o=out_all(trainx,:);
    nfis=genfis2(Tr_i,Tr_o,parent);
    [mm nn]=size(Tr_i);
    for t=1 :size(nn)
        mmin=min(Tr_i(:,t));
        mmax=max(Tr_i(:,t));
        nfis.input(t).range=[0 1];
    end
    warning off all
    [mt nt]=size(Tr_i);
    ctrtrain(k)=0;
    trainaccu=evalfis(Tr_i,nfis);
    for i=1:mt
        if trainaccu(i)<  defuzz
            trainaccu(i)=0;
        else
            trainaccu(i)=1;
        end
    end
    for i = 1:mt
        if trainaccu(i)==Tr_o(i)
            ctrtrain(k) = ctrtrain(k) + 1;
        end
    end
    [mm nn]=size(Ev_i);
    y=evalfis(Ev_i,nfis);
    yout=y;
    n=length(y);
  for i=1:mm
        if  y(i)<= defuzz
            y(i)=0;
        else
            y(i)=1;
        end
    end
    ctr(k) = 0;
    pztf(k)=0;
    ngtf(k)=0;
    tpztf(k)=0;
    tngtf(k)=0;
    for i = 1:mm
        if y(i)==Ev_o(i)
            ctr(k) = ctr(k) + 1;

            if y(i)==1
```

```
                tpztf(k)= tpztf(k)+1;
            else
                tngtf(k)=tngtf(k)+1;
            end
        else
            if y(i)==0;
                pztf(k)=pztf(k)+1;
            else
                ngtf(k)=ngtf(k)+1;
            end
            ab(tt,:)={Ev_i(i,:)} ;
            a(tt)=Ev_o(i);
            tt=tt+1;
        end
    end
   hatax(k)=mm-ctr(k);
   hatatrain(k)=mt-ctrtrain(k);
   rmse(k)= norm(y-Ev_o)/sqrt(length(y));
end
performance=100*ctr./(ctr+hatax);
teststd=std(performance);
performancetrain=100*ctrtrain./(ctrtrain+hatatrain);
trainstd=std(performancetrain);
performance=mean(performance);
hata=100-performance;
performancetrain=mean(performancetrain);
rmse=mean(rmse);
fprintf(1,'parent=%5.5f,defuzz=%5.5f \',parent,defuzz);
fprintf(1,',RMSE=%5.5f,hata=%5.5f,Accuraccy=%5.5f\',rmse,hata,performance)
fprintf(1,'%2.0f \',newindex);
fprintf(1,'# of Features = %5.0f\',length(newindex))
save ('tsk_y2saBC','nfis','ab');
fprintf(1,' \n');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
İnitdata.m
%initiliaze the data for classification task

% fprintf(1,'Select one of the problem and type its corresponding
number\n');
% fprintf(1,' 1-)    PAP SMEAR PROPLEM\n');
% fprintf(1,' 2-)    WISCONSIN BREAST DATA\n');
% fprintf(1,' 3-)    IONOSHPERE STRUCTURE  \n');
% fprintf(1,' 4-)    PIMA INDIANS DIABETS  \n');
% fprintf(1,' 5-)    IONOSHPERE STRUCTURE  \n');
% fprintf(1,' 6-)    HOUSING  \n');
% fprintf(1,' 7-)    BUPA LIVER  \n');
% fprintf(1,' 8-)    AUSTRALIAN CREDIT APROVAL  \n');
% fprintf(1,' 9-)    GERMAN CREDIT APROVAL  \n');
% fprintf(1,'10-)    HEART DISEASE  \n');
% fprintf(1,'11-)    SONAR \n');
% fprintf(1,'12-)    VEHICLEI  \n');
% fprintf(1,'13-)    VOWEL  \n');
% fprintf(1,'14-)    GLASS CLASSIFICATION \n');
% fprintf(1,'15-)    WINE RECOGNATION \n');
% fprintf(1,'15-)    IRIS PLANT \n');
% fprintf(1,'16-)    BALANCE SCALE WEIGHT  \n');
% fprintf(1,'17-)    CREDIT APPROVAL \n');
% fprintf(1,'18-)    LABOR NEGOTIATIONS \n');
% fprintf(1,'19-)    WAWEFORM \n');
uiopen('LOAD');
all=data;
[m n] =size(all);
l=1;
t=1;
[m n] =size(all);
if max(all(:,end)) >5
for i = 1:m % the output (last column) values (0,1,2,3) are mapped to (0,1)
    if all(i,end)>=5
        all(i,end)=1;
    else
        all(i,end)=0;
    end
end
end
for i = 1:n
    range(1,i) = min(all(:,i));
    range(2,i) = max(all(:,i));
end
for i=1:n
    for j=1:m
        all(j,i)=(all(j,i)-range(1,i))/(range(2,i)-range(1,i));
    end
end
[m n] =size(all);
l=1;
t=1;
%%
% FIND VARIATION
for i=1:n-1
    vary(i)=var(all(:,i));
    if vary(i)<0.1
        indicesminus(l)=i;
        l=l+1;
        index1(i)=0;
    else
        indiceplus(t)=i;
        t=t+1;
        index1(i)=i;
    end
end
```

```
%%
%FIND CORRELATION BETWEEN INPUT AND OUTPUT
x=all(:,end);
for i=1:n-1
    y=all(:,i);
    r=corrcoef(y,x);
    rr(i)=abs(r(2,1));
    er(i)=var(y)*(1-rr(i).^2);
    er(i)=var(y)-er(i);
    if rr(i)>.1;
     index2(i)=i;
    else
    index2(i)=0;
    end
end
% SEARCH FOR REDUNDANCY BETWEEN FEATURES
[m n]=size(all);
for i=1:n-1
    for j=1:n-1
        if i ~= j
            r =corrcoef(all(:,i),all(:,j));
            rtrcros=abs(r(2,1));
            crosscorrelation(i,j)=rtrcros;
        end
    end
end
for i=1:n-1
    for j=1:n-1
        if crosscorrelation(i,j)>.9% |i == j
            a(i,j)=1;
        else
            a(i,j)=0;
        end
    end
end
index4=[];
index5=[1:n-1];
for i=1:n-1
    b(i)={find(a(i,:)>0)};
    m(i)=length(b{i});
end
c=b;
setfull=[1:n-1];
setunc=find(m==0);
setcorone=find(m==1);
setcor=find(m>1);
setcorone=c(setcorone);
%c(setunc)=[]
for i=1:n-1
    if m(i)>0
        for j=1 :m(i)
            if isempty(c{i})==0
                aa=c{i}(j);
                c{aa}=[];
            end
        end
    end
end
for i=1:n-1
    if  length(c{i})==0
        c{i}=0;
    else
        c{i}=i;
    end
end
%%
setunc1 = cell2mat(c);
```

```
setfull(setunc)=0;
index3=union(setunc1,setunc);
index=find(index3==0);
index3(index)=[];
%%
in=find(index1>0);
index1=index1(in);
in=find(index2>0);
index2=index2(in);
index=union(index1,index2);
index=union(index,index3);
subindex=union(index1,index2);
index=[1:n-1];
%%
in_all=all(:,index);
out_all=all(:,end);
savefile='initdata';
save(savefile,'in_all','out_all','index1','index2','index3','subindex','inde
x');
clear all;
clear all;
load initdata;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
NFC1.m
NeuroFuzzy Classifier1
function [Classifier Ac1]=NFC1(problem)
%Batch Training RPCL clustering based NeuroFuzzy Classifier
%February 2009;
clc
close all
vis=1;
screen=0;
teta=.05;%
error_bound=.5;
fring_bound=1;
train_opt = [1e-5  1e-3 5e-8  5e-9  100];
N=2;%N  fold  validation
warning off
model=1;
load result_rival50
if nargin<1
    problem=6;
end
data=result_rival{model,problem}.data;
display(result_rival{model,problem}.name)
tolerance = train_opt(1);   % Stop learning once RMSE is below tolerance
eta1 = train_opt(2);            % Learning rate
eta2=train_opt(3);
eta3=train_opt(4);
max_epoch = train_opt(5);   % Max. training epoch
indices = crossvalind('Kfold',data(:,end),N);
for optStep=1:N
    max_epoch = train_opt(5);
    display('**********************************************************')
    fprintf('>>>>>>>>>--------     FOLD   %5.0f  ------<<<<<<<\n',optStep)
    display(' ');
    testx = (indices == optStep);
    trainx = ~testx ;
    %trainx = (indices == optStep);
    data_train=data(trainx,:);
    data_test=data(testx,:);
    epoch=1;
    %% obtaining Classifer parameters
    clear Classifier
    if problem==7 || problem==8||problem==9||problem==10
        Classifier.class=3;
    else
        Classifier.class=2;
    end
    Classifier.tr_input=data_train(:,1:end-1);
    Classifier.tr_output=data_train(:,end);
    Classifier.ts_input=data_test(:,1:end-1);
    Classifier.ts_output=data_test(:,end);
    Classifier.numData=result_rival{model,problem,1}.numData_in_cl;
   Classifier.in_cluster=abs(result_rival{model,problem}.cluster{1}(:,1:end-
1));
  Classifier.out_cluster=abs(result_rival{model,problem}.cluster{1}(:,end));
    Classifier.numRule=size(Classifier.out_cluster,1);
    [Classifier.tr_numPattern Classifier.numInp]=size(Classifier.tr_input);
    [Classifier.ts_numPattern Classifier.numInp]=size(Classifier.ts_input);
    Classifier.sigmas=2*result_rival{model,problem}.sigmas(:,1:end-1);
    Classifier.sigmas=.1*ones(Classifier.numRule,Classifier.numInp);
    Classifier=evalnetwork_nfc(Classifier);
    numRule1=Classifier.numRule;
    RMSE(1)=Classifier.RMSE;
  if visfigure_number=1; display_result(Classifier,figure_number,RMSE); end;
    fnished=0;
    fnished2=0;
    epoch=2;

    while fnished==0;
        RMSE(epoch)=Classifier.RMSE;
        Classifier =bckprop(Classifier)
       if RMSE(epoch) < tolerance
            fnished = 1;
        end
        if epoch == max_epoch
           Ac1.training(optStep)=Classifier.Accurracy;
            break
            if fnished2
                display('                       LEARNING FNISHED          ')
                fprintf('Final Number of Rules is
=%5.0f\n',Classifier.numRule);
                fprintf('Final RMSE  =%5.7f\n',RMSE(epoch));
                fprintf('Final Classification Accurracy: Training =%g,
Testing =%g\n',Classifier.Accuracy,Classifier.Accuracy_ts);
                %display_result(Classifier,figure_number,RMSE
                Ac1.training(optStep)=Classifier.Accuracy;
                Ac1.testing(optStep)=Classifier.Accuracy_ts;
                Classifier=pruning(Classifier,fring_bound);
                Classifier=evalnetwork_nfc(Classifier);
                fprintf('Final Classification Accurracy: Training =%g,
Testing =%g\n',Classifier.Accuracy,Classifier.Accuracy_ts);
                fnished=1;
                epoch=2;
               break;
            end
            % pause(1);
            figure_number=2;
            if vis display_result(Classifier,figure_number,RMSE); end;
            display('Second Phase for Structure Re Organinggg.......... ');
            %Classifier=pruning(Classifier,fring_bound);
            Classifier=remove_redund(Classifier,teta,RMSE,error_bound,vis);
            Classifier=pruning(Classifier,fring_bound);
            fprintf('Final Acc before BP. Training =%5.5f: Testing =%5.5f:
Final RMSE = %.10f:F.Rule=%2.0f:\n',...
Classifier.Accuracy,Classifier.Accuracy_ts,Classifier.RMSE,Classifier.numR
ule);
            display('Re Learning with backpropagation..................');
            fnished2=1;
            max_epoch=train_opt(4)*2;
        end
        if screen
            fprintf('epoch %.0f:  RMSE = %.10f:Train Acc=%5.2f,Test
Acc=%5.2f, eta1=%.10f : eta2=%.10f\n',...
                epoch,
RMSE(epoch),Classifier.Accurracy,Classifier.Accurracy_ts,eta1,eta2);
        end
        leng=length(RMSE);
        if leng==4
            if(RMSE(2)-RMSE(3))<1e-4
                eta1=10*eta1;
                eta2=10*eta2;
            end
            if(RMSE(3)-RMSE(4))<1e-4
                eta1=10*eta1;
                eta2=10*eta2;
            end
        end
        if leng>6
            if(RMSE(epoch)-RMSE(epoch-1))>1e-5
                eta1=eta1*.95;
                eta2=eta2*.95;
            end
            if mean(diff(RMSE(leng-5:leng)))<-0.001
                eta1=eta1*1.1;
        eta2=eta2*1.1;

      elseif abs(mean(diff(RMSE(leng-5:leng))))<1e-10
        eta1=eta1*.95;
        eta2=eta2*.95;
      else
      end
    end
    for n=1:Classifier.class
    w_old=Classifier.Clc{n}.w;
    w_new=w_old-.01*Classifier.Clc{n}.dEdW;
    Classifier.Clc{n}.w=w_new;

    s_old=Classifier.Clc{n}.sigmas;
    s_new=s_old - 1e-12*Classifier.Clc{n}.delta_s;
    Classifier.Clc{n}.sigmas=s_new;

    c_old=Classifier.Clc{n}.in_cluster;
    c_new=c_old-1e-14*Classifier.Clc{n}.delta_c;
    Classifier.Clc{n}.in_cluster=c_new;

    end
   if vis ;figure_number=2;
     display_result(Classifier,figure_number,RMSE) ; end
    Classifier=evalnetwork_nfc(Classifier);
    epoch=epoch+1;
  end

end
fprintf('Final Training Accurraccy:%5.5f  std=:%5.5f
\n',mean(Ac1.training),std(Ac1.training));
fprintf('Final Testing Accurraccy:%5.5f
std=:%5.5f\n',mean(Ac1.testing),std(Ac1.testing));

function display_result(Classifier,figure_number,RMSE)
figure(figure_number);
if Classifier.class==3
subplot(4,1,1),plot(1:Classifier.tr_numPattern,Classifier.Clc{1}.error,...
    1:Classifier.tr_numPattern,Classifier.Clc{2}.error,...
    1:Classifier.tr_numPattern,Classifier.Clc{3}.error)
  Ylabel('Classifier Error') ;
  Xlabel('Input Pattern') ;
  theStr=sprintf(' Classifier Accurracy=   %g',Classifier.Accurracy);
  Title(theStr,'Color','b','FontSize',12)
 subplot(4,1,2),plot(1:Classifier.tr_numPattern,Classifier.Clc{1}.output,...
    1:Classifier.tr_numPattern,Classifier.Clc{2 }.output,...
    1:Classifier.tr_numPattern,Classifier.Clc{3}.output);
  Ylabel('Final Classifier Output ') ;
  Xlabel('Input Pattern') ;
  if Classifier.class==3
  subplot(4,1,3),plot([Classifier.Clc{1}.rule_firing
Classifier.Clc{2}.rule_firing...
    Classifier.Clc{3}.rule_firing])
  Ylabel(' Classifier Output ') ;
  Xlabel('Input Pattern') ;
  else
    subplot(4,1,3),plot([Classifier.Clc{1}.rule_firing
Classifier.Clc{2}.rule_firing]);
  Ylabel(' Classifier Output ') ;
  Xlabel('Input Pattern') ;
  end
  subplot(4,1,4),plot(RMSE);
  Ylabel(' RMSE ') ;
  Xlabel('Number of iteration') ;
  pause(.05)
else if Classifier.class==2
subplot(4,1,1),plot(1:Classifier.tr_numPattern,Classifier.Clc{1}.error,...
      1:Classifier.tr_numPattern,Classifier.Clc{2 }.error);
```

```
        Ylabel('Classifier Error') ;
        Xlabel('Input Pattern') ;
        theStr=sprintf(' Classifier Accurracy=   %g',Classifier.Accurracy);
        Title(theStr,'Color','b','FontSize',12);
subplot(4,1,2),plot(1:Classifier.tr_numPattern,Classifier.Clc{1}.output,...
            1:Classifier.tr_numPattern,Classifier.Clc{2 }.output);
        Ylabel('Final Classifier Output ') ;
        Xlabel('Input Pattern') ;
        subplot(4,1,3),plot([Classifier.Clc{1}.rule_firing
Classifier.Clc{2}.rule_firing]);
        Ylabel(' Classifier Output ') ;
        Xlabel('Input Pattern') ;
        subplot(4,1,4),plot(RMSE);
        Xlabel(num2str(Classifier.Accuracy));
        Ylabel(' RMSE ') ;
        Xlabel('Number of iteration') ;
        pause(.05)
    end
end
%%
function Classifier=remove_redund(Classifier,teta,RMSE,error_bound,vis)
if nargin<2
    teta=.1;
end
c_inc=0;
m_init=0;
for t=1:Classifier.class
    clear empty_index empty_index1 a empty_index2
    empty_index=find(Classifier.Clc{t}.error>=error_bound);
    Classifier.tr_input(empty_index,:);
    m=size(Classifier.tr_input(empty_index,:),1);
    if m>0
        dist=zeros(m,m);
        inc=1;
        for i=1:m
            for j=1:m
                dist(i,j)=sum((Classifier.tr_input(i,:)-
Classifier.tr_input(j,:)).^2);
            end
            a{i}=find(dist(i,:)<teta);

            if size(a{i},2)==1
                empty_index1(i)=empty_index(i);
                cmean(i,:) =Classifier.tr_input(empty_index(i),:);
            else
                empty_index1(i)=empty_index(a{i}(1));
                cmean(i,:)=mean(Classifier.tr_input(a{i},:));
                % dist(i,a{i})=10;
            end
        end
        %empty_index3=empty_index1;
        empty_index2=[];
        inc=1;
        for i=1:size(empty_index1,2)
            indx=find(empty_index1(i)==empty_index1);
            empty_index2(inc)=empty_index1(indx(1));
            empty_index1(indx)=0;
            inc=inc+1;
        end
        empty_index3=find(empty_index2==0)';
        empty_index3=sort(empty_index3,'descend');
        cmean(empty_index3,:)=[];
        empty_index2=empty_index2';
        empty_index2=sort(empty_index2,'descend');
        empty_index2(empty_index2==0)=[];
        c_add=size(empty_index2,1);
        c_size=size(Classifier.Clc{t}.in_cluster,1);
        if not(isempty(c_size+1:c_size+c_add))
```

```
            Classifier.Clc{t}.in_cluster(c_size+1:c_size+c_add,:)=...
                Classifier.tr_input(empty_index2,:);
            Classifier.Clc{t}.sigmas(c_size+1:c_size+c_add,:)=...
                0.05*ones(c_add,Classifier.numInp);
        end
        c_inc=c_add+c_inc;
        m_init=m_init+m;
    end
end
Classifier.numRule=Classifier.numRule+c_inc;
Classifier=evalnetwork_nfc(Classifier);
figure_number=3;
if vis display_result(Classifier,figure_number,RMSE); end;
display('                      RESULT OF STRUCTURE REORGANAZING          ')
fprintf('%5.0f: Rules are added  because of the Error\n',m_init);
fprintf('%5.0f: Rules are removed  because of the Similarity\n',m_init-
c_inc);
%%
function Cl=pruning(Cl,fring_bound)
deleted=[];
Rulenum=0;
for i=1:Cl.class
    numRule=size(Cl.Clc{i}.in_cluster,1);
    m=1;
    for j=1:numRule
        if size(find(j==(Cl.Clc{i}.ind3)),2)<fring_bound
            deleted(m)=j;
            m=m+1;
        end
    end
    %size(deleted,2)
    if not(isempty(deleted))
        Cl.Clc{i}.in_cluster(sort(deleted,'descend'),:)=[];
        Cl.Clc{i}.sigmas(sort(deleted,'descend'),:)=[];
    end
    Rulenum=Rulenum+ size(Cl.Clc{i}.in_cluster,1);
    deleted=[];
end
Cl.numRule=Rulenum;
Cl=evalnetwork_nfc(Cl);
%display('                       RESULT OF RULE PRUNING          ')
fprintf('Final Number of Rules is :%5.0f\n',Rulenum);
%theStr=sprintf('Final Classification Accurracy :%g',Cl.Accurracy);
%display(theStr);
%% bckprop
% back propagation algorithm for Mamdani Neuro Fuzzy Inference System.
%Calculates Delta values
function Cl=bckprop(Cl)
%{
       rule_ind: [2x1 double]5
     in_cluster: [2x4 double]
    out_cluster: [2x1 double]
         sigmas: [2x4 double]
        numData: [2x1 double]
             mf: {[135x4 double]  [135x4 double]}
    rule_output: [135x2 double]   % her bir kuralın ürettiği output
           ind2: [135x2 double]   % her bir kural tarafından ateşlenen
üyelik fonksiyonlarının numarası
           ind3:                  % sonuçta ateşlenen kuralın numarası
    rule_firing: [1x135 double]   %  sonuçta ateşlenen kural
      tr_output: [135x1 double]   % gerçek output
         output: [135x1 double]   %sistemin ürettiği output
          error: [135x1 double]   %hesaplanan error
%}
%Op(4)=Cl.Clc{n}.rule_firing
%Os(3)=Cl.Clc{n}.rule_output
%Oij(2)=Cl.Clc{n}.mf{kural}(i,j)
for i=1:Cl.class
```

```
    indx=zeros(1, length(Cl.Clc{i}.ind3));
    indxx=find( i==Cl.ind_winner | i==Cl.ind_rival);
    indx(indxx)=1;
    % Last Layer
    %dE/dY=-target/Yc
    if isfield(Cl.Clc{i},'numData')
        Cl.Clc{i}.numData(Cl.Clc{i}.numData==0)=1e-5;
    end
        (Cl.Clc{i}.tr_output-Cl.Clc{i}.rule_firing+Cl.rule_not_firing'
).*Cl.Clc{i}.indx';
        Cl.Clc{i}.dEdY=-2*...
        (Cl.Clc{i}.tr_output-Cl.Clc{i}.rule_firing);
    Cl.Clc{i}.dEdY_rival=-1*Cl.Clc{i}.dEdY;
    Cl.Clc{i}.dYdW=Cl.Clc{i}.rule_output_n;
    %Cl.Clc{i}.dEdW=sum(Cl.Clc{i}.dEdY)*sum(Cl.Clc{i}.dYdW);
    Cl.Clc{i}.dEdW=Cl.Clc{i}.dEdY'*Cl.Clc{i}.dYdW/Cl.tr_numPattern;
    Cl.Clc{i}.dEdW_rival=-1*Cl.Clc{i}.dEdW;
    Cl.Clc{i}.dYdO3=(repmat(Cl.Clc{i}.w,Cl.tr_numPattern,1)-
repmat(Cl.Clc{i}.rule_firing,1,Cl.Clc{i}.numRule))...
                    ./repmat(Cl.sum_rule,1,Cl.Clc{i}.numRule);
    for j=1:size(Cl.Clc{i}.in_cluster,1)
        Cl.Clc{i}.deriv_gauss_A{j}=...
deriv_gauss(Cl.tr_input,repmat(Cl.Clc{i}.sigmas(j,:),Cl.tr_numPattern,1),...
            repmat(Cl.Clc{i}.in_cluster(j,:),Cl.tr_numPattern,1),1); % Eqn15
        Cl.Clc{i}.deriv_gauss_S{j}=...
deriv_gauss(Cl.tr_input,repmat(Cl.Clc{i}.sigmas(j,:),Cl.tr_numPattern,1),...
            repmat(Cl.Clc{i}.in_cluster(j,:),Cl.tr_numPattern,1),2); % Eqn17
        Cl.Clc{i}.dO3dAij{j}=Cl.Clc{i}.mf{i}.*Cl.Clc{i}.deriv_gauss_A{j};
        Cl.Clc{i}.dO3dSij{j}=Cl.Clc{i}.mf{i}.*Cl.Clc{i}.deriv_gauss_S{j};
        Cl.Clc{i}.delta_s(j,:)=Cl.Clc{i}.dEdY'*...
            (
repmat(Cl.Clc{i}.dYdO3(:,j),1,Cl.numInp).*Cl.Clc{i}.dO3dSij{j})/Cl.tr_numPat
tern;
        Cl.Clc{i}.delta_c(j,:)=Cl.Clc{i}.dEdY'*...
            (
repmat(Cl.Clc{i}.dYdO3(:,j),1,Cl.numInp).*Cl.Clc{i}.dO3dAij{j})/Cl.tr_numPat
tern;
         Cl.Clc{i}.cl_c=repmat(Cl.Clc{i}.dEdY,1,Cl.numInp).*...
             (
repmat(Cl.Clc{i}.dYdO3(:,j),1,Cl.numInp).*Cl.Clc{i}.dO3dAij{j})/Cl.tr_numPat
tern;
         Cl.Clc{i}.cl_s=repmat(Cl.Clc{i}.dEdY,1,Cl.numInp).*...
             (
repmat(Cl.Clc{i}.dYdO3(:,j),1,Cl.numInp).*Cl.Clc{i}.dO3dSij{j})/Cl.tr_numPat
tern;

    end
 end

function memDeg=deriv_gauss(x,sigma,c,option)
%% [o ind_x a_3]=evalnetwork(in_all,numRule,numPts,c,s,c_out,s_out,numInp)
%computes the final Output and neccessary inner Layer outputs for Mamdani
%Neuro Fuzzy Inference System
%o     ;system output
%ind_x ;indices of Layer 2 that minimum operator chose it for Layer3
%a_3;output of Layer3;
if option==1
    memDeg = ((x-c)./(sigma.^2)).*exp(-(x - c).^2./(2*sigma.^2));
elseif option==2
    memDeg = (((x-c).^2)./(sigma.^3)).*exp(-(x - c).^2./(2*sigma.^2));
end

function [Classifier]=evalnetwork_nfc(Classifier)
%firstLayer,
%second Layer,Fuzzification and obtaining membership Value;
% j grup(NumRule) membership value that each j grup contains (numInp X
% numPts) elements
% for example a_2{4}=[ ]numPts X NumInp ; contains the membership values
for rule 4
Classifier.classes=linspace(0,1,Classifier.class);
if not(isfield(Classifier,'Clc'))
    Classifier=obtain_classifier(Classifier);
end
Classifier=computemembership(Classifier);%% Compute Membership Value
Classifier=compute_rule_out(Classifier);%% Compute Membership Value

function Cl=obtain_classifier(Cl)
display('initializin Classifiers structures.....')
maxx=max(Cl.numData);
minx=min(Cl.numData);
%Cl.numData=(Cl.numData-repmat(minx',Cl.numRule,1))./...
%   (repmat(maxx',Cl.numRule,1)  - repmat(minx',Cl.numRule,1));
n=1;
for i= Cl.classes
    Cl.Clc{n}.rule_ind=find(i==Cl.out_cluster);
    Cl.Clc{n}.in_cluster=Cl.in_cluster(Cl.Clc{n}.rule_ind,:);
    Cl.Clc{n}.out_cluster=Cl.out_cluster(Cl.Clc{n}.rule_ind,:);
    Cl.Clc{n}.sigmas=Cl.sigmas(Cl.Clc{n}.rule_ind,:);
    Cl.Clc{n}.numData=Cl.numData(Cl.Clc{n}.rule_ind);
    Cl.Clc{n}.w=Cl.Clc{n}.numData'./sum(Cl.Clc{n}.numData);
    n=n+1;
end

%% Compute Membership Value
function [Cl]=computemembership(Cl)
for n=1:Cl.class
    Cl.Clc{n}.mf=[];
    for m=1:size(Cl.Clc{n}.in_cluster,1)
Cl.Clc{n}.mf{m}=gaussneuron(Cl.tr_input,repmat(Cl.Clc{n}.sigmas(m,:),Cl.tr_n
umPattern,1),...
            repmat(Cl.Clc{n}.in_cluster(m,:),Cl.tr_numPattern,1));
    end
end

for n=1:Cl.class
    for m=1:size(Cl.Clc{n}.in_cluster,1)
Cl.Clc{n}.mf_t{m}=gaussneuron(Cl.ts_input,repmat(Cl.Clc{n}.sigmas(m,:),Cl.ts
_numPattern,1),...
            repmat(Cl.Clc{n}.in_cluster(m,:),Cl.ts_numPattern,1));
    end
end

%% Gaussian Membership  Function
function memDeg=gaussneuron(x,sigma,c)
memDeg = exp(-(x - c).^2./(2*sigma.^2));

%% Rule firing
function [Cl]=compute_rule_out(Cl)
Cl.tr_rule_output=zeros(Cl.tr_numPattern,Cl.numRule);
tr_ind=zeros(Cl.tr_numPattern,1);
rule_outs=zeros(Cl.class,Cl.tr_numPattern);
Cl.ts_rule_output_ts=zeros(Cl.ts_numPattern,Cl.numRule);
ts_ind=zeros(Cl.ts_numPattern,1);
rule_outs_ts=zeros(Cl.class,Cl.ts_numPattern);
for n= 1:Cl.class
    Cl.Clc{n}.rule_output=[];
    Cl.Clc{n}.ind2=[];
    Cl.Clc{n}.ind3=[];
    Cl.Clc{n}.rule_firing=[];
    Cl.Clc{n}.rule_output_ts=[];
    Cl.Clc{n}.ind2_ts=[];
    Cl.Clc{n}.ind3_ts=[];
    Cl.Clc{n}.rule_firing_ts=[];
    Cl.Clc{n}.numRule=size(Cl.Clc{n}.mf,2);
  %  LAYER3
```

```
     for m=1:Cl.Clc{n}.numRule
        %Os(3)=Cl.Clc{n}.rule_output
        [a b]=min(Cl.Clc{n}.mf{m}'); %%%%% Burası productda
olabilirrrrrrrrrrrrrrrrrrrr
        Cl.Clc{n}.rule_output(:,m)=a';%Her bir class daki her bir kuralın
firing strengini bul
        Cl.Clc{n}.ind2(:,m)=b';%:hangi membership tetikledi onu bul
    end
end
%Find total firing streng
%LAYER4
sum_rule_output=zeros(Cl.tr_numPattern,1);
for n= 1:Cl.class
sum_rule_output=sum_rule_output+sum(Cl.Clc{n}.rule_output')';
end
sum_rule_output(sum_rule_output==0)=1e-2;
Cl.sum_rule=sum_rule_output;

% normalize the frinsgs and compute weighted output
rule_outs=zeros(Cl.tr_numPattern,Cl.class);
for n= 1:Cl.class

Cl.Clc{n}.rule_output_n=Cl.Clc{n}.rule_output./repmat(sum_rule_output,1,Cl.C
lc{n}.numRule);
%LAYER 5
Cl.Clc{n}.rule_firing=sum((Cl.Clc{n}.rule_output_n.*repmat(Cl.Clc{n}.w,Cl.tr
_numPattern,1))')';
rule_outs(:,n)=Cl.Clc{n}.rule_firing;
end
rule_outs2=rule_outs;
[maxx Cl.ind_winner] =max(rule_outs');
for t=1:Cl.tr_numPattern
    rule_outs2(t,Cl.ind_winner(t))=0;
end
[Cl.rule_not_firing  Cl.ind_rival]=max(rule_outs2');
Cl.output=zeros(Cl.tr_numPattern,Cl.class);
Cl.output_ts=zeros(Cl.ts_numPattern,Cl.class);
Cl.RMSE=0;
for i=1:Cl.class
    Cl.Clc{i}.tr_output=zeros(Cl.tr_numPattern,1);
    Cl.output(i==Cl.ind_winner,i)=1;
    Cl.Clc{i}.output=Cl.output(:,i);
    Cl.Clc{i}.tr_output(Cl.tr_output==Cl.classes(i))=1;
    Cl.tr_target(:,i)=Cl.Clc{i}.tr_output;
    indx=zeros(1, length(Cl.tr_output));
    indxx= i==Cl.ind_winner | i==Cl.ind_rival;
    indx(indxx)=1;
    Cl.Clc{i}.indx=indx;
    Cl.Clc{i}.error=.5*...
        ((Cl.Clc{i}.tr_output-
Cl.Clc{i}.rule_firing+Cl.rule_not_firing').^2).*Cl.Clc{i}.indx';
    Cl.RMSE= sqrt(sum(sum(Cl.Clc{i}.error.^2)))+Cl.RMSE;
end
Cl.RMSE=Cl.RMSE/Cl.tr_numPattern;
a=Cl.tr_target&Cl.output;
b=Cl.ts_target&Cl.output_ts;
a=size(find(a==1),1);
b=size(find(b==1),1);
Cl.Accurracy=100*a/Cl.tr_numPattern;
Cl.Accurracy_ts=100*b/Cl.ts_numPattern;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
NFC2.m
NeuroFuzzy Classifier2
function [Classifier Ac1]=NFC2(problem)
%Neurofuzzy Classifier based on Rival Penalized Competitive learning
%and  incremental type GD RPCL
%%NFC2(problem)
%Yunus TORUN,Gaziantep Univeristy M.Y.O
%torun@gantep.edu.tr
%May 2009,Gaziantep
%June 2009,Similarity Measurement is added
clc
close all
vis=1;
screen=1;
teta=.01;
error_bound=.5;
fring_bound=1;
train_opt = [1e-5  1e-5 1e-5   50];
N=10;% N  fold  validation
warning off
model=1;
load result_rival50
if nargin<1
    problem=7;
end
data=result_rival{model,problem}.data;
display(result_rival{model,problem}.name)
tolerance = train_opt(1);   % Stop learning once RMSE is below tolerance
eta1 = train_opt(2);        % Learning rate
eta2=train_opt(3);
max_epoch = train_opt(4);   % Max. training epochs
indices = crossvalind('Kfold',data(:,end),N);
for optStep=1:N
    max_epoch = train_opt(4);
    display(' ');
    display(' ');
    display('****************************************************')
    fprintf('>>>>>>>-------   FOLD   %5.0f  -----<<<<<<<\n',optStep)
    display(' ');
    testx = (indices == optStep);
    trainx = ~testx ;
    %trainx = (indices == optStep);
    data_train=data(trainx,:);
    data_test=data(testx,:);
    epoch=1;
    %% obtaining Classifer parameters
    clear Classifier
    if problem==7 || problem==8||problem==9||problem==10
        Classifier.class=3;
    else
        Classifier.class=2;

    end
    Classifier.tr_input=data_train(:,1:end-1);
    Classifier.tr_output=data_train(:,end);
    Classifier.ts_input=data_test(:,1:end-1);
    Classifier.ts_output=data_test(:,end);
    Classifier.numData=result_rival{model,problem,1}.numData_in_cl;

Classifier.in_cluster=abs(result_rival{model,problem}.cluster{1}(:,1:end-
1));

Classifier.out_cluster=abs(result_rival{model,problem}.cluster{1}(:,end));
    Classifier.numRule=size(Classifier.out_cluster,1);
    [Classifier.tr_numPattern Classifier.numInp]=size(Classifier.tr_input);
    [Classifier.ts_numPattern Classifier.numInp]=size(Classifier.ts_input);
    Classifier.sigmas=result_rival{model,problem}.sigmas(:,1:end-1);
    Classifier=evalnetwork_nfc(Classifier);
    numRule1=Classifier.numRule;
    RMSE(1)=Classifier.RMSE;
    %Classifier=remove_redund(Classifier,teta,RMSE,error_bound,vis);
    if vis  figure_number=1; display_result(Classifier,figure_number,RMSE);
end;

    %%
    fnished=0;
    fnished2=0;
    epoch=2;
    %pause
    while fnished==0;
        %Classifier=evalnetwork_nfc(Classifier);
        %display(size(Classifier.Clc{2}.in_cluster,1))
        RMSE(epoch)=Classifier.RMSE;
        Classifier =bckprop(Classifier);
        if RMSE(epoch) < tolerance
            fnished = 1;
        end
        if epoch == max_epoch
            if fnished2
                display('                       LEARNING FNISHED          ')
                fprintf('Final Number of Rules is
=%5.0f\n',Classifier.numRule);
                fprintf('Final RMSE  =%5.7f\n',RMSE(epoch));
                fprintf('Final Classification Accurracy: Training =%g,
Testing =%g\n',Classifier.Accurracy,Classifier.Accurracy_ts);
                %display_result(Classifier,figure_number,RMSE)

                Ac1.training(optStep)=Classifier.Accurracy;
                Ac1.testing(optStep)=Classifier.Accurracy_ts;
                Classifier=pruning(Classifier,fring_bound);
                Classifier=evalnetwork_nfc(Classifier);
                fprintf('Final Classification Accurracy: Training =%g,
Testing =%g\n',Classifier.Accurracy,Classifier.Accurracy_ts);
                fnished=1;
                epoch=2;
                break;
            end

            figure_number=2;
            if vis display_result(Classifier,figure_number,RMSE); end;
            display('Second Phase for Structure Re Organazingggg...... ');
            %Classifier=pruning(Classifier,fring_bound);
            Classifier=remove_redund(Classifier,teta,RMSE,error_bound,vis);
            Classifier=pruning(Classifier,fring_bound);
            fprintf('Final Accuraccy before second Back Propagation.
Training =%5.5f: Testing =%5.5f: Final RMSE = %.10f:F.Rule=%2.0f:\n',...

Classifier.Accurracy,Classifier.Accurracy_ts,Classifier.RMSE,Classifier.numR
ule);
            display('Re Learning with backpropagation..................');
            fnished2=1;
            %eta1=eta1/2;
            %eta2=eta2/2;
            max_epoch=train_opt(4)*2;
        end
        if screen
            fprintf('Ep
%.0f:RMSE=%.5f:TrainA=%5.2f,TestA=%5.2f,eta1=%2.2e:eta2=%2.2e\n',...
                epoch,
RMSE(epoch),Classifier.Accurracy,Classifier.Accurracy_ts,eta1,eta2);
        end
        leng=length(RMSE);
        if leng==4
```

```
            if(RMSE(2)-RMSE(3))<1e-4
                eta1=10*eta1;
                eta2=10*eta2;
            end
            if(RMSE(3)-RMSE(4))<1e-4
                eta1=10*eta1;
                eta2=10*eta2;
            end
        end
        if leng>6
            if(RMSE(epoch)-RMSE(epoch-1))>1e-5
                eta1=eta1*.95;
                eta2=eta2*.95;
            end
            if mean(diff(RMSE(leng-5:leng)))<-0.001
                eta1=eta1*1.1;
                eta2=eta2*1.1;

            elseif abs(mean(diff(RMSE(leng-5:leng))))<1e-10
                eta1=eta1*.95;
                eta2=eta2*.95;
            else
            end
        end
        for n=1:Classifier.tr_numPattern
            %  if n==71 ;display('Problemli Data');end
            ind2=Classifier.Clc{Classifier.ind_winner(n)}.ind2(n,:);
            ind3=Classifier.Clc{Classifier.ind_winner(n)}.ind3(n);
            s_old_winner=
Classifier.Clc{Classifier.ind_winner(n)}.sigmas(ind3,ind2(ind3));
            c_old_winner=
Classifier.Clc{Classifier.ind_winner(n)}.in_cluster(ind3,ind2(ind3));

            ind2r=Classifier.Clc{Classifier.ind_rival(n)}.ind2(n,:);
            ind3r=Classifier.Clc{Classifier.ind_rival(n)}.ind3(n);
            s_old_rival=
Classifier.Clc{Classifier.ind_rival(n)}.sigmas(ind3r,ind2r(ind3r));
            c_old_rival=
Classifier.Clc{Classifier.ind_rival(n)}.in_cluster(ind3r,ind2r(ind3r));


            s_new=s_old_winner -(1-
n/(Classifier.tr_numPattern*3))*eta1*Classifier.Clc{Classifier.ind_winner(n)
}.delta_s(n);
            c_new=c_old_winner- (1-
n/(Classifier.tr_numPattern*3))*eta2*Classifier.Clc{Classifier.ind_winner(n)
}.delta_c(n);

            s_new_rival=s_old_rival -(1-
n/(Classifier.tr_numPattern*3))*(eta1/2)*Classifier.Clc{Classifier.ind_winne
r(n)}.delta_s_rival(n);
            c_new_rival=c_old_rival-(1-
n/(Classifier.tr_numPattern*3))*(eta2/2)*Classifier.Clc{Classifier.ind_winne
r(n)}.delta_c_rival(n);
            % winner phase
Classifier.Clc{Classifier.ind_winner(n)}.sigmas(ind3,ind2(ind3))=s_new;
Classifier.Clc{Classifier.ind_winner(n)}.in_cluster(ind3,ind2(ind3))=c_new;
            % rival phase
Classifier.Clc{Classifier.ind_rival(n)}.sigmas(ind3r,ind2r(ind3r))=s_new_riv
al;
Classifier.Clc{Classifier.ind_rival(n)}.in_cluster(ind3r,ind2r(ind3r))=c_new
_rival;
        end
        if vis ;figure_number=2;
display_result(Classifier,figure_number,RMSE) ;
figure_number=3;Classifier=plot_mf(Classifier,figure_number,RMSE,optStep);
            Classifier=evalnetwork_nfc(Classifier);
        end
        epoch=epoch+1;
    end

end
Classifier=similarity(Classifier);
Classifier=evalnetwork_nfc(Classifier);
fprintf('Training Accurraccy:%5.5f  \n',Classifier.Accurracy);
fprintf('Testing Accurraccy:%5.5f    std=:%5.5f\n',Classifier.Accurracy_ts);
save Classifier Classifier
display('************************************************')
display('All fold is completed.........................')
fprintf('Final Training Accurraccy:%5.5f  std=:%5.5f
\n',mean(Ac1.training),std(Ac1.training));
fprintf('Final Testing Accurraccy:%5.5f
std=:%5.5f\n',mean(Ac1.testing),std(Ac1.testing));
%%
function display_result(Classifier,figure_number,RMSE)
figure(figure_number);
if Classifier.class==3
 subplot(4,1,1),plot(1:Classifier.tr_numPattern,Classifier.Clc{1}.error,...
        1:Classifier.tr_numPattern,Classifier.Clc{2}.error,...
        1:Classifier.tr_numPattern,Classifier.Clc{3}.error)
    Ylabel('Classifier Error') ;
    Xlabel('Input Pattern') ;
    theStr=sprintf(' Classifier Accurracy=   %g',Classifier.Accurracy);
    Title(theStr,'Color','b','FontSize',12)

subplot(4,1,2),plot(1:Classifier.tr_numPattern,Classifier.Clc{1}.output,...
        1:Classifier.tr_numPattern,Classifier.Clc{2 }.output,...
        1:Classifier.tr_numPattern,Classifier.Clc{3}.output);
    Ylabel('Final Classifier Output ') ;
    Xlabel('Input Pattern') ;
    subplot(4,1,3),plot([Classifier.Clc{1}.rule_firing'
Classifier.Clc{2}.rule_firing'...
        Classifier.Clc{3}.rule_firing'])
    Ylabel(' Classifier Output ') ;
    Xlabel('Input Pattern') ;
    subplot(4,1,4),plot(RMSE);
    Ylabel(' RMSE ') ;
    Xlabel('Number of iteration') ;
    set(0,'Units','pixels')
    scnsize = get(0,'ScreenSize');
    figure(figure_number);
    position=[5  5       scnsize(3)/3  scnsize(4)/1.1  ];
    set(figure(figure_number),'Position',position)
    pause(.01)
else if Classifier.class==2
subplot(4,1,1),plot(1:Classifier.tr_numPattern,Classifier.Clc{1}.error,...
            1:Classifier.tr_numPattern,Classifier.Clc{2 }.error);
        Ylabel('Classifier Error') ;
        Xlabel('Input Pattern') ;
        theStr=sprintf(' Classifier Accurracy=   %g',Classifier.Accurracy);
        Title(theStr,'Color','b','FontSize',12);
 subplot(4,1,2),plot(1:Classifier.tr_numPattern,Classifier.Clc{1}.output,...
            1:Classifier.tr_numPattern,Classifier.Clc{2 }.output);
        Ylabel('Final Classifier Output ') ;
        Xlabel('Input Pattern') ;
        subplot(4,1,3),plot([Classifier.Clc{1}.rule_firing'
Classifier.Clc{2}.rule_firing']);
        Ylabel(' Classifier Output ') ;
        Xlabel('Input Pattern') ;
        subplot(4,1,4),plot(RMSE);
        Xlabel(num2str(Classifier.Accurracy));
        Ylabel(' RMSE ') ;
        Xlabel('Number of iteration') ;
        pause(.01)

    end
```

```
end
%%
%Graphical representation of dynamics of network
function Classifier=plot_mf(Classifier,figure_number,RMSE,optStep)
set(0,'Units','pixels')
scnsize = get(0,'ScreenSize');
for j=1:Classifier.class
    set(0,'Units','pixels')
    scnsize = get(0,'ScreenSize');
    figure(figure_number);
    position=get(figure(figure_number),'Position');
    position=[scnsize(3)/1.5   (j-1)*scnsize(4)/3     scnsize(3)/3
scnsize(4)/4  ];
    set(figure(figure_number),'Position',position)
    x=linspace(0,1,100);
    n=size(Classifier.Clc{j}.in_cluster,1);
    for i=1:4
        memDeg{j}{i} = exp(-((repmat(x,n,1) -
repmat(Classifier.Clc{j}.in_cluster(:,i),1,100)).^2)./(2*repmat(Classifier.C
lc{j}.sigmas(:,i),1,100).^2));
        subplot(2,2,i),plot(x,memDeg{j}{i})
        theStr=sprintf(' Input=   %g',i);
        Ylabel('Mebership Degree ') ;
        Xlabel(theStr) ;
    end
    figure_number=figure_number+1;
end
Classifier.memDeg=memDeg;
pause(.01);
%%
%Membership Function Ploting
function Classifier=remove_redund(Classifier,teta,RMSE,error_bound,vis)
if nargin<2
    teta=.1;
end
c_inc=0;
m_init=0;
for t=1:Classifier.class
    clear empty_index empty_index1 a empty_index2
    empty_index=find(Classifier.Clc{t}.error>=error_bound);
    Classifier.tr_input(empty_index,:);
    m=size(Classifier.tr_input(empty_index,:),1);
    if m>0
        dist=zeros(m,m);
        inc=1;
        for i=1:m
            for j=1:m
                dist(i,j)=sum((Classifier.tr_input(i,:)-
Classifier.tr_input(j,:)).^2);
            end
            a{i}=find(dist(i,:)<teta);

            if size(a{i},2)==1
                empty_index1(i)=empty_index(i);
                cmean(i,:) =Classifier.tr_input(empty_index(i),:);
            else
                empty_index1(i)=empty_index(a{i}(1));
                cmean(i,:)=mean(Classifier.tr_input(a{i},:));
                % dist(i,a{i})=10;
            end
        end
        %empty_index3=empty_index1;
        empty_index2=[];
        inc=1;

        for i=1:size(empty_index1,2)
            indx=find(empty_index1(i)==empty_index1);
            empty_index2(inc)=empty_index1(indx(1));
            empty_index1(indx)=0;
            inc=inc+1;
        end
        empty_index3=find(empty_index2==0)';
        empty_index3=sort(empty_index3,'descend');
        cmean(empty_index3,:)=[];
        empty_index2=empty_index2';
        empty_index2=sort(empty_index2,'descend');
        empty_index2(empty_index2==0)=[];
        c_add=size(empty_index2,1);
        c_size=size(Classifier.Clc{t}.in_cluster,1);
        if not(isempty(c_size+1:c_size+c_add))
            Classifier.Clc{t}.in_cluster(c_size+1:c_size+c_add,:)=...
                Classifier.tr_input(empty_index2,:);
            %      Classifier.Clc{t}.in_cluster(c_size+1:c_size+c_add,:)=...
            %         cmean;
            Classifier.Clc{t}.sigmas(c_size+1:c_size+c_add,:)=...
                0.05*ones(c_add,Classifier.numInp);
        end
        c_inc=c_add+c_inc;
        m_init=m_init+m;
    end

end
Classifier.numRule=Classifier.numRule+c_inc;
Classifier=evalnetwork_nfc(Classifier);
figure_number=3;
if vis display_result(Classifier,figure_number,RMSE); end
display('                       RESULT OF STRUCTURE REORGANAZING          ')
fprintf('%5.0f: Rules are added  because of the Error\n',m_init);
fprintf('%5.0f: Rules are removed  because of the Similarity\n',m_init-
c_inc);
%%
%Remove Reduntand cluster and rules
function Cl=pruning(Cl,fring_bound)
deleted=[];
Rulenum=0;
for i=1:Cl.class
    numRule=size(Cl.Clc{i}.in_cluster,1);
    m=1;
    for j=1:numRule
        if size(find(j==(Cl.Clc{i}.ind3)),2)<fring_bound

            deleted(m)=j;
            m=m+1;
        end
    end
    %size(deleted,2)
    if not(isempty(deleted))
        Cl.Clc{i}.in_cluster(sort(deleted,'descend'),:)=[];
        Cl.Clc{i}.sigmas(sort(deleted,'descend'),:)=[];
    end
    Rulenum=Rulenum+ size(Cl.Clc{i}.in_cluster,1);
    deleted=[];
end
Cl.numRule=Rulenum;
Cl=evalnetwork_nfc(Cl);
%display('                       RESULT OF RULE PRUNING          ')
fprintf('Final Number of Rules is :%5.0f\n',Rulenum);
%theStr=sprintf('Final Classification Accurracy :%g',Cl.Accurracy);
%display(theStr);
%%
% Rule Pruning strategy
function Cl=bckprop(Cl)
% back propagation algorithm for Mamdani Neuro Fuzzy Inference System.
%Calculates Delta values
%{
       rule_ind: [2x1 double]5
```

```
     in_cluster: [2x4 double]
    out_cluster: [2x1 double]
         sigmas: [2x4 double]
        numData: [2x1 double]
             mf: {[135x4 double]  [135x4 double]}
    rule_output: [135x2 double]   % her bir kuralın ürettiği output
           ind2: [135x2 double]   % her bir kural tarafından ateşlenen
üyelik fonksiyonlarının numarası
           ind3:                  % sonuçta ateşlenen kuralın numarası
    rule_firing: [1x135 double]   %  sonuçta ateşlenen kural
      tr_output: [135x1 double]   % gerçek output
         output: [135x1 double]   %sistemin ürettiği output
          error: [135x1 double]   %hesaplanan error
%}
%Op(4)=Cl.Clc{n}.rule_firing
%Os(3)=Cl.Clc{n}.rule_output
%Oij(2)=Cl.Clc{n}.mf{kural}(i,j)
for i=1:Cl.class
    indx=zeros(1, length(Cl.Clc{i}.ind3));
    indxx=find( i==Cl.ind_winner | i==Cl.ind_rival);
    indx(indxx)=1;
    % Last Layer
    %dE/dY=-target/Yc
    if isfield(Cl.Clc{i},'numData')
        Cl.Clc{i}.numData(Cl.Clc{i}.numData==0)=1e-5;
        %Cl.Clc{i}.dEdY=-Cl.Clc{i}.tr_output'.*(1./Cl.Clc{i}.rule_firing-
Cl.Clc{i}.rule_firing);
    end
    %Cl.Clc{i}.dEdY=-1*Cl.Clc{i}.tr_output'.*...
    %(Cl.Clc{i}.tr_output'-
Cl.Clc{i}.rule_firing+Cl.rule_not_firing)*Cl.Accurracy/100;
    Cl.Clc{i}.dEdY=-5*...
        (Cl.Clc{i}.tr_output'-
Cl.Clc{i}.rule_firing+Cl.rule_not_firing).*indx;
    Cl.Clc{i}.dEdY_rival=-1*Cl.Clc{i}.dEdY;
    %else
    %    Cl.Clc{i}.dEdY=0;
    %end

    for j=1:size(Cl.Clc{i}.in_cluster,1)
        ind=Cl.Clc{i}.ind3==j;

%Cl.Clc{i}.dEdW(j)=sum(Cl.Clc{i}.dEdY(ind)*Cl.Clc{i}.numData(j))/Cl.tr_numPa
ttern;
        Cl.Clc{i}.deriv_gauss_A{j}=...

deriv_gauss(Cl.tr_input,repmat(Cl.Clc{i}.sigmas(j,:),Cl.tr_numPattern,1),...
            repmat(Cl.Clc{i}.in_cluster(j,:),Cl.tr_numPattern,1),1); % Eqn15
        Cl.Clc{i}.deriv_gauss_S{j}=...

deriv_gauss(Cl.tr_input,repmat(Cl.Clc{i}.sigmas(j,:),Cl.tr_numPattern,1),...
            repmat(Cl.Clc{i}.in_cluster(j,:),Cl.tr_numPattern,1),2); % Eqn17

    end

    %Cl.Clc{i}.delta_s=Cl.Clc{i}.dEdY*Cl.Clc{i}.delta_s';
    %Cl.Clc{i}.delta_c=Cl.Clc{i}.dEdY*Cl.Clc{i}.delta_c';
end

for i=1:Cl.class
    for n=1:Cl.tr_numPattern
        Cl.Clc{i}.delta_s(n)=Cl.Clc{i}.dEdY(n)*...
            Cl.Clc{i}.deriv_gauss_S{Cl.Clc{i}.ind3(n)}(n,Cl.Clc{i}.ind2(n));
        Cl.Clc{i}.delta_c(n)=Cl.Clc{i}.dEdY(n)*...
            Cl.Clc{i}.deriv_gauss_A{Cl.Clc{i}.ind3(n)}(n,Cl.Clc{i}.ind2(n));

        Cl.Clc{i}.delta_s_rival(n)=Cl.Clc{i}.dEdY_rival(n)*...
Cl.Clc{Cl.ind_rival(n)}.deriv_gauss_S{Cl.Clc{Cl.ind_rival(n)}.ind3(n)}(n,Cl.
Clc{Cl.ind_rival(n)}.ind2(n));
        Cl.Clc{i}.delta_c_rival(n)=Cl.Clc{i}.dEdY_rival(n)*...

Cl.Clc{Cl.ind_rival(n)}.deriv_gauss_A{Cl.Clc{Cl.ind_rival(n)}.ind3(n)}(n,Cl.
Clc{Cl.ind_rival(n)}.ind2(n));

    end
end
%%
%back propagation algorithm for Mamdani Neuro Fuzzy Inference System.
%Calculates Delta values
function memDeg=deriv_gauss(x,sigma,c,option)
%% [o ind_x a_3]=evalnetwork(in_all,numRule,numPts,c,s,c_out,s_out,numInp)
%computes the final Output and neccessary inner Layer outputs for Mamdani
%Neuro Fuzzy Inference System
%o     ;system output
%ind_x ;indices of Layer 2 that minimum operator chose it for Layer3
%a_3;output of Layer3;
if option==1
    memDeg = ((x-c)./(sigma.^2)).*exp(-(x - c).^2./(2*sigma.^2));
elseif option==2
    memDeg = (((x-c).^2)./(sigma.^3)).*exp(-(x - c).^2./(2*sigma.^2));
end
%%
%Derivative of gaussian function
function [Classifier]=evalnetwork_nfc(Classifier)
%firstLayer,
%second Layer,Fuzzification and obtaining membership Value;
% j grup(NumRule) membership value that each j grup contains (numInp X
% numPts) elements
% for example a_2{4}=[ ]numPts X NumInp ; contains the membership values
for rule 4
Classifier.classes=linspace(0,1,Classifier.class);
if not(isfield(Classifier,'Clc'))
    Classifier=obtain_classifier(Classifier);
end
Classifier=computemembership(Classifier);%% Compute Membership Value
Classifier=compute_rule_out(Classifier);%% Compute Membership Value
%%
%Evulate network
function Cl=obtain_classifier(Cl)
display('initializin Classifiers structures.....')
maxx=max(Cl.numData);
minx=min(Cl.numData);
%Cl.numData=(Cl.numData-repmat(minx',Cl.numRule,1))./...
%   (repmat(maxx',Cl.numRule,1)  - repmat(minx',Cl.numRule,1));

n=1;
Cl.input_select=ones(Cl.class,Cl.numInp);
Cl.input_select=[1 1 1 1;1 1 1 1; 1 1 1 1];
for i= Cl.classes
    Cl.Clc{n}.rule_ind=find(i==Cl.out_cluster);
    Cl.Clc{n}.in_cluster=Cl.in_cluster(Cl.Clc{n}.rule_ind,:);
    Cl.Clc{n}.out_cluster=Cl.out_cluster(Cl.Clc{n}.rule_ind,:);
    Cl.Clc{n}.sigmas=Cl.sigmas(Cl.Clc{n}.rule_ind,:);
    % Cl.Clc{n}.numData=Cl.numData(Cl.Clc{n}.rule_ind);
    n=n+1;
end

%%
% Obtain Structure
function [Cl]=computemembership(Cl)
if isfield(Cl.Clc{1},'two_c')
    for n=1:Cl.class
        Cl.Clc{n}.mf=[];
        for m=1:size(Cl.Clc{n}.two_c,1)
```

```
Cl.Clc{n}.mf{m}=gauss2neuron(Cl.tr_input(:,Cl.input_select(1,:)==1),...

repmat(Cl.Clc{n}.two_s(m,repmat(Cl.input_select(1,:),1,2)==1),Cl.tr_numPatte
rn,1),...

repmat(Cl.Clc{n}.two_c(m,repmat(Cl.input_select(1,:),1,2)==1),Cl.tr_numPatte
rn,1),2*Cl.numInp);
        end
    end

    for n=1:Cl.class
        for m=1:size(Cl.Clc{n}.in_cluster,1)

Cl.Clc{n}.mf_t{m}=gauss2neuron(Cl.ts_input(:,Cl.input_select(1,:)==1),...

repmat(Cl.Clc{n}.two_s(m,repmat(Cl.input_select(1,:),1,2)==1),Cl.ts_numPatte
rn,1),...

repmat(Cl.Clc{n}.two_c(m,repmat(Cl.input_select(1,:),1,2)==1),Cl.ts_numPatte
rn,1),2*Cl.numInp);
        end
    end
else
    for n=1:Cl.class
        Cl.Clc{n}.mf=[];
        for m=1:size(Cl.Clc{n}.in_cluster,1)

Cl.Clc{n}.mf{m}=gaussneuron(Cl.tr_input(:,Cl.input_select(1,:)==1),...

repmat(Cl.Clc{n}.sigmas(m,Cl.input_select(1,:)==1),Cl.tr_numPattern,1),...

repmat(Cl.Clc{n}.in_cluster(m,Cl.input_select(1,:)==1),Cl.tr_numPattern,1));
        end
    end
    for n=1:Cl.class
        for m=1:size(Cl.Clc{n}.in_cluster,1)
Cl.Clc{n}.mf_t{m}=gaussneuron(Cl.ts_input(:,Cl.input_select(1,:)==1),...
repmat(Cl.Clc{n}.sigmas(m,Cl.input_select(1,:)==1),Cl.ts_numPattern,1),...
repmat(Cl.Clc{n}.in_cluster(m,Cl.input_select(1,:)==1),Cl.ts_numPattern,1));
        end
    end
end
%}
%%
%Compute Membership Value
function memDeg=gaussneuron(x,sigma,c)
memDeg = exp(-(x - c).^2./(2*sigma.^2));
%%
%Gaussian Membership  Function
function memDeg=gauss2neuron(x,sig,c2,nInp)
clow=c2(:,1:nInp/2);
cup=c2(:,nInp/2+1:nInp);
slow=sig(:,1:nInp/2);
sup=sig(:,nInp/2+1:nInp);
c1Index=(x<=clow);
c2Index=(x>=cup);
y1 = exp(-(x-clow).^2./(2*slow.^2)).*c1Index + (1-c1Index);
y2 = exp(-(x-cup).^2./(2*sup.^2)).*c2Index + (1-c2Index);
memDeg = y1.*y2;
% Two sided Gaussian Membership  Function
function [Cl]=compute_rule_out(Cl)
%% training phase
Cl.tr_rule_output=zeros(Cl.tr_numPattern,Cl.numRule);
tr_ind=zeros(Cl.tr_numPattern,1);
rule_outs=zeros(Cl.class,Cl.tr_numPattern);
Cl.ts_rule_output_ts=zeros(Cl.ts_numPattern,Cl.numRule);
ts_ind=zeros(Cl.ts_numPattern,1);
```

```
rule_outs_ts=zeros(Cl.class,Cl.ts_numPattern);

for n= 1:Cl.class
    Cl.Clc{n}.rule_output=[];
    Cl.Clc{n}.ind2=[];
    Cl.Clc{n}.ind3=[];
    Cl.Clc{n}.rule_firing=[];
    Cl.Clc{n}.rule_output_ts=[];
    Cl.Clc{n}.ind2_ts=[];
    Cl.Clc{n}.ind3_ts=[];
    Cl.Clc{n}.rule_firing_ts=[];

    for m=1:size(Cl.Clc{n}.mf,2)
        %Os(3)=Cl.Clc{n}.rule_output
        [a b]=min(Cl.Clc{n}.mf{m}');
        Cl.Clc{n}.rule_output(:,m)=a';%Her bir class daki her bir kuralın
firing strengini bul
        Cl.Clc{n}.ind2(:,m)=b';%:hangi membership tetikledi onu bul
    end
    % hangi kural ateşlendi onu bul
    %Op(4)=Cl.Clc{n}.rule_firing
    if size(Cl.Clc{n}.rule_output,2)>1
        [Cl.Clc{n}.rule_firing Cl.Clc{n}.ind3]=max(Cl.Clc{n}.rule_output');
        %Cl.Clc{n}.rule_firing=Cl.Clc{n}.rule_firing.*
Cl.Clc{n}.numData(Cl.Clc{n}.ind3)';
    else
        %
Cl.Clc{n}.rule_firing=Cl.Clc{n}.rule_output'.*repmat(Cl.Clc{n}.numData,1,Cl.
tr_numPattern);
        Cl.Clc{n}.ind3=ones(1,Cl.tr_numPattern);
        Cl.Clc{n}.rule_firing=Cl.Clc{n}.rule_output';
    end

    for m=1:size(Cl.Clc{n}.mf_t,2)
        %Os(3)=Cl.Clc{n}.rule_output
        [a b]=min(Cl.Clc{n}.mf_t{m}');
        Cl.Clc{n}.rule_output_ts(:,m)=a';%Her bir class daki her bir kuralın
firing strengini bul
        Cl.Clc{n}.ind2_ts(:,m)=b';%:hangi membership tetikledi onu bul
    end

    if size(Cl.Clc{n}.rule_output_ts,2)>1
        [Cl.Clc{n}.rule_firing_ts
Cl.Clc{n}.ind3_ts]=max(Cl.Clc{n}.rule_output_ts');

    else
        Cl.Clc{n}.ind3_ts=ones(1,Cl.tr_numPattern);
        Cl.Clc{n}.rule_firing_ts=Cl.Clc{n}.rule_output_ts';
    end
    rule_outs_ts(n,:)=Cl.Clc{n}.rule_firing_ts;
    rule_outs(n,:)=Cl.Clc{n}.rule_firing;
end

rule_outs2=rule_outs;
[maxx Cl.ind_winner] =max(rule_outs);
rule_outs2_ts=rule_outs_ts;
[maxx Cl.ind_winner_ts] =max(rule_outs_ts);

for t=1:Cl.tr_numPattern
    rule_outs2(Cl.ind_winner(t),t)=0;
end
[Cl.rule_not_firing  Cl.ind_rival]=max(rule_outs2);

for t=1:Cl.ts_numPattern
    rule_outs2_ts(Cl.ind_winner_ts(t),t)=0;
end
[Cl.rule_not_firing_ts  Cl.ind_rival_ts]=max(rule_outs2_ts);
Cl.output=zeros(Cl.tr_numPattern,Cl.class);
Cl.output_ts=zeros(Cl.ts_numPattern,Cl.class);
Cl.RMSE=0;
for i=1:Cl.class
    Cl.Clc{i}.tr_output=zeros(Cl.tr_numPattern,1);
    Cl.output(i==Cl.ind_winner,i)=1;
    Cl.Clc{i}.output=Cl.output(:,i);
    Cl.Clc{i}.tr_output(Cl.tr_output==Cl.classes(i))=1;
    Cl.tr_target(:,i)=Cl.Clc{i}.tr_output;
    Cl.Clc{i}.rule_firing(Cl.Clc{i}.rule_firing==0)=1e-10;

    Cl.Clc{i}.ts_output=zeros(Cl.ts_numPattern,1);
    Cl.output_ts(i==Cl.ind_winner_ts,i)=1;
    Cl.Clc{i}.output_ts=Cl.output_ts(:,i);
    Cl.Clc{i}.ts_output(Cl.ts_output==Cl.classes(i))=1;
    Cl.ts_target(:,i)=Cl.Clc{i}.ts_output;
    Cl.Clc{i}.rule_firing_ts(Cl.Clc{i}.rule_firing_ts==0)=1e-10;


    indx=zeros(1, length(Cl.Clc{i}.ind3));
    indxx= i==Cl.ind_winner | i==Cl.ind_rival;
    indx(indxx)=1;
    Cl.Clc{i}.error=.5*...
        ((Cl.Clc{i}.tr_output-
Cl.Clc{i}.rule_firing'+Cl.rule_not_firing').^2).*indx';

    Cl.RMSE= sqrt(sum(sum(Cl.Clc{i}.error)))+Cl.RMSE;
end
Cl.RMSE=Cl.RMSE/Cl.tr_numPattern;
a=Cl.tr_target&Cl.output;
b=Cl.ts_target&Cl.output_ts;
a=size(find(a==1),1);
b=size(find(b==1),1);

Cl.Accurracy=100*a/Cl.tr_numPattern;
Cl.Accurracy_ts=100*b/Cl.ts_numPattern;
%%
%%Calculate Rule firing & network output
function Cl=similarity(Cl)
UpThr=1;
LowThr=.87;
k=1;
n_class=Cl.class;
k=1;
figure_number=1;
for j=1:Cl.class
    set(0,'Units','pixels')
    scnsize = get(0,'ScreenSize');
    figure(figure_number);
    position=get(figure(figure_number),'Position');
    position=[scnsize(3)/1.5  (j-1)*scnsize(4)/3     scnsize(3)/3
scnsize(4)/4  ];
    set(figure(figure_number),'Position',position)
    x=linspace(0,1,1000);
    n=size(Cl.Clc{j}.in_cluster,1);
    for i=1:4
        memDeg{j}{i} = exp(-((repmat(x,n,1) -
repmat(Cl.Clc{j}.in_cluster(:,i),1,1000)).^2)./(2*repmat(Cl.Clc{j}.sigmas(:,
i),1,1000).^2));
        subplot(2,2,i),plot(x,memDeg{j}{i})
        theStr=sprintf(' Input=   %g',i);
        Ylabel('Mebership Degree ') ;
        Xlabel(theStr) ;
    end
    figure_number=figure_number+1;
end
Cl.memDeg=memDeg;
for i=1:n_class
    n_r=size(Cl.Clc{i}.sigmas,1);
    sx(k:n_r+k-1,:)=Cl.Clc{i}.sigmas;
    cx(k:n_r+k-1,:)=Cl.Clc{i}.in_cluster;
    k=k+n_r;
end
%% search for Feature Redundancy;
k=1;
input_select=zeros(Cl.class,Cl.numInp);

for c=1:Cl.class
    for c2=1:Cl.class
        for i=1:Cl.numInp
            if size(Cl.memDeg{c}{i},1)>1
                a=max(Cl.memDeg{c}{i});
            else
                a=Cl.memDeg{c}{i};
            end
            if size(Cl.memDeg{c2}{i},1)>1
                b=max(Cl.memDeg{c2}{i});
            else
                b=Cl.memDeg{c2}{i};
            end
            ab=[a ;b];
            min_a=min(ab);
            max_a=max(ab);%sum(a)-min_a;
            dist{c}(c2,i)=sum(min_a./max_a);
            %inter{c}(c2,i)=sum(min_a./Cl.memDeg{k}{1,1}(i,:));
        end
    end
    dist_logical{c}=(dist{c}>18);
    a=find(sum(dist_logical{c})==1);
    if isempty(a)
        input_select(c,:)= 1;
    else
        input_select(c,a)= 1;
    end

end
Cl.input_select=input_select;
k=1;
%% Search for Membership Similarity;
for c=1:Cl.class
    for i=1:Cl.numInp
        [numRule sizeofMf]=size(Cl.memDeg{c}{1});
        for j=1:numRule
            for k=1:numRule

                c1=Cl.Clc{c}.in_cluster(j,i);%c inci klasifierin i.inputunun
j inci clusteri
                c2=Cl.Clc{c}.in_cluster(k,i);%c inci klasifierin i.inputunun
j inci clusteri
                s1=Cl.Clc{c}.sigmas(j,i);
                s2=Cl.Clc{c}.sigmas(k,i);

                %  c2>x>c1
                intersect(1)=(c1*s2+c2*s1)/(s1+s2);
                % x>c(2)>c(1)
                intersect(2)=(c1*s2-c2*s1)/(s2-s1);

                memdeg(1)=exp(-((intersect(1)-c1)^2)/(2*s1^2));
                memdeg(2)=exp(-((intersect(2)-c1)^2)/(2*s1^2));
                index=find(UpThr>memdeg &memdeg>LowThr, 1);
                if not(isempty(index));
                    found{c}{i}(j,k)= find(UpThr>memdeg &memdeg>LowThr, 1,
'last' );
                    if found{c}{i}(j,k)==2
                        if s1>s2
                            s2=(s1+s2)/2;
                        else
```

```
                            s1=(s1+s2)/2;
                        end
                    end
                    if c1>c2
                        Cl.Clc{c}.two_c( i,j)=c2;
                        Cl.Clc{c}.two_s(i,j)=s2;
                        Cl.Clc{c}.two_c(i,j+ Cl.numInp)=c1;
                        Cl.Clc{c}.two_s( i,j+ Cl.numInp)=s1;
                        cup=c1;
                        sup=s1;
                        clow=c2;
                        slow=s2;
                    else
                        Cl.Clc{c}.two_c(i, j)=c1;
                        Cl.Clc{c}.two_s(i,j)=s1;
                        Cl.Clc{c}.two_c(i,j+ Cl.numInp)=c2;
                        Cl.Clc{c}.two_s(i, j+ Cl.numInp)=s2;
                        cup=c2;
                        sup=s2;
                        clow=c1;
                        slow=s1;
                    end
                else
                    found{c}{i}(j,k)=0;
                    Cl.Clc{c}.two_c(i,[j j+ Cl.numInp])=c1;
                    Cl.Clc{c}.two_s(i,[j j+ Cl.numInp])=s
                    cup=c1;
                    sup=s1;
                    clow=c1;
                    slow=s1;
                end


                c1Index=(x<=clow);
                c2Index=(x>=cup);
                y1 = exp(-(x-clow).^2/(2*slow^2)).*c1Index + (1-c1Index);
                y2 = exp(-(x-cup).^2/(2*sup^2)).*c2Index + (1-c2Index);
                y = y1.*y2;
                min_a=min(a);
                min_a(min_a==0)=0;
                max_a=max(a);%sum(a)-min_a;
                max_a(max_a==0)=1e-2;
                sim_mf{c}{i}(j,k)=sum(min_a./max_a);
                %  sim_mf{c}{i}(j,k)=sum(min_a./max_a)
            end

        end

    end
end

clear l;
Cl.Clc{1}.two_c=[];
Cl.Clc{2}.two_c=[];
Cl.Clc{3}.two_c=[];

for c=1:Cl.class
    for i=1:Cl.numInp
        [numRule sizeofMf]=size(Cl.memDeg{c}{1});
        for j=1:numRule
            k=find(found{c}{i}(j,:));
            m=k;
            for t=1:size(k,2)
                l{t}=find(found{c}{i}(k(t),:));
                m=union(l{t},m);
            end
            found2{c}{i}{j}=m;
            if isempty(m)
                clow=Cl.Clc{c}.in_cluster(j,i);
                slow=Cl.Clc{c}.sigmas(j,i);
                sup=slow;
                cup=clow;
                m=j;
            else
                cl=Cl.Clc{c}.in_cluster(m,i);
                s=Cl.Clc{c}.sigmas(m,i);
                cls=[cl s];
                cup=max(sortrows (cl));
                clow=min(sortrows (cl));
                sup=s(cl==cup(1));
                slow=s(cl==clow(1));
            end
            Cl.Clc{c}.two_c(m,i)=clow;
            Cl.Clc{c}.two_c(m,i+Cl.numInp)=cup;
            Cl.Clc{c}.two_s(m,i)=slow;
            Cl.Clc{c}.two_s(m,i+Cl.numInp)=sup;
        end
    end
end
figure_number=4;
for j=1:Cl.class
    set(0,'Units','pixels')
    scnsize = get(0,'ScreenSize');
    figure(figure_number);
    position=get(figure(figure_number),'Position');
    position=[scnsize(3)/3  (j-1)*scnsize(4)/3     scnsize(3)/3
scnsize(4)/4  ];
    set(figure(figure_number),'Position',position)
    x=linspace(0,1,1000);
    n=size(Cl.Clc{j}.in_cluster,1);
    for i=1:Cl.numInp

        % memDeg{j}{i} = exp(-((repmat(x,n,1) -
        %
repmat(Cl.Clc{j}.in_cluster(:,i),1,1000)).^2)./(2*repmat(Cl.Clc{j}.sigmas(:,
i),1,1000).^2))
        clow=Cl.Clc{j}.two_c(:,i);
        cup=Cl.Clc{j}.two_c(:,i+Cl.numInp);
        slow=Cl.Clc{j}.two_s(:,i);
        sup=Cl.Clc{j}.two_s(:,i+Cl.numInp);
        for k=1:size(Cl.Clc{j}.in_cluster,1);
            c1Index=(x<=clow(k));
            c2Index=(x>=cup(k));
            y1 = exp(-(x-clow(k)).^2/(2*slow(k)^2)).*c1Index + (1-c1Index);
            y2 = exp(-(x-cup(k)).^2/(2*sup(k)^2)).*c2Index + (1-c2Index);
            y(k,:) = y1.*y2;
        end
        memDeg{j}{i}=y;
        subplot(2,2,i),plot(x,memDeg{j}{i})
        theStr=sprintf(' Input=   %g',i);
        Ylabel('Mebership Degree ') ;
        Xlabel(theStr) ;
    end
    figure_number=figure_number+1;

end
Cl.memDeg=memDeg;
```

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name:	TORUN,Yunis
Nationality:	Turkish (TC)
Date and Place of Birth:	25 Sept. 1978 , Sivas
email:	torun@gantep.edu.tr

EDUCATION

Degree	**Institution**	**Year of Graduation**
MS	Electrical& Electronic Eng. University of Gaziantep	2004
BS	Electrical& Electronic Eng. University of Gaziantep	2001

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2001- Present	GAZÜ- Gaziantep Vocational high school	Lecturer

PUBLICATIONS

1. Torun,Y., Eker,İ., “Experimental Application Of Fuzzy Logic Control To An Electrical Drive System” International Conference On Intelligent Knowledge Systems , Vol.1, No.1, August 2004
2. Eker, İ. ,Torun, Y. ,”Fuzzy logic control to be as a conventional method”, Energy Conversion and Management, Volume 47, Issue 4, March 2006, Pages 377-394
3. Torun, Y., Tohumoğlu, G., Veri Kümeleme Yöntemlerininin Rahim Ağzı Kanserinde Kullanımı, *ASYU, INISTA* 2008.
4. Torun, Y., Tohumoğlu, G., Benzetilmiş Tavlama ve Çıkarımlı Kümeleme Tabanlı Bulanık Sınıflandırcıların Biyomedikal Sınıflandırma Problemlerine Uygulanması, *IEEE SIU 2009*.
5. Torun, Y., Tohumoğlu G., Designing Simulated Annealing Subtractive Clustering based Fuzzy Classifier, *Applied Soft Computing*, Accepted Manuscript, doi:10.1016/j.asoc. 2010.07.020