KNOWLEDGE DISCOVERY USING NEURAL NETWORKS

THESIS

Presented to the Graduate Council of
Texas State University — San Marcos
in Partial Fulfillment of
the Requirements

For the Degree

Master of SCIENCE

By

Sandesh Doddameti, B.Arch

San Marcos, Texas
December 2003

To

My Parents

ACKNOWLEDGMENT

I would like to express special gratitude to my thesis advisor, Dr. Khosrow Kaikhah. I thank him for all his support, patience, encouragement and involvement. This would not have been possible without him.

I wish to thank Dr. Gerald Farr for his support and help during my graduate studies.

I wish to thank Mr. Howard M.Hankcock, Ms. Sue Neskorik and the Texas STEP team for their support, patience and encouragement during my graduate studies and during the course of this thesis.

I express my special thanks to Mr. Sanjay M. Kumar. Thank you for all your inspiration and motivation. You brought out the best in me. I wouldn t have reached this step in life without your influence.

I dedicate this thesis to my parents Dr. Ashok Doddameti and Dr. Sunanda Doddameti, who are in India. Thank you for all your support, patience and belief in me. This would not been possible without you. I thank you for all your sacrifices.

I would like to thank all my grandparents and family in India for your support and belief in me.

Lastly, I would like to give my special thanks to my best friend in India, Dr. Naveen Niranjan, for his friendship and belief in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A vital type of knowledge that can be acquired from vast amounts of data generated in today's world are the hidden trends These hidden trends highlight the generality that exist in the data and can be expressed as rules or correlations These trends, which are specific to the application, represent a type of knowledge discovery The acquired knowledge is extremely helpful in understanding the domain, which the data describes

In this thesis, a process for discovering trends in datasets using neural networks is presented The process consists of five phases – Data preparation, Training, Pruning and re-training, Clustering, and Extraction

In phase one, the data is encoded into binary vectors in the data preparation phase In the training phase, a supervised learning method is used to train the neural network The network learns the correlations that exist in the dataset During training, inconsistent patterns are removed via a filtering process In the pruning and re-training phase, the unnecessary connections and neurons are pruned and the network is re-trained The clustering phase superimposes a layer of adaptive clustering neural network on the hidden layer of the network The purpose of the superimposed layer is to create generalized regions of activation for hidden layer neuron activation values and to identify a representative value for each region The extraction phase of uses the trained network with the superimposed layer, to discover the trends in the dataset

The process provides several control parameters such as frequency, radius, and activation level to achieve flexibility and stringency for the extracted trends Predicted trends are discovered during this phase using all combinations of the input patterns

Finally, the applicability and robustness of the process is demonstrated by applying the process to real world datasets demographic and crime, dietary factors and Plasma Retinol and Beta-Carotene concentrations, system measurements and CPU usage, body measurements and body

fat percentage, pollution and mortality The process was used to predict trends from acquired knowledge in the demographics-crime dataset

Keywords Adaptive Clustering, Data Analysis, Data Mining, Hidden Layer Neuron Activation Values, Knowledge Discovery, Neural Networks, Prediction, Supervised Learning

CHAPTER 1. INTRODUCTION

1.0 Explanation of the Problem

Enormous amounts of data are being generated and recorded for almost any kind of event or transaction that we perform. Advances in data storage and database technology have enabled us to store this vast amount of data. A small piece of data may be quite insignificant. However, taken as a whole, data encompasses a vast amount of knowledge. We can perform data analyses from different perspectives to obtain meaningful results. A vital type of knowledge that we could acquire is the hidden trends in the data. These hidden trends highlight the generality that exists in the data and can be expressed as rules and correlations. These trends, which are specific to the application, represent a type of knowledge discovery. The acquired knowledge is helpful in understanding the domain, which the data describes.

The need to device methods to discover and extract these hidden trends is obvious. Several researchers in the field of Knowledge Discovery and Data Mining have proposed methods for finding the trends in data with varied degree of success.

In this thesis, we define a process to extract rules and correlations in datasets, which we call Patterns. In addition, we define a process to predict extended rules and correlations based on available datasets (patterns), thus providing some form of rule generalization.

1.1 Patterns

Patterns are defined as tuples of data. A Pattern is a collection of attributes. Each attribute has a defined domain, which describes some characteristic of a real world entity. The attribute values can be discrete or continuous. For example, a pattern may describe the characteristics (personal

data) of a credit card holder and a different pattern may describe the frequencies and types of his transactions

The following are three types of patterns

a {set of events} $\rightarrow$ {set of consequences}

data set of events A, set of related consequences B

eg $\{a_1, a_2, a_3, a_4, a_5\} \rightarrow \{b_1, b_2, b_3\}$

b {set of left hand attributes} $\rightarrow$ {set of right hand attributes}

data The pattern split up as LHS and RHS

eg $\{a_1, a_2, a_3\} \rightarrow \{a_4, a_5\}$

c {set of characteristics} $\rightarrow$ {class to which it belongs OR result / consequence}

1.2 Trends and Rules in Patterns

A dataset is a collection of several patterns There are hidden trends in large datasets based on the associativity of patterns These trends describe the commonality that exists in data The trends can be expressed as rules

The rules can be expressed using the following syntax

$$IF\left[(x_1 \le a_1 \le x_2) \wedge (y_1 \le a_2 \le y_2) \wedge \ldots\right] \qquad THEN\left[(z_1 \le a_3 \le z_2) \wedge \ldots\right]$$

Or

$$IF\left[(x_1 \le a_1 \le x_2) \wedge (y_1 \le a_2 \le y_2) \wedge \ldots\right] \qquad THEN\ Outcome$$

For example, in a financial institution environment, where information about customers characteristics and activities are maintained, the following rule may exist

Persons who are between 25-30 yrs old, having at least a bachelor s degree and earning greater than 50K have greater than 6 entertainment activities and greater than 10 restaurant activities in each cycle

1.3 Data Mining and Associative Rules

Association Analysis is a type of data mining, which deals with the discovery of association relationships or correlation among sets of items [Zhang 2002] Associations are often expressed in rule form showing attribute values that occur frequently in a given set of data An association rule of the form

$$X \rightarrow Y$$

is interpreted as IF X occurs THEN Y is likely to occur [Han2001]

Associative rule mining can be formally defined as

Let I = $\{I_1, I_2, I_3, , I_n\}$ be the set of items in a transaction or tuple in a database Or a relation like $R(a_1, a_2, a_3, , a_n)$ in a relational database

Then X is an itemset if it is a subset of I

Let D=$\{t_i, t_{i+1}, t_{i+2}, , t_n\}$ be a set of transactions or tuples in a database where each t has an unique identifier tid

Then Itemset T = (tid,k-itemset)

T is a transaction tid which contains k items and these items are a subset of I

A transaction t contains the itemset X iff all the items in X are in that transaction

Each association rule has 2 quailty measurements, Support and Confidence

Support is defined as

$$Supp(X) = \frac{|X(t)|}{|D|}$$

where $X(t)$ is $p\{t \in D \mid t \ contains \ X\}$

If an ıtemset has a support greater than or equal to a defıned mınımum support then the ıtemset ıs saıd to be a *frequent ıtemset*

Confıdence ıs defıned as

Confıdence (X→Y) = |(X U Y)| / |(X)|

Support Confıdence Framework for an assocıatıve rule can be defıned on 2 ıtemsets as

X, Y havıng a rule X → Y, such that X ∩ Y = fl and

a) supp (X U Y) > mınımum support

b) conf (X→Y) = supp(XUY) / supp(X) ≥ mınımum confıdence

The followıng are three types of data on whıch Associatıon Analysıs can be performed

1 Item based · Transactıonal data (each ınput contaınıng a subset of ıtems)

2. Quantıtatıve Based on relatıonal data tuple where each attrıbute has sıts own domaın.

3. Causalıty · Associatıon between occurrence of words wıth respect to other words

1.4 Neural Networks

Neural Network ıs a hıghly parallel connectıonıstıc model of computatıon Neural Networks consıst of a number of unıts, whıch perform sımple operatıons, connected vıa adaptable lınks The major dıfference between the Von Neumann model of computıng and the neural network model ıs that neural networks are adaptable and they learn by examples Neural networks perform well ın non-lınear problem spaces and problems ınvolvıng hıgh dımensıonal data Neural networks posses a generalızatıon property and tolerant to noıse ın datasets

Neural networks are used ın classıfıcatıon, clusterıng, modelıng functıons (approxımatıon), pattern associatıon, forecastıng and control applıcatıons Some of the applıcatıons where neural networks are used ınclude credıt card fraud detectıon, pattern recognıtıon, fınancıal forecastıng, medıcal dıagnosıs and data vısualızatıon Neural networks are used ın varıous applıcatıons where Von Neumann model of computatıon ıs not feasıble or ınefficıent to use

1.5 Supervised Learning

In supervised learning, the desired outcome of the neural network is available to measure the degree of error in network s performance [Mehrotra 1997] With supervised learning, the network learns to map input patterns to output patterns and generates a mapping model based on the training dataset This is accomplished by measuring the discrepancy between the network s output and the desired output and by using it to adjust the free parameters of the network The occurrence of similar input-output patterns and their frequency strengthen the mapping between such patterns and allows the network to develop high tolerance to noisy patterns Therefore, given a particular dataset the neural network attempts to approximate the overall model of the dataset Hence the rules, which describe the mapping in the dataset are said to be stored in the network

Neural networks are eminent at mapping non-linear problems and problems involving high dimensionality These problems are quite difficult to solve using other methods The downside of using neural networks is their comprehensibility, since they are not able to explain how the mapping process is performed

1.6 Neural Networks and Associative Rule Mining

There has not been much research done to explore the use of association rule mining using neural networks The objective is to define a process for finding associations in data using neural networks To accomplish this task we need to consider the following

- How to capture the Support-Confidence framework, which are frequency and probability counts?
- How to define and measure parameters of Correlation and Interestingness ?
- How to extract the rules from the neural networks and how to represent them?

Neural networks are able to solve highly complex problems due to the non-linear processing capabilities of their neurons In addition, the inherent modularity of the neural network structure makes it adaptable to a wide range of applications The neural network adjusts its parameters to

accurately model the distribution of a provided dataset [Rogers 1997] Therefore, exploring the use of neural networks for finding association and correlation between data should produce interesting results

Several methods have been developed to extract rules from neural networks However, none of the methods so far is superior to others We will describe the existing methods in Chapter 2

1.7 Our Approach

We have developed a method for discovering hidden rules and trends in patterns utilizing neural networks We provide parameters to control the quality and quantity of rules We can also predict rules based on existing patterns The advantage of using neural networks is that they can learn non-linear mappings within the dataset without the need for a complicated or application specific algorithm

The significance of our approach lies in using neural networks for discovering rules from data, with control parameters In our approach, we can control the accuracy of the rule and probability of its occurrence, which are similar to support and confidence framework of associative data mining

We have also developed a method for predicting and extracting rules based on the model generated by the neural network

The following steps describe our approach in more detail

Step 1 Encoding the data into appropriate Binary Patterns

The data may be real values or discrete values A proper encoding scheme has to be designed to discretize data into binary patterns without losing the meaning or the accuracy of the data The real values are discretized into intervals and groups of intervals are defined

Goal To map the data into a binary vector which can be used as input and output to the neural network

Step 2 Neural Network Training

A supervised learning method will be used to train the neural network The neural network learns the existing associations in the dataset During training, the corrupted patterns will be filtered via a filtering process, to remove any inconsistencies in data

Goal To train the neural network in order to capture the inherent relationships in the data

Step 3 Pruning the neural network and Re-Training

Prune unnecessary connections and units from the neural network without losing significant performance and re-train the neural network

Goal To reduce the complexity of the network and to remove unnecessary connection

Step 4 Clustering the Hidden Unit Activation Values

Cluster the Hidden unit activation values of the network using an adaptable clustering technique

Goal To group similar activation values and to define a representative for each group as the centroid of the group

Step 5 Extraction of Rules

Using the centroid of the clusters of hidden unit activation values, extract the existing hidden and predicted rules from the neural network

Goal To find the trends in data

CHAPTER 2. RELATED WORK

2.0 Overview

This chapter briefly explains the related research in using neural networks for data mining Section 2 1 introduces the research in data mining Section 2 2 sheds light on a few applications which demonstrate how neural networks are used for data analysis tasks Section 2 3 describes related work with respect to finer aspects of out process such as pruning and rule extraction Section 2 4 gives a brief explanation about our approach and how it compares to the related work

2.1 Research in Data Mining

The techniques that are used in the data mining process are generally drawn from diverse areas of research [Deogun 1998] The last few years have seen an increasing use of techniques in data mining that draw upon or are based on statistics In many data analysis problems, statistical methods are not suitable either because of strong statistical assumption, such as adherence to a particular probability distribution model, or due to fundamental limitations of the statistical approach The primary limitation is the inability to recognize and generalize relationships, such as the set inclusion, that capture structural aspects of a dataset as a result of being entirely confined to arithmetic manipulations of probability measures [Deogun 1998]

Machine learning has been used for data mining problems such as learning from examples, formation of concepts from instances, discovering regular patterns, noisy and incomplete data, etc [Deogun 1998]

Neural networks are inherent data mining engines Although neural networks learning algorithms have been successfully applied to a wide range of supervised and unsupervised learning problems, they have not often been successfully applied in data mining settings, in which two fundamental considerations are comprehensibility of learned models and the time required to induce models from large datasets [Craven 1998] One such example is Semantic integration of heterogenous databases using Neural Networks, which is an application of using neural networks to semantically match attributes from different databases In this approach, clustering is used to recognize and make groups of attributes It uses the different attributes learned to train the neural network and uses that to classify the unknown data into a particular attribute [Li 1994]

2.2 Neural Network Applications for Data Mining

In the article Using Neural Networks for Data Mining, Shlavik discusses the suitability of neural networks for data mining tasks He states that neural networks provide a more suitable inductive bias for learning the hypothesis than competing algorithms In other cases, neural networks are the preferred learning method not because of the class of hypotheses that they are able to represent, but simply because they induce hypotheses that generalize better than those of competing algorithms

Neural networks are difficult to comprehend because of the sheer number of parameters in a typical network and the non-linear non-monotonic relationships between the input and output which are not possible to determine in isolation Understanding hidden units is often difficult because they learn distributed representations In a distributed representation, the individual hidden units do not correspond to well understood features of the problem domain Instead features, which are meaningful in the context of the problem domain, are often encoded by patterns of activation across many hidden units

Two different directions of using neural networks for data mining have been surveyed The first approach uses methods to extract the hypothesis learned by a fully trained network The second

approach to data mining using neural networks uses learning methods that directly learn comprehensible hypotheses by producing simple neural network [Craven 1998] We develop our methods in accordance to the first approach to extract the rules from a fully trained neural network

Some of the researchers have demonstrated applications and usage of neural networks for data mining problems A few of the approaches are briefly described in the remaining part of the section

In the article Effective Data Mining Using Neural Networks, Setiono applies feedforward multilayer neural networks to data mining classification problem The overall process is as follows

Train a neural network for a classification problem using the dataset The network will be trained to the desired accuracy

The network is then pruned to obtain a minimal architecture to improve generalization and to decrease the complexity

The knowledge learned is then extracted in the form of rules

An example, which classifies persons based on their age and income, has been demonstrated in the article The rules extracted are of the if—then form

The overall process is defined for three layer networks (1 input, 1 hidden and 1 output layer) and for classification problems There are no significant control parameters for data analysis We have used a similar pruning technique, which we will explain in detail in Chapter 4

The extraction phase relies on the complexity of the hidden layer and the number of activation values for each hidden neuron Based on the average number of clusters of the hidden unit activation values, the outputs are calculated for each combination of the cluster center Then the

ınputs, whıch generate those cluster center combınatıons, are found and mapped to the output value Thıs ınput-output combınatıon ıs expressed as a rule

Our overall process provıdes a framework for mappıng m dımensıonal ınput vectors to n dımensıonal output vectors We have developed the extractıon procedure whıch provıdes control and flexıbılıty In addıtıon, ın our approach the complexıty of the hıdden layer does not pose a burden on the extractıon process

In some demonstrated applıcatıons a feedforward neural network ıs used for summarızıng text artıcles [Chuang 2000] A neural network ıs traıned wıth ınputs, whıch are features sentences found ın an artıcle lıke locatıon, length, occurrence of thematıc words, number of tıtle words The respectıve outputs are theır relatıve rankıngs wıth respect to the artıcle Based on categorızatıon learnt by the neural network, sentences from a test artıcle can be ranked The hıghest ranked sentences wıll become the summary of the artıcle Thıs ıs an example of text data mınıng The network may also be pruned and the rules are extracted to determıne what the network uses as a factor to rank the sentences Pedagogıcal, black box methods are used to extract the rules ın some of the approaches

Wang, Ma, Shasha and Wu present an example of bıologıcal data mınıng usıng neural networks ın the artıcle Applıcatıon of Neural Networks to Bıologıcal Data Mınıng A Case Study ın Proteın Sequence Classıfıcatıon In thıs case study, a bayesıan neural network ıs traıned wıth protein sequences to classıfy them as belongıng to a partıcular super famıly The ınput to the network ıs the protein sequence encoded ın a specıal form and the output ıs a sıngle neuron, whıch ıs actıvated ıf the protein sequence ıs ın a partıcular class In thıs work, no rule extractıon ıs performed and the process ıs a sımple classıfıcatıon on the test data The relevance of thıs artıcle to our work lıes ın applıcabılıty of neural network to learn the hypothesıs of the dataset

In the article A Novel Neural Network for Data Mining, Chan, Tan and Haralalka use feedforward neural networks to analyze financial data and develop an efficient market hypothesis The network proposed here utilizes the backpropagation learning algorithm with modifications to include the temporal factor and the concept of Bollinger Band Crossover This network is known as the Bollinger and Crossover Supervised Network (BBCSN) [Kai 2001]
The input to the network consists of parameters of a stock price ticker and the output refers to the outcome of the stock ticker at the next time instant This is predictive data mining based on time series data

In the article Mining Sales Data using a Neural Network Model of Market Response, Gruca, Klemz and Petersen use neural networks to predict the market share for a brand based on the sales data This network is similar to the one discussed in A Novel Neural Network for Data Mining In this work a feedforward neural network with backproagation learning is used

2.3 Neural Networks and Rule Extraction

In A Survey And Critique Of Techniques For Extracting Rules From Trained Artificial Neural Networks , Andrews, Diederich and Tickle discuss the difficulty in comprehending the internal process of how a neural network learns a hypothesis Knowledge acquired during the training phase is encoded as (a) the network architecture (i e the number of hidden units), (b) an activation function associated with each (hidden and output) unit of the neural network, and (c) a set of (real-valued) numerical parameters (called weights) [Andrews 1995] Several methods have been proposed to understand this acquired knowledge under the terminology of Rule Extraction Without the capability to extract rules from the neural networks, the role of neural networks in the field of data mining will be minimal

According to the survey [Andrews 1995], rule extraction methods have been categorized into decompostional and pedagogical techniques This article discusses various techniques including

the pros and cons of each method They conclude that no single rule extraction/rule refinement technique is currently in a dominant position to the exclusion of all others The distinguishing characteristic of the decompositional' approach is that the focus is on extracting rules at the level of individual (hidden and output) units within the trained Artificial Neural Network In pedagogical' approaches to rule extraction, the trained neural network is treated as a black-box, in other words, the view of the underlying trained Artificial Neural Network is opaque A third category in this classification scheme is a composite approach in which elements of both the decompositional and pedagogical rule extraction techniques are incorporated Boolean decompositional approaches which analyze the architecture and connection strengths and some pedagogical approaches are summarized in this survey [Andrews 1995]

The algorithms for boolean rule extraction can be divided into two categories, described in this article

Boolean Decompostional Approaches a Subset Algorithm

b M-0f-N Technique

c RULEX

Boolean Pedagogical Approaches a VIA Algorithm

b RULENEG Algorithm

In the article Generalized Analytic Rule Extraction for Feedforward Neural Networks, Gupta, Park, and Lam propose an algorithm — GLARE to extract classification rules from feedforward and fully connected neural networks trained by backpropagation The major characteristics of the GLARE algorithm are (a) its analytic approach for rule extraction, (b) its applicability to standard network structure and training method, and (c) its rule extraction mechanism as direct mapping between input and output neurons This method is designed for a neural network with only one hidden layer This approach uses the significance of connection strengths based on their absolute magnitude and uses only a few important connections (highest absolute values) to analyze the rules

2.4 Comparison Between Our Approach and the Related Work

Our approach provides an overall process for finding correlations and rules within a dataset with m dimensional input space and n dimensional output space Our process is not confined to classification Like most neural network applications, our process is independent of the application However our process is not applicable to data analysis which is dependent on sequential nature of some data for example, temporal sequences, DNA sequences or finite state machine sequences

We define a framework for associative data mining, by providing control parameters for data analysis These parameters give control over the probabilities of occurrences and accuracy which are similar to Support and Confidence framework of the associative data mining

Our rule extraction procedure is both decompositional and pedagogical It is decompositional in nature, since we examine the weights for pruning and clustering the hidden unit activation values It is pedagogical, since we use the neural network as a black-box to extract the rules

Our approach is neither limited by the complexity of the hidden layer nor by the number of hidden layers Therefore our approach can be extended to networks with several hidden layers

Another aspect of our approach is the predictive capability for generalized rules This predictive capability is dependent on the accuracy of training and the generalization achieved by the network

CHAPTER 3. NEURAL NETWORK TRAINING

3.0 Overview

This chapter explains in detail the first step of the process — training the neural network To learn the hypothesis of the dataset, we need to first train the neural network Section 3 1 describes pre-processing and encoding of the data Section 3 2 explains the architecture of the neural network Section 3 3 discusses the learning algorithm Section 3 4 discusses other details of the training process Section 3 5 discusses filtering the data patterns

3.1 Pre-Processing of Data

Datasets are collections of input-output patterns The characteristics of patterns are described in chapter 1

Each pattern in the dataset consists of attributes Each attribute has an associated semantics and a value which describes its strength Certain attributes of the data can be removed from the pattern, if it is determined that they are not relevant to the analysis For example, in a dataset consisting of credit card transactions, a unique identifier for each transaction does not provide any relevant information about the nature of the transactions Therefore, the transaction ID attribute can be removed, without any loss of vital information

Our goal is to find the correlations that exist between m input attributes and n output attributes in the entire dataset For example, in a credit card transaction application, we are interested in the correlations that exist between the type of customers and the characteristics of their transactions

Encoding of input and output attributes to make it suitable for training a neural network is a very important part of the process In this process we train the network only on binary inputs and expect binary outputs Thus we need to choose an appropriate encoding scheme for each domain we wish to use

In our approach, the neural network is trained with binary input/output patterns Therefore, the raw data must be discretized and encoded into binary patterns This can be done by grouping the attribute values into intervals and assigning a binary value to each interval

For example, in the credit card transaction application, an attribute may represent a person s age which may be a value greater than 21 Based on our needs and reasoning, we can discretize these into 4 different intervals [21-30],[31-45],[45-65] and 65+ Therefore [0 1 0 0] would represent a customer between the ages of 31 and 45 For some attributes, several intervals may be chosen to represent the attribute The size of the interval depends on the attribute it is representing It can be as small as one unit to several units

3.2 Multilayer Feedforward Neural Networks

Single layer networks are capable of solving only linearly separable problems — problems where the solution space can be divided by a single hyperplane To solve non-linear problems, we need to use multilayer network architecture Multilayer networks have a single input and output layer and several hidden layers in between them Multilayer networks are capable of generating a model where the solution space is divided by more than one hyperplane

Feedforward networks are acyclic networks where the connection between neurons in layer i is allowed only to neurons in layer i+1 All connections have an associated weight value

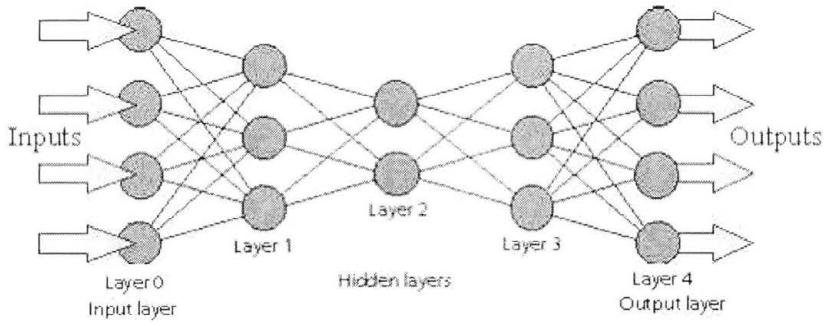

Figure 3.1 Multilayer Feedforward Neural Architecture

Feedforward neural networks have been used for a wide range of applications, such as classification, pattern recognition, control and financial forecasting. The flow of information in a feedforward network is from the input to the output layer. This type of architecture is required for the supervised backpropagation algorithm.

The input and output layers correspond to the dimensionality of the problem space. The number of neurons in the input layer represents the dimension of the input patterns and the number of neurons in the output layer represents the dimension of the output patterns.

Theoretically any number of hidden layers can be used in a Multilayer feedforward network. For most applications, however, one or two hidden layers have been used.

In our approach, we use a feedforward architecture with one hidden layer. The number on neurons in the input layer is the total number of intervals for all input attributes and the number of neurons in the output layer is the total number of intervals for all output attributes.

3.3 Supervised Backpropagation Learning Algorithm

In supervised learning, the desired output pattern for a given input pattern is known Therefore, the neural network is trained to learn the associations among input/output patterns Backpropagation is a feedback-based weight adaptation approach, which is widely used with multilayer networks for a wide range of supervised learning applications [Mehrotra 1997]

Learning in neural networks corresponds to changing its connection strengths (weights) until the desired performance has been achieved There are several methods for changing the connection strengths One such method which is used in backpropagation is the method of gradient descent This method uses the mean square error (MSE) of the network at the output layer and performs an intelligent search on the MSE surface to find its global minima

3.3.1 Definitions

The dataset consists of input-output pairs,

$$\{(a^p, d^p) : p = 1,..., P\}$$

where a^p is the p^{th} input pattern (vector)

d^p is the p^{th} output pattern (vector)

The actual outputs of the network are,

$$\{o^p : p = 1,..., P\}$$

The error of the network for the p^{th} pattern is

where $Err(o^p, d^p) = \left| o^p - d^p \right|^2$

The goal of the training algorithm is to minimize this error

Mean Squared Error of the entire dataset is

$$MSE = \frac{1}{P} \sum_{1=p}^{P} \left| o^p - d^p \right|^2$$

3.3.2 Algorithm

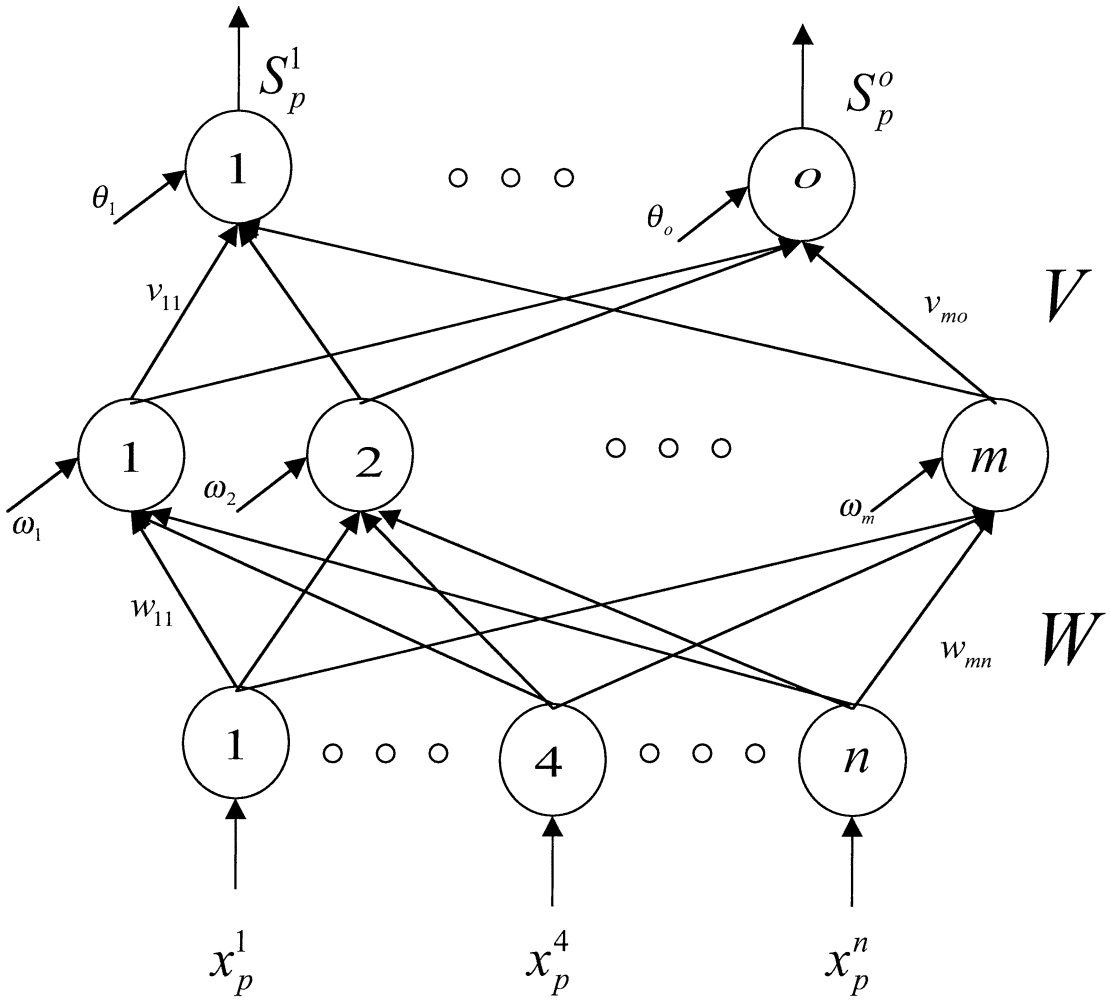

Figure 3 2 Neural Network

1 Initialize all the weights to random values between 0 and 1

2 While (MSE > Error Threshold OR Number of Cycles is below the desired value)

3 For each Pattern p, $1 \le p \le P$,

4 Pass the input through the input layer

$$x_p^1, x_p^2, x_p^3, ..., x_p^n$$

5 Compute the total ınput to each hıdden neuron Net ınput to hıdden neuron J ıs

$$Hnet_J^p = \sum_{ı=0}^{n} w_{ıJ} \cdot x_ı^p - \omega_J$$

6 Compute Hıdden neuron outputs Output of hıdden neuron J ıs

$$y_J^p = S(Hnet_J^p)$$

where $S(net) = \dfrac{1}{1+e^{-net}}$

7 Compute the total ınput to each output neuron Net ınput to output neuron k ıs

$$Onet_k^p = \sum_{J=0}^{m} v_{Jk} \cdot y_J^p - \theta_k$$

8 Compute Network Outputs Output of the network at neuron k ıs

$$o_k^p = S(Onet_k^p)$$

9 Compute error between the network output and desıred output

$$\left| o_k^p - d_k^p \right|$$

10 Correct the hıdden to output connectıons by

$$\Delta v_{Jk} = \eta (d_k^p - o_k^p) \cdot o_k^p \cdot (1 - o_k^p) \cdot y_J^p$$

where η ıs the learnıng factor, typıcally set between 0 1 and 0 5

11 Correct the ınput to hıdden connectıons by

Let $\partial_k^p = (d_k^p - o_k^p) \cdot o_k^p \cdot (1 - o_k^p)$

$$\Delta w_{ıJ} = \eta \sum_{k=0}^{l} \left(\partial_k^p \cdot w_{Jk} \right) \cdot x_J^p \cdot (1 - x_J^p) \cdot x_ı^p$$

12 End For — Step 3

13 End Whıle Step 2

3.4 Network Dynamics and Parameters

The connection strengths are initialized to random values between —1 0 and +1 0 The weights are updated for each pattern in every epoch of training

The choice of the learning rate η is based on experience and empirical judgement A large value of η causes rapid learning but weights may oscillate and never converge A low value of η leads to a stable convergence but results in slow learning In our experiments, the networks have performed better when η is between 0 1 and 0 5

A sigmoid function is used as the activation function for each neuron The sigmoid function is a non-linear function, hence introduces non-linearity in the network The supervised backpropagation learning algorithm uses a gradient descent method, for which a continuous function provides the most accurate implementation Since the sigmoid function is differentiable everywhere, it is a good choice for the backpropagation algorithm

We use the following sigmoid function

$$S(net) = \frac{1}{1+e^{-net}}$$

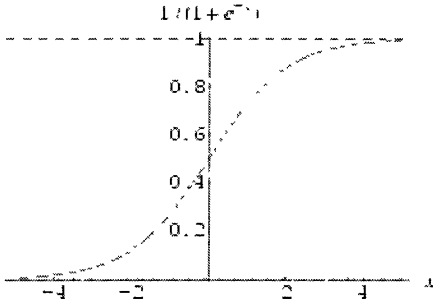

Figure 3 3 Sigmoid Function

The network accuracy is measured as the percentage of training samples which are classified correctly

$$accuracy = \frac{numCorrOuput}{TotNumSamples}$$

The output of a pattern is considered to be correct if the output of each neuron of the network is the same as the desired output for that neuron This is a tight check on the correctness of the output

3.5 Filtering the Data and Re-Training

During the training process, we may not able to achieve an acceptable average mean square error over the entire dataset This may be due to scattered or inconsistent patterns which have high errors These patterns can be filtered out, thus lowering the average mean squared error To achieve this, we define an upper bound for the mean squared error and remove all the patterns whose error is above this bound In our process, we set the upper bound to twice the mean squared error for the current cycle

3.5.1 Procedure

1. Choose the cycle in the training phase in which the data needs to be filtered.
2. When that cycle is reached during training, compute the Mean Square Error of the network for the entire dataset

$$MSE = \frac{1}{P}\sum_{1=p}^{P}\left|o^p - d^p\right|^2$$

3. For each Pattern p, $1 \le p \le P$,
4. Compute the error for the pattern

$$Err(p) = \left|o^p - d^p\right|^2$$

5. If ($Err(p) > 2 \cdot MSE$)

Remove the Pattern p from the dataset.

End-For

6. Continue training the network with the remaining Samples

In our experiments, we have observed that the error stablilizes after 5000-7000 cycles of training. So we train the network for 7000 cycles, and apply the filtering process which removes those patterns whose errors are above twice the MSE. The cycle at which the filtering is done can be changed based on the size of the dataset and the number of cycles required for achieving stability.

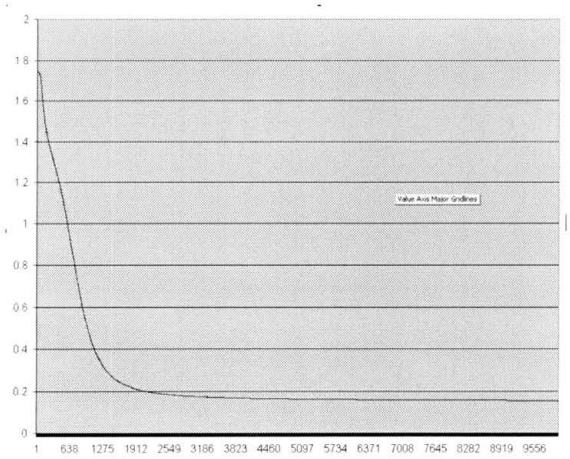

Figure 3.4 Plot of Training for a Sample Set s Without Filtering

Figure 3.4 Plot of Training for a Sample Set s Without Filtering

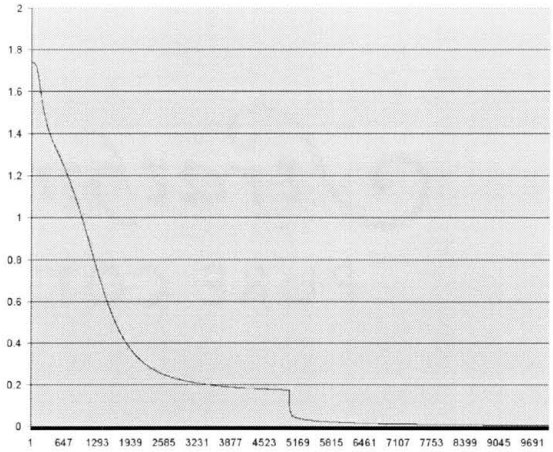

Figure 3.5 Plot of Training for a Sample Set s With Filtering

After filtering, the network will be re-trained with the remaining patterns in the dataset. The filtering process can be applied more than once to achieve desired error. The filtering process helps in removing the scattered or inconsistent patterns. By doing so, the network is trained only on the consistent patterns.

The number of patterns filtered out is critical to the completeness of the extracted rules. If a large percentage of patterns from the dataset are filtered out, we can conclude that there is high probability of scattered or inconsistent patterns. Hence, we cannot infer any conclusive rules about the dataset. The network is not able to generalize the scattered patterns.

In our process, we use the percentage of the patterns filtered out from the original dataset as an upper level Support parameter. The confidentiality of extracted rules are based on the desired level of the upper Support Parameter.

CHAPTER 4. PRUNING THE TRAINED NEURAL NETWORK

4.0 Overview

This chapter deals with the process of pruning the trained network. Pruning is the process of removing unnecessary connections and neurons to obtain a minimal architecture. This improves generalization and also reduces the complexity of the network. Section 4.1 introduces pruning the neural network. Section 4.2 describes the penalty term for training the network. Section 4.3 discusses the pruning criteria. Section 4.4 defines the pruning procedure. Section 4.5 discusses the significance. Section 4.6 describes some test results.

4.1 Introduction to Pruning Neural Networks

One of the problems facing neural network applications is finding the optimal architecture of the network, i.e. the optimal number of hidden neurons and connections. Too many hidden neurons may result in poor generalization of the network and too few hidden neurons may produce an unstable network [Setiono 1996]. Various methods of pruning have been introduced. We use a simple pruning method similar to the method developed by Setiono [Setiono 1996]. In fact, any pruning method can be used for the pruning process as per the application requirements as long as the desired accuracy is achieved.

The connections which are insignificant or provide minimal relevance (based on their connection strengths) to the output of the network are removed. Finally the neurons which do not have any outgoing or incoming connections can be removed.

The pruning process will eliminate some input layer, hidden layer and output layer neurons, thus providing a better generalization. This helps to eliminate insignificant attributes and extract more

concise and more accurate rules Pruning the networks results in a less complex network and improves the generalization A less complex network helps to lower the complexity of the rule extraction process

4.2 Training Using the Penalty Term

In order to identify the insignificant connections and therefore to prune the network, a penalty function must be added to the error function during the training phase of the network The penalty function would force the unnecessary connections to have very small absolute weights which will result in minimal impact and, therefore, can be removed without affecting the output of the network

The penalty function consists of two terms The first term drives the decaying of small weights to values close to zero The second term prevents weights from getting too large

The penalty function is defined as

$$P(w,v) = \rho_{decay}\left(P_1(w,v) + P_2(w,v)\right)$$

where ρ_{decay} is the scaling factor

$$P_1(w,v) = \varepsilon_1\left(\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{\beta w_{ij}^2}{1+\beta w_{ij}^2} + \sum_{j=1}^{n}\sum_{k=1}^{o}\frac{\beta v_{jk}^2}{1+\beta v_{jk}^2}\right)$$

$$P_2(w,v) = \varepsilon_2\left(\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}^2 + \sum_{j=1}^{n}\sum_{k=1}^{o} v_{jk}^2\right)$$

where n is the number of input neurons

m is the number of hidden neurons

o is the number of output neurons

w_{ij} is the i^{th} input to j^{th} hidden layer connection strength

v_{jk} is the j^{th} hidden to k^{th} output layer connection strength

ε_1 is scaling factor typically set at 0 1

ε_2 is the scaling factor typically set at 0 00001

β is the scaling factor typically set at 10

The total energy function to be minimized during the training process is

$$\theta(w,v) = E(w,v) + P(w,v)$$

where $$E(w,v) = \frac{1}{L}\sum_{l=1}^{L}\sum_{k=1}^{o}(S_{lk} - d_{lk})^2$$

L is the number of patterns

o is the number of output neurons

S_{lk} is the output of the k^{th} output neuron for pattern l

d_{lk} is the expected output of the k^{th} output neuron for pattern l

The connection strengths are updated using the gradient descent approach The gradient descent approach is an intelligent search for the global minima of the energy function The gradient of the error function is defined as

$$\nabla\theta(w,v) = \nabla E(w,v) + \nabla P(w,v)$$

The connection strengths are updated proportional to the negative direction of the gradient, since the gradient provides the steepest upward slope

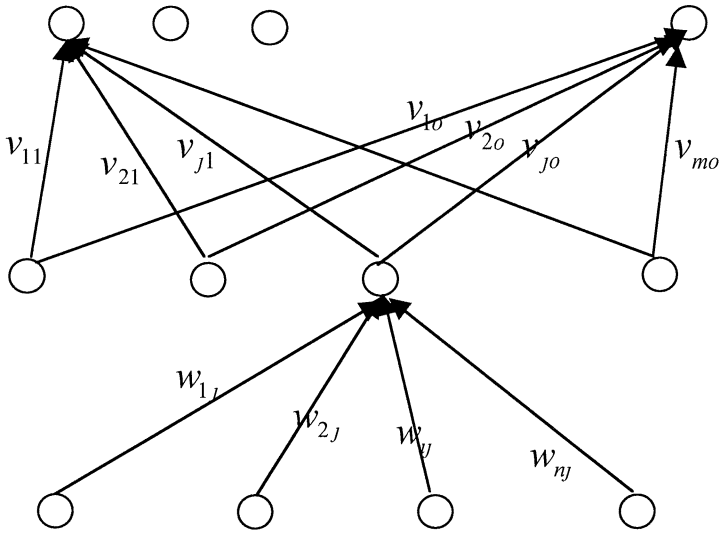

Figure 4 1 Partial Neural Network Showing Weighted Connections

$$\Delta v_{jk} = -\left(\nabla E(v,w) + \nabla P(v,w)\right)$$

$$\nabla E(w,v) = \frac{\partial E}{\partial V} = -2 \cdot e_k^p \cdot S_k^p$$

$$\nabla P(w,v) = \frac{\partial P}{\partial (W,V)} = \frac{\partial P_1}{\partial (W,V)} + \frac{\partial P_2}{\partial (W,V)}$$

Derıvatıve of penalty term P_1

$$\frac{\partial P_1}{\partial v_{jk}} = \frac{\partial\left(\varepsilon_1 \frac{2\beta w_{ij}}{1+\beta w_{ij}^2}\right)}{\partial v_{jk}} + \mathrm{K} + \frac{\partial\left(\varepsilon_1 \frac{2\beta w_{nm}}{1+\beta w_{nm}^2}\right)}{\partial v_{jk}} + \mathrm{L} + \frac{\partial\left(\varepsilon_1 \frac{2\beta v_{11}}{1+\beta v_{11}^2}\right)}{\partial v_{jk}} + \mathrm{K} + \frac{\partial\left(\varepsilon_1 \frac{2\beta v_{21}}{1+\beta v_{21}^2}\right)}{\partial v_{jk}} + \mathrm{K} + \frac{\partial\left(\varepsilon_1 \frac{2\beta v_{jk}}{1+\beta v_{jk}^2}\right)}{\partial v_{jk}} + \mathrm{K} + \frac{\partial\left(\varepsilon_1 \frac{2\beta v_{mo}}{1+\beta v_{mo}^2}\right)}{\partial v_{jk}}$$

$$\frac{\partial P_1}{\partial v_{jk}} = 0 + 0 + \mathrm{K} + \frac{\partial\left(\varepsilon_1 \frac{2\beta v_{jk}}{1+\beta v_{jk}^2}\right)}{\partial v_{jk}} + \mathrm{K}\ 0$$

$$\frac{\partial P_1}{\partial v_{jk}} = \varepsilon_1\ \ 2\ \ \beta\ \left(\frac{v_{jk}}{(1+\beta v_{jk}^2)^2}\right)$$

Sımılarly, $\frac{\partial P_1}{\partial w_{ij}} = \varepsilon_1\ \ 2\ \ \beta\ \left(\frac{w_{ij}}{(1+\beta w_{ij}^2)^2}\right)$

Derıvatıve of penalty term P_2

$$\frac{\partial P_2}{\partial v_{jk}} = \frac{\partial w_{11}^2}{\partial v_{jk}} + \mathrm{K} + \frac{\partial w_{ij}^2}{\partial v_{jk}} + \mathrm{K} + \frac{\partial w_{nm}^2}{\partial v_{jk}} + \frac{\partial v_{11}^2}{\partial v_{jk}} + \frac{\partial v_{12}^2}{\partial v_{jk}} + \mathrm{K} + \frac{\partial v_{21}^2}{\partial v_{jk}} + \mathrm{K} + \frac{\partial v_{jk}^2}{\partial v_{jk}} + \mathrm{K} + \frac{\partial v_{mo}^2}{\partial v_{jk}}$$

$$\frac{\partial P_2}{\partial v_{jk}} = 0 + \mathrm{K} + 0 + \mathrm{K} + 0 + 0 + 0 + \mathrm{K} + 0 + \mathrm{K} + \frac{\partial v_{jk}^2}{\partial v_{jk}} + \mathrm{K} + 0$$

$$\frac{\partial P_2}{\partial v_{jk}} = \varepsilon_2 \cdot 2 \cdot v_{jk}$$

Sımılarly, $\frac{\partial P_2}{\partial w_{ij}} = \varepsilon_2 \cdot 2 \cdot w_{ij}$

Therefore, the hidden to output neuron connections are updated by

$$\Delta v_{jk} = \eta(d_k^p - S_k^p)\cdot S_k^p \cdot (1-S_k^p)\cdot x_j^p - \left(\varepsilon_1 \ \ 2\cdot\beta\cdot\left(\frac{v_{jk}}{(1+\beta v_{jk}^2)^2}\right)\right) - \varepsilon_2 \cdot 2 \cdot v_{jk}$$

and the input to hidden neuron connections are updated by

$$\Delta w_{ij} = \eta \sum_{k=1}^{o} \left(\partial_{p,k}\cdot v_{jk}\right) x_j^p \cdot (1-x_j^p)\cdot x_i^p - \left(\varepsilon_1 \cdot 2\cdot\beta\cdot\left(\frac{w_{ij}}{(1+\beta w_{ij}^2)^2}\right)\right) - \varepsilon_2 \cdot 2 \cdot w_{ij}$$

where $\partial_{p,k} = (d_k^p - S_k^p)\cdot S_k^p \cdot (1-S_k^p)$

4.3 Soundness of the Pruning Criteria

The pruning criteria is defined as follows

For each w_{ij} in the network,

if $\max_k \left| v_{jk} w_{ij} \right| < 4\eta_2$ **, remove** w_{ij}

For each v_{jk},

if $\left| v_{jk} \right| \le 4\eta_2$ **, remove** v_{jk}

where w_{ij} is the i^{th} input to j^{th} hidden layer connection strength

v_{jk} is the j^{th} hidden to k^{th} output layer connection strength

The proof of correctness of the pruning criteria is shown in [Setiono 1996] We use the same pruning criteria used by Setiono

The output of a network for an input pattern x_i at output neuron k is

$$S_k^i = \sigma\left(\sum_{J=1}^{m} \sigma\left(x_i \cdot w_{ij}\right) v_{jk}\right)$$

where S_k^i is the output of neuron k for input i

m is the number of hidden neurons

x_i is the input to neuron i

σ is the sigmoid activation function

w_{ij} is the i^{th} input to j^{th} hidden layer connections

v_{jk} is the j^{th} hidden to k^{th} output layer connections

It is shown in [Setiono 1996] that for a connection of zero strength and considering S_k^i as a function of a single variable weight,

$$\left|S_k^i(0) - S_k^i(w_{ij})\right| \leq \left|v_{jk} \cdot w_{ij}\right| / 4$$

$$\left|S_k^i(0) - S_k^i(v_{jk})\right| \leq \left|v_{jk}\right| / 4$$

A pattern is correctly classified if the following condition is satisfied

$$\left|e_k^i\right| = \left|S_k^i - d_k^i\right| \leq \eta_1 \text{, where } \eta_1 \in [0, 0.1)$$

where e_k^i is the error of the output neuron k for input i

S_k^i is the actual output of the neuron k for input i

d_k^i is the desired output of the neuron k for input i

For $\eta_1 + \eta_2 < 0.5$, suppose a trained network classifies an input x_t correctly The effect of the output on removing the connections is shown by

$$\left|S_k^t(0) - d_k^t\right| \le \left|S_k^t(0) - S_k^t(w_{ij})\right| + \left|S_k^t(w_{ij}) - d_k^t\right|$$

$$\left|S_k^t(0) - d_k^t\right| \le \eta_2 + \eta_1$$

$$\left|S_k^t(0) - d_k^t\right| \le 0.5$$

It is shown that if $\max_p \left|v_{jk} \cdot w_{ij}\right| \le 4\eta_2$, w_{ij} can be removed and the overall accuracy will not deteriorate significantly Similarly if $\max_p \left|v_{jk}\right| \le 4\eta_2$, v_{jk} can be removed

4.4 Pruning Procedure

The pruning process consists of four phases In Phase 1, the pruning criteria is applied to all connections of a trained network This is called the sweep phase The network is retrained after this phase In Phase 2, the pruning criteria is applied to the connections leading into and out of each hidden layer neuron and the network is re-trained, one neuron at a time Phase 3, is a repeated sweep as in Phase 1 In Phase 4, all unnecessary neurons, those having no incoming or outgoing connections, are removed

The pruning procedure is defined as follows

0a Train the neural network till desired accuracy is achieved, or the required number of cycles has been reached

0b Choose a value η_2 such that

$$\eta_1 + \eta_2 < 0.5$$

Prune Phase 1 . (Sweep Phase)

1a Apply the pruning criteria to all connection strengths

1b Retrain the network

1c If the accuracy decreases, restore the previous connections

Prune Phase 2 .

2a For each hidden neuron j ,

2b Apply the pruning criteria to the connections coming into and out of neuron j

2c Retrain the network

2d If the accuracy decreases, restore the connections of neuron j and continue

End For

Prune Phase 3 . (Sweep Phase)

3a Apply the pruning criteria to all connection strengths

3b Retrain the network

3c If the accuracy decreases, restore the previous connections

Prune Phase 4 . (Neuron Removal Phase)

4a For each input layer neuron i in the network,

If there are no outgoing connections, remove neuron i

4b For each Hidden Layer Neuron j in the network,

If there are no incoming connections, remove neuron j

4c For each remaining Hidden Layer Neuron j in the network,

If there are no outgoing connections, remove neuron j

4d For each Output Layer Neuron k in the network,

If there are no incoming connections, remove neuron k

4.5 Significance of Pruning to the Extraction Process

Pruning is an important step for the extraction of rules Not only does the pruning decrease the complexity of the network in terms of connections, but also it removes unnecessary neurons Neurons which do not have any incoming or outgoing connections can be removed from the network The removal of neurons can offer significant network generalization

Removal of input neurons suggest that those inputs are irrelevant to the data model Similarly output neurons which are removed suggest that those outputs are irrelevant These input and output neurons will not be included in the rules, hence the generalization is enhanced The removal of hidden neurons reduce the complexity for clustering process

4.6 Pruning Test Results

An Illustration of pruning the neural network is shown below.

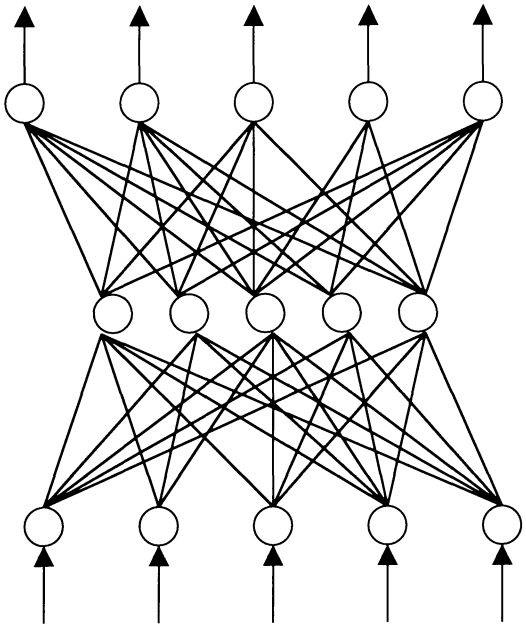

Figure 4.2 Before Pruning

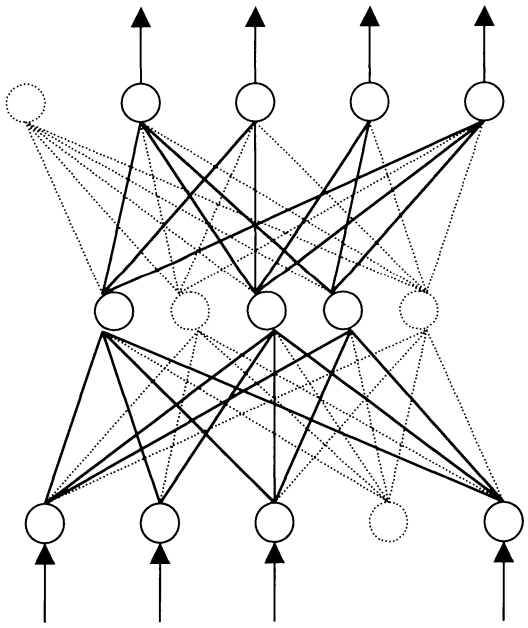

Figure 4.3 After Pruning

The dashed connections and neurons represent the pruned connections and neurons.

The results obtained for different values chosen for ρ_{decay} are shown in below

Learning Rate α	0 5		0 3		0.4	
Hidden-Output ρ_{decay}	0 01		0 01		0.01	
Input-Hidden ρ_{decay}	0 05		0 03		0 03	
Pruning Criteria (4 x)	0 49		0 49		0 49	
Test Dataset 1	96% 39/250 15 6%	96% 45/175 25 7%	96% 49/250 19 6%	96% 66/175 37 7%	96% 32/250 12.8%	96% 52/175 29 7%
Test Dataset 2	98% 9/250 3 6%	98% 85/175 48%	74% 37/250 14 8%	74% 118/175 67 4%	90% 10/250 4%	90% 104/175 59.4%
Test Dataset 3	60% 26/250 10.4%	66% 112/175 64%	46% 25/250 10%	46% 124/175 70%	96% 5/250 2%	96% 104/175 59 4%
Test Dataset 4	90% 73/250 29%	90% 98/175 56%			88% 63/250 25%	88% 112/175 64%

Index	Accuracy before	Accuracy after
	Number of connections Pruned	
	Input-Hidden	Hidden-output

Table 4 1 Summary of Pruning Tests

CHAPTER 5. ADAPTIVE CLUSTERING OF HIDDEN NEURON ACTIVATIONS

5.0 Overview

This chapter describes the clustering of hidden layer activation values of the neural network. Section 5.1 discusses the significance and benefits of the clustering phase. Section 5.2 defines and illustrates the clustering algorithm. Section 5.3 defines and illustrates the re-clustering algorithm. Section 5.4 discusses the overall clustering technique and illustrative examples.

Clustering is the third phase of the rule extraction process. In the clustering phase :

1. The hidden layer activation values for each neuron are dynamically clustered having a cluster radius of r_c
2. The hidden layer activation values for each neuron are re-clustered having a confidence radius of $rConf_c$

Theoretically, clustering superimposes a new layer of neurons in place of each hidden layer neuron. This can be visualized as shown below :

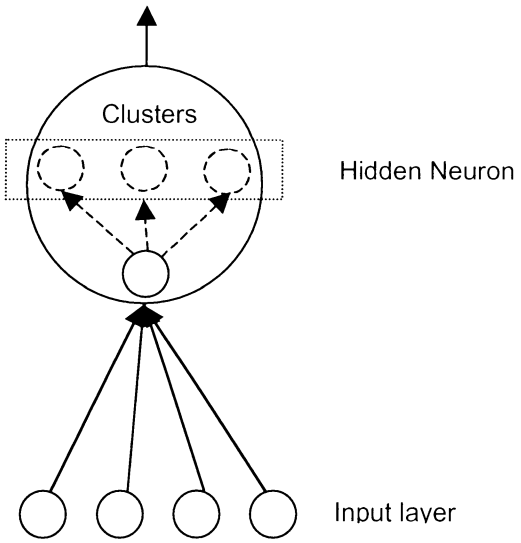

Figure 5.1 Hidden Layer Neuron Superimposed with Activation Clusters

5.1 Significance and Benefits of Clustering and Re-Clustering

The clustering of hidden layer neuron activation values provide representative values for hidden neuron activations. The centroid of each cluster represents the mean of the values in the cluster and can be used as the representative value of the cluster. This is a form of generalization. By using the centroids of the clusters, each hidden neuron has a minimal set of activations. This helps with getting generalized outputs at the output layer.

Our clustering process has control parameters for cluster radius and frequency. More importantly, it provides dynamic control parameters for confidence radius and confidence frequency. The confidence radius which is a subset of cluster radius controls the tolerance and accuracy of the extraction process. As the confidence radius decreases, the confidence of the extracted rule increases. The confidence frequency determines the frequency of the activity within the confidence radius. As confidence frequency increases so does the consistency and support of the extracted rule.

We use a dynamic clustering technique. This allows flexibility in terms of number of clusters based on cluster radius.

The significance of clustering process is to determine the closeness and frequency of hidden neuron activation values. Clusters with high frequency represent high activities and confidence, whereas, clusters with low frequency represent inconsistent and infrequent activities.

5.2 Adaptive Clustering of Hidden Layer Neuron Activation Values

The output of a hidden layer neuron for any input pattern is called the hidden layer neuron activation The activations of a hidden layer neuron for a set of input patterns are scattered over the activation space (one dimensional) In this phase of the process, regions of similar activations are grouped and a representative value for each region is determined by means of clustering algorithm The purpose of this is to find generalizations for regions of consistent activities and also to filter inconsistent activities

Element Set The element set E_j is the set of activations of a hidden layer neuron j for all the patterns in the dataset D

$E_j = \{activations_j^p\}, 1 \le p \le P$, where P is the set of patterns

Cluster Cluster c is a region having a radius of r_c which includes elements e_c^i, where $1 \le i \le n_c$ and n_c is the number of elements in the cluster Clusters may be overlapping or disjoint

Cluster Frequency The number of elements in a cluster c is called the frequency of a cluster denoted by $freq_c$

Centroid The center of a cluster c is denoted by G_c The centroid is adjusted dynamically as a new element e_c^t is added to the cluster

$$G_c^{new} = \frac{(G_c^{old} \cdot freq_c) + e_c^t}{freq_c + 1}$$

The centroid is the representative value of the cluster

$Dist(G_c, e)$ is the numerical distance of the element e from the centroid G_c

$$Dist(G_c, e) = |G_c - e|$$

Cluster Radius The radius of a cluster defines the distance of the farthest element to the centroid

$$|G_c - e_c^i| \leq r_c \quad \text{for any cluster } c$$

Cluster space Clusterspace U_{E_J} is defined as the collection of clusters c_m for a particular Elementset E_J, where $m \geq 0$ is the number of clusters in the clusterspace

$c_{\min}(e)$ is the cluster whose $Dist(G_c, e)$ is the least among all existing clusters in the clusterspace U_{E_J}

$$c_{\min}(e) = \min(Dist(G_c, e)) \quad \forall c \in U_{E_J}$$

Confidence Radius the distance of the farthest confidence element to the centroid The confidence radius is usually less than the cluster radius It is denoted as $rConf_c$

Confidence Frequency The number of elements enclosed within each confidence radius It is denoted by $freqConf_c$

The clustering algorithm is adaptable, that is the clusters are created dynamically as elements are added into the clusterspace Therefore, the number of clusters and the number of elements in each cluster are not known apriori

Initial Condition 1 Elementset E exists

2 Clusterspace U_E is empty

Prodecure

1 For each element e in E

1a Find $c_{min}(e)$ in the clusterspace U

Case

1b (i) If $c_{min}(e)$ is null, i e no cluster in the clusterspace

2a Create a new cluster $c_{new}(e)$

2b Set $freq_{c_{new}(e)} = 1$

1b (ii) If $Dist(G_{c_{min}}, e) > r_c$, then e lies outside the cluster radius of $c_{min}(e)$

2a Create a new cluster $c_{new}(e)$

2b Set $freq_{c_{new}(e)} = 1$

1a (iii) If $Dist_e(G_{c_{min}}, e) \leq r_c$, then e lies within the cluster radius of $c_{min}(e)$

a Add e to $c_{min}(e)$ and adjust the centroid

b Increment $freq_{c_{min}(e)}$ by 1

End For

5.3 Re-Clustering Based on Confidence Radius

The re-clustering is performed based on a confidence radius It is performed after an initial clusterspace is built on an elementset using the clustering process described in the previous section Unlike clustering, re-clustering is non-dynamic The cluster centoids remain fixed during the re-clustering and no new clusters are created The clusterspace is adjusted using the confidence radius around the fixed centroids The re-clustering can be performed any number of times using different confidence radii The confidence radius should always be less than the cluster radius

Re-clustering helps re-organize the clusterspace using the confidence radius around the fixed centroids The purpose of this is to define a confidence area around the fixed centroids Only activations within these areas are considered for the confidence frequency This helps to eliminate inconsistent activations The activation values are not within any clusters are inconsistent activations since (1) the bound for cluster radius enforces the correctness of the cluster centroids, and (2) the confidence radius is always set to be less than or equal to the cluster radius

The re-clustering is also necessary to solve the problem of overlapping clusters Since the adaptive is order dependent, some elements may be clustered in a cluster whose centroid is not the nearest to the element By re-clustering the clusterspace, the clusters are re-organized and some elements end up in confidence regions An illustration of re-clustering is shown below

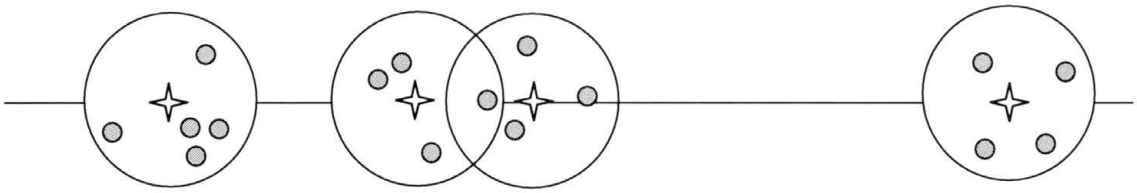

Figure 5.2 The clusterspace after clustering

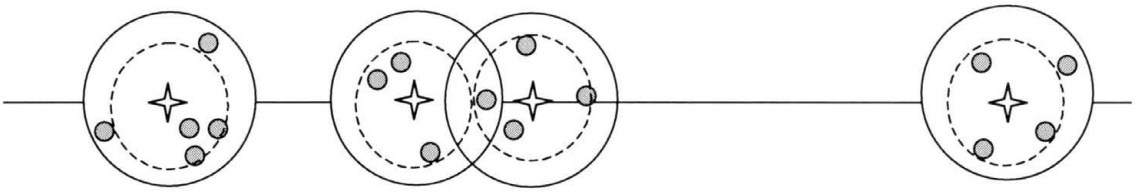

Figure 5.3 Effect of re-clustering on the clusterspace

The solid circles represent the clusters and the dashed circles represent the confidence clusters. After re-clustering, the centroids of the clusters represent the clusters. The centroid of the confidence cluster may not necessarily be the mean of all elements. However, all elements of the confidence cluster are guaranteed to satisfy the cluster bound requirement.

Assumption : 1.Elementset E exists

2. E is clustered and clusterspace U_E exists

Procedure :

1. Initialize the Clusterspace U_E to have

1a.Confidence Radius $rConf$

1b.Initialize Confidence Freq of all existing clusters to 0

2. For each element e in E

2a. Find $c_{\min}(e)$ in the clusterspace U_E

2b. If $c_{\min}(e)$ exists and $Dist_e\left(G_{c_{\min}}, e\right) \leq rConf_c$

Increment $freqConf_c$ by 1

End For

5.4 Clustering Principle and Soundness

5.4.1 Limitations of representative values and cluster radius

By using the cluster centroids of hidden layer neuron activation values, the output of a hidden layer neuron is generalized A representative value is used in place of a group of similar values, which are close in the activation space The representative value is the mean of the values in the cluster The radius of the cluster has an impact on the output of the hidden layer neuron hence affecting the accuracy of the neural network Therefore, a large cluster radius can impair the accuracy of the network significantly We impose a bound on the radius of the cluster based on the desired accuracy of the network The derivation of the bound for cluster radius for a particular network is shown below

The range of the output values of a hidden layer neuron is

$$S_j^p = G_{c_j}^p \pm r_{c_j}$$

Where S_j^p is the output of the hidden neuron j after clustering, for pattern p

$G_{c_j}^p$ is the centroid of the cluster activated for neuron j, for pattern p

r_{c_j} is the radius of the cluster activated for neuron j, for pattern p

The output of the k^{th} output layer neuron, for a pattern p, is

$$S_k^p = Sig\left(\sum_{j=1}^{m} x_j \cdot v_{jk}\right)$$

where m is the number of hidden layer neurons

and $$Sig(x) = \frac{1}{1+e^{-a\,x}}$$

The tolerance ρ is defined as

$$\rho = S_k^p - d_k^p$$

where d_k^p is the desired output of neuron k, for pattern p

S_k^p is the actual output of neuron k, for pattern p

Let,

$$\gamma_c = \pm r_c$$

considering the worst case, if the representative value (the centroid) of each hidden layer cluster is used, the output of the k^{th} output layer neuron, for a pattern, would be

$$S'^p_k = Sig\left(\sum_{j=1}^{m}(x_j + \gamma_c)\cdot v_{jk}\right)$$

$$S'^p_k = Sig\left(\sum_{j=1}^{m} x_j \cdot v_{jk} + \sum_{j=1}^{m}\gamma_c \cdot v_{jk}\right)$$

For maintaining the accuracy of the network, $\left|S_k^p - S'^p_k\right| \le \rho$ must hold

This implies that,

$$\left|Sig\left(\sum_{j=1}^{m} x_j \cdot v_{jk}\right) - Sig\left(\sum_{j=1}^{m} x_j \cdot v_{jk} + \sum_{j=1}^{m}\gamma_c \cdot v_{jk}\right)\right| \le \rho \qquad (1)$$

For any increasing function $f(x)$, the following inequality is true

$$f(a+b) \le f(a) + f(b)$$

Therefore,

$$Sig\left(\sum_{j=1}^{m} x_j \cdot v_{jk} + \sum_{j=1}^{m}\gamma_c \cdot v_{jk}\right) \le Sig\left(\sum_{j=1}^{m} x_j \cdot v_{jk}\right) + Sig\left(\sum_{j=1}^{m}\gamma_c \cdot v_{jk}\right) \qquad (2)$$

Combining (1) and (2),

$$\left|Sig\left(\sum_{j=1}^{m} x_j \cdot v_{jk}\right) - Sig\left(\sum_{j=1}^{m} x_j \cdot v_{jk}\right) + Sig\left(\sum_{j=1}^{m}\gamma_c \cdot v_{jk}\right)\right| \le \rho$$

$$\left|Sig\left(\sum_{j=1}^{m}\gamma_c \cdot v_{jk}\right)\right| \le \rho$$

$$\left|Sig\left(\gamma_c \sum_{j=1}^{m} v_{jk}\right)\right| \le \rho$$

Denoting $\sum_{j=1}^{m} v_{jk}$ as $\sum v$

$$\frac{1}{1+e^{-a\gamma_c \sum v}} \leq \rho$$

$$1+e^{-a\gamma_c \sum v} \geq \frac{1}{\rho}$$

$$e^{-a\gamma_c \sum v} \geq \frac{1}{\rho}-1$$

Taking natural logarithm on both sides,

$$-a\cdot\gamma_c \cdot \sum v \geq \ln\left(\frac{1}{\rho}-1\right)$$

$$-\gamma_c \geq \frac{\ln\left(\frac{1}{\rho}-1\right)}{a\cdot\sum v}$$

$$\gamma_c \leq \frac{\ln\left(\frac{1}{\rho}-1\right)}{-a\cdot\sum v}$$

Since the value $\sum v$ is different for each output neuron k, we use

$$\sum v^* = \max_k \left| a\cdot\sum_{j=1}^{m} v_{jk} \right|$$

Therefore the upper bound for r_c is

$$|r_c| \leq \frac{\ln\left(\frac{1}{\rho}-1\right)}{\sum v^*}$$

Where a is the scaling factor for the Sigmoid function, typically between 0 01 and 0 005

5.4.2 Confidence Radius

The confidence radius for re-clustering is typically set to be one-half of the cluster radius to eliminate any possible overlaps among clusters.

$$rConf_c = \frac{1}{2} r_c$$

5.4.3 Illustrative example of adaptable clustering procedure

Below is a plot of activation values for a hidden layer neuron j, for 81 input patterns. The red ovals show possible clusters or regions of activity.

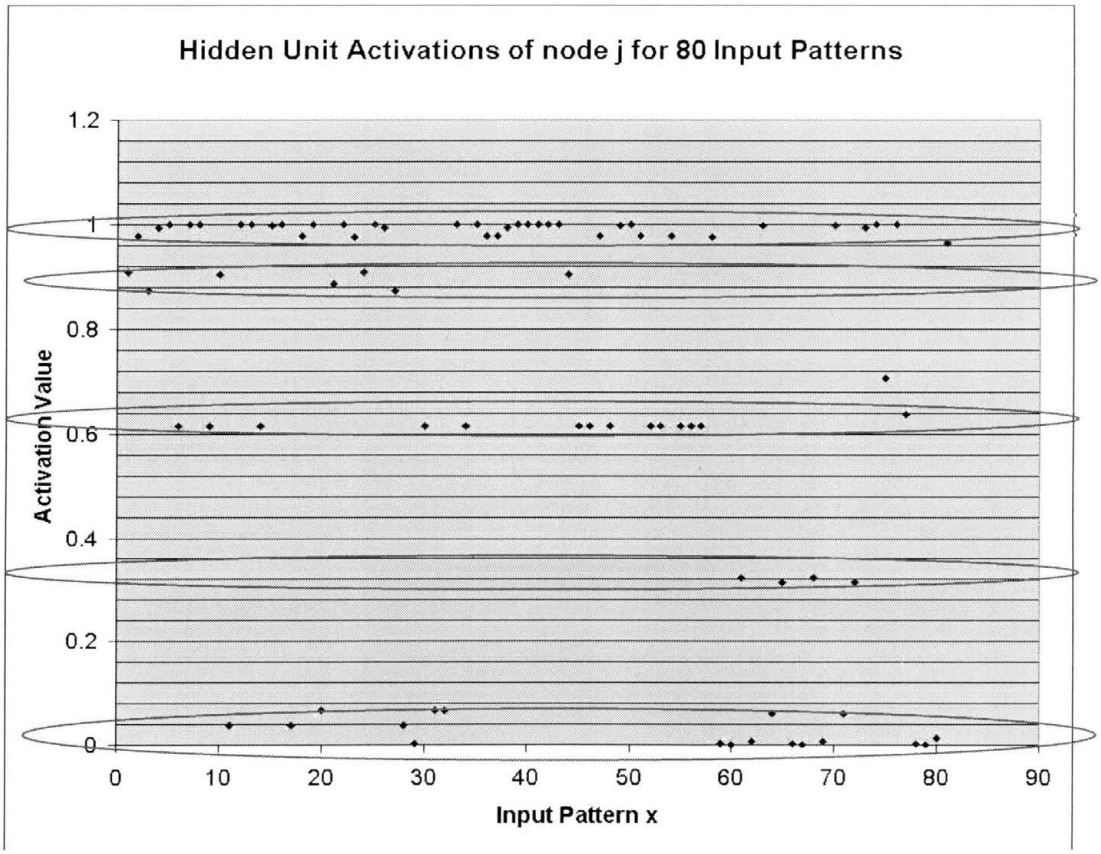

Figure 5.4 Plot showing Hidden Layer Neuron Activation Values of a Neuron j for 81 Input Patterns

The clustering process is applied to the above elementset and three possible results are shown below for different cluster radii. We can see the fineness of the clustering increase as the radius decreases. Inconsistent activity can be seen through clusters with relatively low freqeuncy.

Clustering Example 1 : Cluster Radius : 0.1

Cluster 0 : G=0.976982 | Freq=44

Cluster 1 : G=0.623161 | Freq=15

luster 2 : G=0.0261928 | Freq=18

Cluster 3 : G=0.318042 | Freq=4

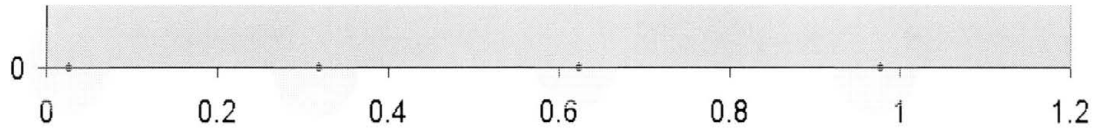

Figure 5.5 Clustering Example 1

Clustering Example 2 : Cluster Radius : 0.05

Cluster 0 : G=0.894242 | Freq=7

Cluster 1 : G=0.992635 | Freq=37

Cluster 2 : G=0.61717 | Freq=14

Cluster 3 : G=0.0261928 | Freq=18

Cluster 4 : G=0.318042 | Freq=4

Cluster 5 : G=0.707038 | Freq=1

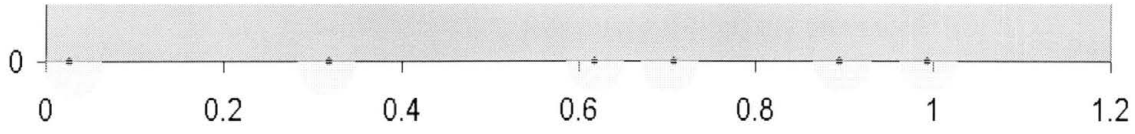

Figure 5.5 Clustering Example 2

Clustering Example 2 : Cluster Radius : 0.01

Cluster 0 : G=0.906447 | Freq=4

Cluster 1 : G=0.977062 | Freq=9

Cluster 2 : G=0.873794 | Freq=2

Cluster 3 : G=0.998891 | Freq=27

Cluster 4 : G=0.615573 | Freq=13

Cluster 5 : G=0.037844 | Freq=3

Cluster 6 : G=0.0644019 | Freq=5

Cluster 7 : G=0.886316 | Freq=1

Cluster 8 : G=0.00359298 | Freq=10

Cluster 9 : G=0.318042 | Freq=4

Cluster 10 : G=0.707038 | Freq=1

Cluster 11 : G=0.63793 | Freq=1

Cluster 12 : G=0.963879 | Freq=1

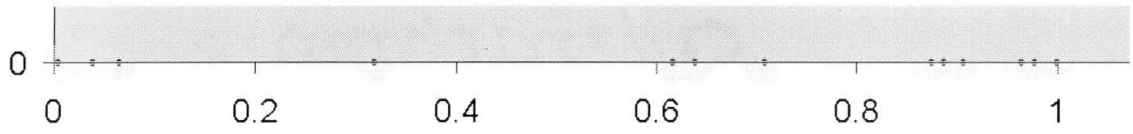

Figure 5.5 Clustering Example 3

5.4.4 Confidence frequency calculation and re-clustering

The effects of re-clustering are shown below for the same elementset with a cluster radius of 0.05 and different confidence radii. As can be seen from the examples below the confidence frequency is always less than the cluster frequency, since the confidence radius is less then the cluster radius.

Re-Clustering Example 1 . Cluster Radius . 0 05 , Confidence Radius 0.025

Cluster 0 G=0 894242 | Freq=7 | Conf Freq=7

Cluster 1 G=0 992635 | Freq=37 | Conf Freq=36

Cluster 2 G=0 61717 | Freq=14 | Conf Freq=14

Cluster 3 G=0 0261928 | Freq=18 | Conf Freq=10

Cluster 4 G=0 318042 | Freq=4 | Conf Freq=4

Cluster 5 G=0 707038 | Freq=1 | Conf Freq=1

Re-clustering example 2 . Cluster Radius 0 05 , Confidence Radius 0 01

Cluster 0 G=0 894242 | Freq=7 | Conf Freq=3

Cluster 1 G=0 992635 | Freq=37 | Conf Freq=27

Cluster 2 G=0 61717 | Freq=14 | Conf Freq=13

Cluster 3 G=0 0261928 | Freq=18 | Conf Freq=0

Cluster 4 G=0 318042 | Freq=4 | Conf Freq=4

Cluster 5 G=0 707038 | Freq=1 | Conf Freq=1

CHAPTER 6. RULE EXTRACTION

6.0 Overview

This chapter deals with the final phase of the process i e extracting the rules from the network Section 6 1 discusses the rule extraction principle used in our work Section 6 2 describes the Centroid Activation Layer Section 6 3 discusses the framework parameters Section 6 4 describes the extraction procedure for existing rules Section 6 5 describes the procedure for extraction of predicted rules Section 6 6 describes the compression of extracted rules Section 6 7 discusses the significance, benefits and limitations of our approach

In this final phase of the process, the knowledge acquired by the trained neural network is extracted in the form of rules The procedure uses the generalization of the hidden layer neuron activation values to extract the rules along with control parameters to check the confidence of the rules

6.1 Learning the Correlations in the Data Patterns

A neural network is a mapping network that learns to approximate a complex functional relationship between the input and output patterns [Mehrotra 1997] An example of the problem space in terms of input-output pair mappings is depicted in Figure 6 1

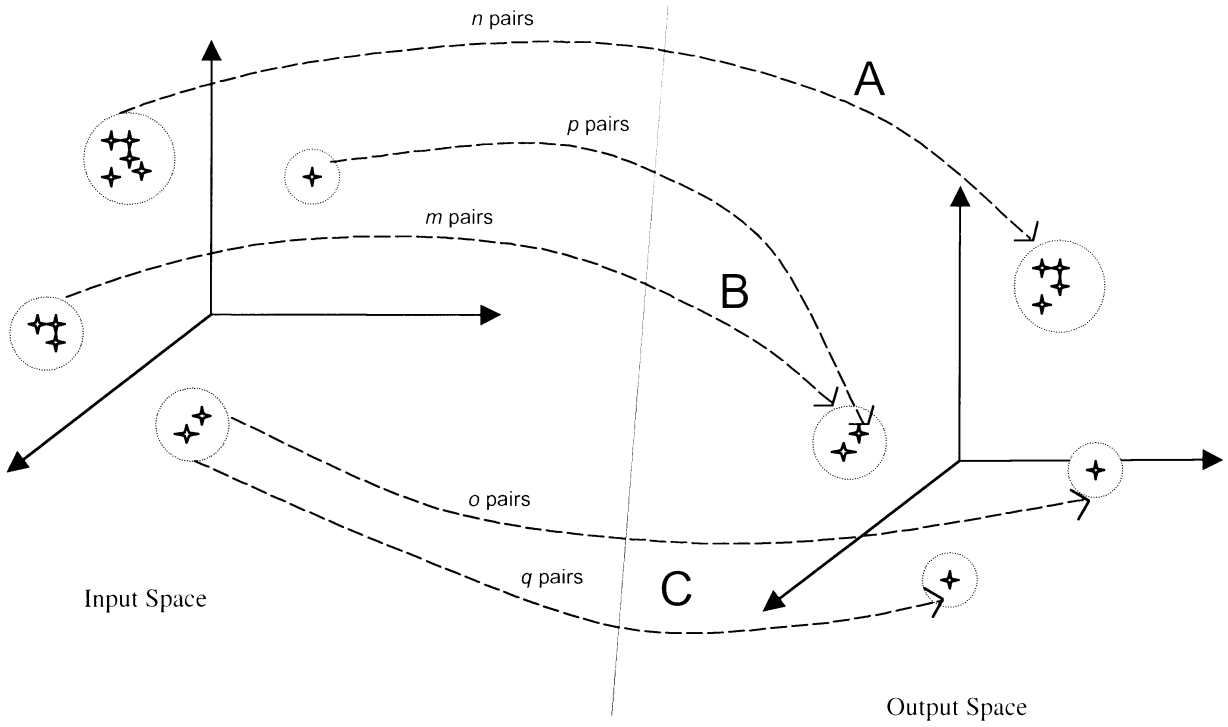

Figure 6.1 Type of mappings that exist in a given dataset

In any dataset there can be 3 types of mappings. Type A mapping is from similar input patterns to similar output patterns. They are said to have consistent mapping, i.e. close input pairs (closely grouped in input space) map to close output pairs (closely grouped in output space). Type B mapping is from different regions in the input space to similar regions in the output space. Type C mapping is from similar region in the input space to various regions in the output space. Type C is inconsistent mapping, since similar inputs are mapped to different outputs. In the above example, there are *n* pairs of type A mapping, *p* + *m* pairs of type B mapping, and *o* + *q* pairs of type C mappings.

Consistent patterns lead to strong learning. For example, patterns of type A mapping are strongly learned, whereas patterns of type C mapping are inconsistent.

The frequencies of these patterns largely affect the learning and generalization capability of the neural network Frequent patterns strengthen the learning For example, for patterns of type C mapping, if either *o* or *q* is largely greater than the other, the one with the high frequency dominates the mapping and one with the low frequency should be filtered out

The filtering process removes the inconsistent patterns of type C mapping from the dataset The inconsistency of a particular pattern can be measured in terms of its error with respect to the mean error For patterns in Figure 6 1, in the above example, the filtering process removes either set *o* or set *p* patterns, whichever that has a higher error, or both sets, if both sets have relatively high error with respect to the whole dataset

	High Frequency	Low Frequency
Type A	Strong Mapping with high confidence	Strong mapping with low confidence
Type B	Strong Mapping with high confidence	Weak mapping with low confidence
Type C	Inconsistent and weakens the generalizability of the dataset If one pattern dominates by frequency, that is learned strongly	Weak patterns, they can be filtered without affecting the accuracy of learning

Table 6 1 Different types of Mappings in a dataset and their effect on neural network learning

The key to the extraction process is the clustering of hidden neuron activation values of a trained neural network. Since the activation values of the hidden neurons are distributed, clustering helps to identify regions of activations along with the frequency of such activities. Also clustering generates a representative value for such region by which we can retrieve generalized outputs. Since we utilize a desired confidence frequency, we can examine the level of activities in all regions across the entire hidden layer. Only patterns which satisfy the desired confidence frequency across the entire hidden layer, or a percentage of the hidden layer are considered. This ensures that inconsistent patterns and those which fall within the regions with low level of activity are not considered.

6.2 Centroid Activation Layer

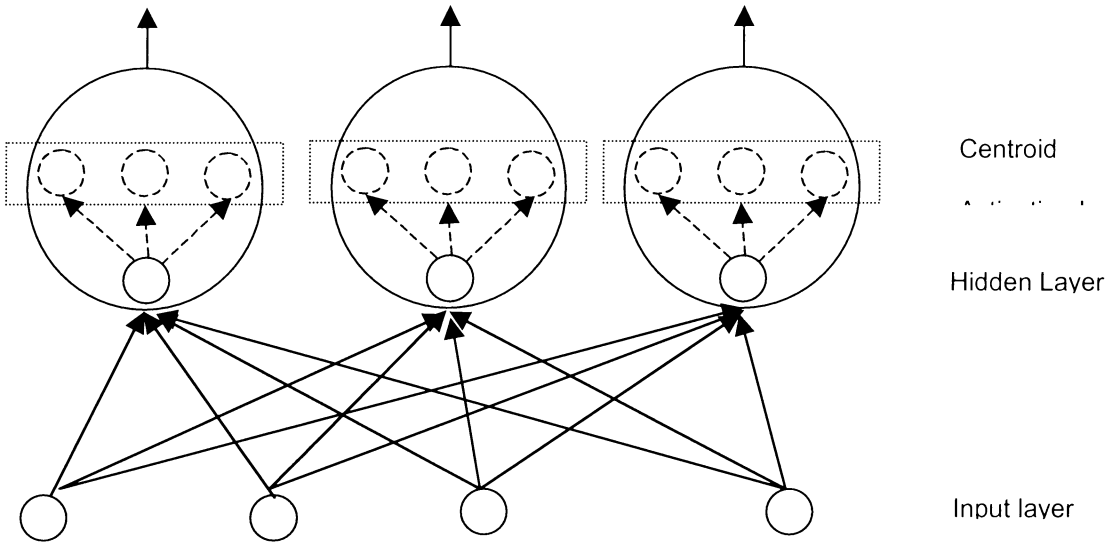

Figure 6.2 Centroid Activation Layer

The following describes the steps involved in building the centroid activation layer.

Initial Condition : a. Trained neural network

b. Dataset D_f used for training the neural network.

procedure :

1.For each input pattern p in D_f, compute the hidden layer neuron activation values $HAct_j$ of all hidden layer neurons $j, \quad 1 \le j \le m$

2. Create clusterspaces U_{HAct_j} for all hidden layer neurons, $1 \le j \le m$, using the required cluster radius.

3. Create clusterspaces U_{HConf_j} for all hidden layer neurons, $1 \le j \le m$, using the confidence radius.

Theoretically, the hidden neuron centroid activation layer superimposes the hidden layer neurons by replacing each neuron with its clusterspace centroid as demonstrated in chapter 5 The activation value of a hidden layer neuron j, for a pattern p, is replaced with the centroid of a cluster c, if a) the activation value falls within the confidence radius of the cluster, and b) the confidence frequency of the cluster meets the required frequency defined for the extraction phase If either of these conditions is not satisfied, the centroid activation layer outputs a zero (no activation) for that particular hidden layer neuron activation

If a desired percentage of the centroid activation layer units are activated, then the extraction phase can proceed This provides some flexibility and tolerance to the extraction phase where a small percentage of hidden layer activation values may not belong to a confidence cluster or the confidence cluster may not have the required frequency

6.3 Parameters of the Extracted Rules

The rules extracted can be defined with the following parameters

1 Accuracy of Trained Network — Percentage of patterns correctly classified by the network

2 Percentage of Filtered Patterns — Percentage of patterns filtered out during training process

3 Cluster radius — radius of the clusters in the clusterspace The confidence radius is calculated with

respect to the cluster radius

4 Confidence Frequency — Desired percentage of patterns in the confidence clusters

5 Hidden layer activation level — Percentage of hidden layer neurons that are activated

6.4 Extraction of Existing Rules

Initial Condition a Trained network with Centroid Activation Layer added

b Patterns which are classified correctly, D_c, from the original dataset D

c $Freq_{conf}$ is defined based on the desired confidence of rules

Procedure

1 For Each Pattern p in D_c

2 Present the pattern to the input layer

3 Calculate the activation value y_j^p for each hidden layer neuron

4 For each hidden layer neuron j (in parallel)

4a If ($Dist(G_c, y_j^p) \leq rConf$ AND $freqConf_c \geq Freq_{Conf}$)

$$GAct_j^p = G_c$$

else

$$GAct_j^p = 0$$

End for

5 If the percentage of activated hidden layer neurons exceeds the desired hidden layer activation level, propagate the hidden layer values to the output layer

End For

The input-output pairs that satisfy the rigorous extraction phase represent generalization and correlations that exist in the dataset The level of generalization and correlation is controlled by confidence frequency and radius as well as the desired hidden layer activation level There may be many duplicates in this set Further processing and interpretation needs to be performed to determine the rules A brief explanation is provided in Section 6 6

6.5 Extraction of Predicted Rules

Neural networks are good in generalizing the data model underlying the dataset This property of neural networks can be used to predict rules from patterns that are not in the dataset The accuracy of prediction depends on the generalizability and accuracy of the trained network

The extraction phase can be performed with all possible combinations of the input patterns Since all possible combinations of the input patterns include all input patterns in the dataset D_c, the input-output pairs extracted includes all those identified in the extraction of existing rules Any additional input-output pairs represent predicted rules

6.6 Compression of Input-Output Pairs

The set of input—output pairs obtained as a result of the extraction process needs to be processed to identify the rule set

1 Remove any duplicate input-output pairs

2 Group the input-output pairs, which have the same output

3 If possible, combine the input patterns in each group to produce a single rule

4 Each remaining input-output pair represents a single rule

6.7 Significance, Benefits and Limitations

In our work, we have provided a framework for knowledge discovery using neural networks and defined a process of extracting rules with control parameters The significance of this process is discussed below

Simplicity

The process is simple The complexity of the hidden layer neurons or the architecture of the network does not bear a restriction on this process The extraction phase of this process is fast and resembles the recall phase of a neural network

Control

The most important contribution of this process is providing a framework for control paramters for the knowledge discovery process The framework provides control parameters such as accuracy of training, filtering of inconsistent data during training, generalized regions of activations with representative values, confidence frequency indicator for extracting rules with desired confidence, and acceptable tolerance level

The analysis of the data is dynamic once the neural network is trained The hidden unit activation values can be clustered and re-clustered as many times as desired for different radius and frequency values, and the rules can be extracted for different confidence frequencies and tolerance

Confidence

If the correlation of input-output pairs is weak in terms of confidence frequency, the extracted rules would have low level of confidence On the other hand, if the correction is strong in terms of the confidence frequency, the extracted rules would have high level of confidence

Prediction

The predictive capability of rules which can be generalized from the knowledge learned by the network is a desirable byproduct of the process

Insignificance of attributes

Another important aspect of the process is the identification of insignificant attributes in the rules The process maps *m* inputs to *n* outputs and defines a relationship among them It is possible that some of the input or output attributes may be eliminated from the extracted rules These attributes are identified by input or output neurons that are not activated

Soundness of prıncıple

The clusterıng phase ıs performed wıth cluster radıus less than the computed cluster radıus bound Thıs ensures that the accuracy of the network ıs not compromısed

CHAPTER 7. APPLICATIONS AND ANALYSIS

7.0 Overview

In this chapter, we demonstrate the effectiveness and flexibility of our process via several applications. Section 7.1 presents an application of discovering trends in crimes across cities in the USA. Section 7.2 presents an application which discovers dominant rules which determine user CPU usage, given various system measurements. Section 7.3 presents an application which finds correlations between dietary characteristics of a person and Plasma Retinol and Beta-Carotene levels in the body. Section 7.4 presents an application which discovers dominant rules which determine the body fat percentage of a person, given various body measurements. Section 7.5 presents an applications which finds correlations relating physical and environmental characteristics to mortality rate.

7.1 Discovering Trends in Crimes Across Cities in the USA

Description

This is an application where the trends in the frequencies of crimes are analyzed using statistical datasets. The datasets were obtained from three different sources namely 1) US Census, 2) Uniform Crime Reports published annually by the Federal Bureau of Investigation, and 3) Unemployment Information from the Bureau of Labor Statistics. The dataset correlates frequencies of different crimes with respect to the demographic characteristics of 6100 towns across the United States. We analyze the frequencies of four types of crimes (murder, rape, robbery, and auto theft) for year 1999 with respect to the demographics of the cities to discover trends.

The dataset is divided into 3 segments based on the population of cities.

Category of Towns/Cities	Population	No. of Patterns
Small	0-20k	4706
Medium	20k-100k	1193
Large	100k-7Million	201

Table 7.1 Categories of Towns/Cities by Population

Variable dictionary and encoding

Each dataset was analyzed for four types of crimes. The variables of the dataset are described below:

Variable	Description
POP	Population
SINGP	Percentage of single-parent households
MINOR	Percentage of Minorities
YOUNG	Percentage of young people (between the ages of 15 and 24)
HOMEOW	Percentage of Home Owners
SAMEHO	Percentage of people living in the same house since 1985
UNEMPL	Percentage of Unemployed
MURDER	No. of murders
RAPE	No. of rape occurrences
ROB	No. of robberies
AUTO	No. of auto thefts

Table 7.2 Description of Variables for Crime Data

Variable	No. of Nodes	Intervals
POP (small)	5	[0-4k],(4k-8k],(8k,12k],(12k-16k],(16k-20]
POP (medium)	5	(20k-40k],(40k-60k],(60k-80k],(80k-90k],(90k-100k]
POP (large)	5	(100k-130k],(130k-160k],(160k-200k],(200k-500k],500k+
SINGP	7	[0-5],(5-7],(7-9],(9-11],(11-14],(14-20],(20-100]
MIN	6	[0-5],(5-10],(10-20],(20-40],(40-70],(70-100]
YOUNG	7	[0-12],(12-13],(13-14],(14-15],(15-17],(17-25],(25-100]
HOMEOW	7	[0-40],(40-50],(50-60],(60-70],(70-80],(80-90],(90-100]
SAMEHO	6	[0-45],(45-50],(50-55],(55-60],(60-65],(65-100]
UNEMPL	6	[0-4],(4-6],(6-8],(8-12],(12-20],(20-100]
MURDER	4	0,(1-5],(5-10],10+
RAPE	5	0,(1-5],(5-10],(10-70],70+
ROB	5	0,(1-5],(5-10],(10-100],100+
AUTO	5	[0-10],(10-100],(100-500],(500-1000],1000+

Table 7.3 Data Encoding for Crime Data

Network Architecture and Training Parameters:

Number of Input Nodes : 45
Number of Hidden Nodes : 60
Number of Output Nodes : 4-5
Error Tolerance : 0.001
Learning Rate : 0.4
MAX Cycles : 10000
Penalty Factor HI : 0.03
Penalty Factor OH : 0.01

Table 7.4 Network Architecture for Crime Data

Training and Pruning:

Small Towns				
TEST ID :	Murder	Rape	Robbery	Auto-thefts
Number of Training Patterns :	4706	4706	4706	4706
Percentage of Patterns filtered :	4%	10%	10%	6%
Number of cycles of Training :	10,000	10,000	10,000	10,000
Accuracy Acieved :	99.88%	97.88%	98.12%	99.61%
No. correctly recognized :	4549	4137	4117	4489
No. pruned : (IH | HO)	392 | 94	45 | 68	37 | 76	52 | 97
Nodes pruned :	None	None	None	None

Table 7.5 Training Results for Crime Data — Small Towns

Medium Towns				
TEST ID :	Murder	Rape	Robbery	Auto-thefts
Number of Training Patterns :	1193	1193	1193	1193
Percentage of Patterns filtered :	9%	10%	10%	9%
Number of cycles of Training :	10,000	10,000	10,000	10,000
Accuracy Acieved :	99.90%	99.71%	100%	99.90%
No. correctly recognized :	1144	1105	1109	1128
No. pruned : (IH | HO)	173 | 126	90 | 112	169 | 129	131 | 153
Nodes pruned :	None	None	None	None

Table 7.6 Training Results for Crime Data — Medium Towns

Large Cities				
TEST ID :	Murder	Rape	Robbery	Auto-thefts
Number of Training Patterns :	201	201	201	201
Percentage of Patterns filtered :	1%	2%	1%	1%
Number of cycles of Training :	10,000	10,000	10,000	10,000
Accuracy Acieved :	100%	100%	100%	100%
No. correctly recognized :	200	199	199	198
No. pruned : (IH | HO)	395 | 54	564 | 68	478 | 33	457 | 83
Nodes pruned :	16 | 43	16 | 43	16 | 43	16 | 43

Table 7.7 Training Results for Crime Data — Large Cities

Summary of Extraction Tests:

$$\text{Cluster radius bound, } |r_c| \leq \frac{\ln\left(\frac{1}{\rho} - 1\right)}{\sum v^*}$$

$\sum v^*$ calculated to be $\cong$ 20-24

Sigmoid scaling factor, $a = 0.01$

Error Tolerance, $\rho = 0.01$

Therefore, The bound on the cluster radius was found to be $\cong$ 0.2

No. of rules extracted:

Small towns

Existing Rules		Murder	Rape	Robbery	Auto Thefts
Frequency		30	30	30	30
100% Pass	r=0.2	0	0	0	0
96% Pass	r=0.2	0	4	5	3
95% Pass	r=0.2	1	-	-	-
Predicted Rules					
96%-97% Pass	r=0.2	10	5	4	8

Table 7.8 No. of Extracted Rules — Small Towns

Medium Cities

Existing Rules		Murder	Rape	Robbery	Auto Thefts
Frequency		30	25	25	30
100% Pass	r=0.2	0	0	0	0
96% Pass	r=0.2	1	0	0	0
95% Pass	r=0.2	3	2	2	2
Predicted Rules					
95%-97% Pass		3(r=0.2)	4(r=0.1)	7(r=0.1)	0(r=0.2)

Table 7.9 No. of Extracted Rules — Medium Towns

Large Cities

Existing Rules		Murder	Rape	Robbery	Auto Thefts
Frequency		25	25	30	20
100% Pass	r=0.2	0	0	0	0
96% Pass	r=0.2	0	0	3	0
95% Pass	r=0.2	1	1	8	1
Predicted Rules					
		1(r=0.2,95%)	4(r=0.1,99%)	6(r=0.1,96%)	2(r=0.2,97%)

Table 7.10 No. of Extracted Rules — Large Cities

Knowledge Discovery:

The knowledge discovered for the 3 types of populations with respect to 4 types of crimes are as follows

Trends in Small Towns

Existing

Murder (Cluster radius=0.2, frequency=30%, Pass=96%)

```
RULE 0 :
IF    Population between 0 and 4k
      Minority between 0 and 5
      Unemployment between 0 and 4
      Single-Parent between 7 and 9
      Same-House between 50 and 55
      Young between 14 and 15
      HomeOwners between 70 and 80
THEN  Murder is 0
```

Rape (Cluster radius=0 2, frequency=30%, Pass=96%)

```
RULE 0 :
IF    Population between 0 and 4k
      Minority between 20 and 40
      Unemployment between 4 and 6
      Single-Parent between 7 and 9
      Same-House between 50 and 55
      Young between 13 and 14
      HomeOwners between 70 and 80
THEN  Rape between 1 and 5

RULE 1 :
IF    Population between 0 and 4k
      Minority between 0 and 5
      Unemployment between 6 and 8
      Single-Parent between 7 and 9
      Same-House between 60 and 65
      Young between 0 and 12
      HomeOwners between 70 and 80
THEN  Rape is 0
```

Robbery (Cluster radius=0 2, frequency=30%, Pass=96%)

```
RULE 0 :
IF    Population between 4k and 8k
      Minority between 0 and 5
```

```
       Unemployment between 0 and 4
       Single-Parent between 7 and 9
       Same-House between 45 and 50
       Young between 12 and 13
       HomeOwners between 60 and 70
THEN   Robbery between 1 and 5

RULE 1 :
IF     Population between 0 and 4k
       Minority between 5 and 10
       Unemployment between 4 and 6
       Single-Parent between 7 and 9
       Same-House between 50 and 55
       Young between 12 and 13
       HomeOwners between 60 and 70
THEN   Robbery between 1 and 5

RULE 2 :
IF     Population between 0 and 4k
       Minority between 5 and 10
       Unemployment between 4 and 6
       Single-Parent between 5 and 7
       Same-House between 50 and 55
       Young between 12 and 13
       HomeOwners between 60 and 70
THEN   Robbery between 1 and 5

RULE 3 :
IF     Population between 0 and 4k
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 0 and 5
       Same-House between 0 and 45
       Young between 0 and 12
       HomeOwners between 60 and 70
THEN   Robbery between 1 and 5
```

Auto-Thefts (Cluster radius=0 2, frequency=30%, Pass=96%)

```
RULE 0 :
IF     Population between 4k and 8k
       Minority between 0 and 5
       Unemployment between 6 and 8
       Single-Parent between 9 and 11
       Same-House between 65 and 80+
       Young between 13 and 14
       HomeOwners between 60 and 70
THEN   Auto-Thefts between 1 and 5

RULE 1 :
IF     Population between 8k and 12k
       Minority between 0 and 5
       Unemployment between 8 and 12
```

```
       Single-Parent between 7 and 9
       Same-House between 55 and 60
       Young between 13 and 14
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 0

RULE 2 :
IF     Population between 12k and 16k
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 7 and 9
       Same-House between 55 and 60
       Young between 13 and 14
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 0
```

Prediction

Murder (Cluster radius=0 2, frequency=30%, Pass=96%)

```
RULE 0 :
IF     Population between 0 and 4k
       Minority between 0 and 5
       Unemployment between 4 and 6
       Single-Parent between 7 and 9
       Same-House between 60 and 65
       Young between 14 and 15
       HomeOwners between 70 and 80
THEN   Murder is 0

RULE 1 :
IF     Population between 0 and 4k
       Minority between 0 and 5
       Unemployment between 4 and 6
       Single-Parent between 7 and 9
       Same-House between 50 and 55
       Young between 14 and 15
       HomeOwners between 70 and 80
THEN   Murder is 0

RULE 2 :
IF     Population between 0 and 4k
       Minority between 0 and 5
       Unemployment between 8 and 12
       Single-Parent between 5 and 7
       Same-House between 45 and 50
       Young between 13 and 14
       HomeOwners between 70 and 80
THEN   Murder is 0

RULE 3 :
IF     Population between 0 and 4k
```

```
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 0 and 5
       Same-House between 50 and 55
       Young between 13 and 14
       HomeOwners between 70 and 80
THEN   Murder is 0

RULE 4 :
IF     Population between 12k and 16k
       Minority between 0 and 5
       Unemployment between 4 and 6
       Single-Parent between 0 and 5
       Same-House between 45 and 50
       Young between 13 and 14
       HomeOwners between 70 and 80
THEN   Murder is 0

RULE 5 :
IF     Population between 0 and 4k
       Minority between 0 and 5
       Unemployment between 4 and 6
       Single-Parent between 0 and 5
       Same-House between 45 and 50
       Young between 13 and 14
       HomeOwners between 70 and 80
THEN   Murder is 0

RULE 6 :
IF     Population between 0 and 4k
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 0 and 5
       Same-House between 0 and 45
       Young between 13 and 14
       HomeOwners between 70 and 80
THEN   Murder is 0

RULE 7 :
IF     Population between 4k and 8k
       Minority between 0 and 5
       Unemployment between 4 and 6
       Single-Parent between 9 and 11
       Same-House between 45 and 50
       Young between 12 and 13
       HomeOwners between 70 and 80
THEN   Murder is 0

RULE 8 :
IF     Population between 0 and 4k
       Minority between 0 and 5
       Unemployment between 4 and 6
       Single-Parent between 5 and 7
       Same-House between 45 and 50
```

```
       Young between 25 and 39+
       HomeOwners between 60 and 70
THEN   Murder is 0
```

Rape (Cluster radius=0 2, frequency=30%, Pass=97%)

```
RULE 0 :
IF     Population between 0 and 4k
       Minority between 20 and 40
       Unemployment between 4 and 6
       Single-Parent between 9 and 11
       Same-House between 0 and 45
       Young between 13 and 14
       HomeOwners between 50 and 60
THEN   Rape is 0

RULE 1 :
IF     Population between 4k and 8k
       Minority between 5 and 10
       Unemployment between 4 and 6
       Single-Parent between 5 and 7
       Same-House between 60 and 65
       Young between 15 and 17
       HomeOwners between 40 and 50
THEN   Rape between 1 and 5
```

Robbery (Cluster radius=0 2, frequency=30%, Pass=97%)

```
RULE 1 :
IF     Population between 0 and 4k
       Minority between 5 and 10
       Unemployment between 12 and 20
       Single-Parent between 7 and 9
       Same-House between 45 and 50
       Young between 17 and 25
       HomeOwners between 80 and 90
THEN   Robbery is 0

RULE 2 :
IF     Population between 12k and 16k
       Minority between 5 and 10
       Unemployment between 4 and 6
       Single-Parent between 5 and 7
       Same-House between 0 and 45
       Young between 17 and 25
       HomeOwners between 80 and 90
THEN   Robbery between 1 and 5

RULE 3 :
IF     Population between 0 and 4k
       Minority between 10 and 20
       Unemployment between 4 and 6
       Single-Parent between 9 and 11
       Same-House between 65 and 80+
```

```
       Young between 17 and 25
       HomeOwners between 40 and 50
THEN   Robbery between 1 and 5
```

Auto-Thefts (Cluster radius=0 2, frequency=30%, Pass=97%)

```
RULE 0 :
IF     Population between 4k and 8k
       Minority between 0 and 5
       Unemployment between 4 and 6
       Single-Parent between 14 and 20
       Same-House between 50 and 55
       Young between 17 and 25
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 0

RULE 1 :
IF     Population between 4k and 8k
       Minority between 5 and 10
       Unemployment between 20 and 40+
       Single-Parent between 7 and 9
       Same-House between 60 and 65
       Young between 17 and 25
       HomeOwners between 60 and 70
THEN   Auto-Thefts between 1 and 5

RULE 2 :
IF     Population between 4k and 8k
       Minority between 5 and 10
       Unemployment between 0 and 4
       Single-Parent between 7 and 9
       Same-House between 60 and 65
       Young between 17 and 25
       HomeOwners between 60 and 70
THEN   Auto-Thefts between 1 and 5

RULE 3 :
IF     Populatıon between 4k and 8k
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 7 and 9
       Same-House between 60 and 65
       Young between 17 and 25
       HomeOwners between 60 and 70
THEN   Auto-Thefts between 1 and 5

RULE 4 :
IF     Population between 4k and 8k
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 7 and 9
       Same-House between 55 and 60
       Young between 17 and 25
       HomeOwners between 60 and 70
```

```
THEN  Auto-Thefts between 0 and

RULE 5 :
IF    Population between 12k and 16k
      Minority between 0 and 5
      Unemployment between 8 and 12
      Single-Parent between 5 and 7
      Same-House between 55 and 60
      Young between 17 and 25
      HomeOwners between 60 and 70
THEN  Auto-Thefts between 1 and 5

RULE 6 :
IF    Population between 8k and 12k
      Minority between 0 and 5
      Unemployment between 0 and 4
      Single-Parent between 5 and 7
      Same-House between 45 and 50
      Young between 17 and 25
      HomeOwners between 60 and 70
THEN  Auto-Thefts between 1 and 5

RULE 7 :
IF    Population between 12k and 16k
      Minority between 0 and 5
      Unemployment between 0 and 4
      Single-Parent between 14 and 20
      Same-House between 65 and 80+
      Young between 15 and 17
      HomeOwners between 60 and 70
THEN  Auto-Thefts between 1 and 5
```

Trends in Medium Cities

Existing

Murder (Cluster radius=0 2, frequency=30%, Pass=95%)

```
RULE 0 :
IF    Population between 20k and 40k
      Minority between 0 and 5
      Unemployment between 4 and 6
      Single-Parent between 9 and 11
      Same-House between 45 and 50
      Young between 12 and 13
      HomeOwners between 60 and 70
THEN  Murder is 0

RULE 1 :
IF    Population between 20k and 40k
```

```
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 5 and 7
       Same-House between 65 and 80+
       Young between 12 and 13
       HomeOwners between 70 and 80
THEN   Murder between 1 and 5

RULE 2 :
IF     Population between 20k and 40k
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 5 and 7
       Same-House between 50 and 55
       Young between 12 and 13
       HomeOwners between 80 and 90
THEN   Murder is 0

RULE 3 :
IF     Population between 20k and 40k
       Minority between 0 and 5
       Unemployment between 0 and 4
       Single-Parent between 5 and 7
       Same-House between 50 and 55
       Young between 12 and 13
       HomeOwners between 70 and 80
THEN   Murder is 0
```

Rape (Cluster radius=0 2, frequency=25%, Pass=95%)

```
RULE 0 :
IF     Population between 20k and 40k
       Minority between 10 and 20
       Unemployment between 0 and 4
       Single-Parent between 5 and 7
       Same-House between 0 and 45
       Young between 12 and 13
       HomeOwners between 60 and 70
THEN   Rape between 1 and 5

RULE 1 :
IF     Population between 20k and 40k
       Minority between 5 and 10
       Unemployment between 4 and 6
       Single-Parent between 11 and 14
       Same-House between 45 and 50
       Young between 14 and 15
       HomeOwners between 40 and 50
THEN   Rape between 1 and 10
```

Robbery (Cluster radius=0 2, frequency=25%, Pass=95%)

```
RULE 0 :
IF    Population between 20k and 40k
      Minority between 10 and 20
      Unemployment between 4 and 6
      Single-Parent between 7 and 9
      Same-House between 0 and 45
      Young between 13 and 14
      HomeOwners between 70 and 80
THEN  Robbery between 1 and 5

RULE 1 :
IF    Population between 20k and 40k
      Minority between 5 and 10
      Unemployment between 0 and 4
      Single-Parent between 7 and 9
      Same-House between 0 and 45
      Young between 17 and 25
      HomeOwners between 60 and 70
THEN  Robbery between 5 and 10
```

Auto-Thefts (Cluster radius=0 2, frequency=30%, Pass=95%)

```
RULE 0 :
IF    Population between 20k and 40k
      Minority between 0 and 5
      Unemployment between 4 and 6
      Single-Parent between 0 and 5
      Same-House between 65 and 80+
      Young between 12 and 13
      HomeOwners between 80 and 90
THEN  Auto-Thefts between 10 and 100

RULE 1 :
IF    Population between 20k and 40k
      Minority between 0 and 5
      Unemployment between 4 and 6
      Single-Parent between 9 and 11
      Same-House between 0 and 45
      Young between 14 and 15
      HomeOwners between 50 and 60
THEN  Auto-Thefts between 100 and 500
```

Prediction

Murder (Cluster radius=0 2, frequency=30%, Pass=97%)

```
RULE 0 :
IF    Population between 20k and 40k
      Minority between 0 and 5
      Unemployment between 0 and 4
      Single-Parent between 5 and 7
```

```
       Same-House between 50 and 55
       Young between 12 and 13
       HomeOwners between 80 and 90
THEN   Murder is 0

RULE 2 :
IF     Population between 20k and 40k
       Minority between 10 and 20
       Unemployment between 4 and 6
       Single-Parent between 5 and 7
       Same-House between 55 and 60
       Young between 14 and 15
       HomeOwners between 60 and 70
THEN   Murder is 0
```

Rape (Cluster radius=0 2, frequency=30%, Pass=96%)

```
RULE 0 :
IF     Population between 20k and 40k
       Minority between 20 and 40
       Unemployment between 0 and 4
       Single-Parent between 5 and 7
       Same-House between 45 and 50
       Young between 13 and 14
       HomeOwners between 50 and 60
THEN   Rape between 5 and 10

RULE 1 :
IF     Population between 20k and 40k
       Minority between 40 and 70
       Unemployment between 0 and 4
       Single-Parent between 7 and 9
       Same-House between 45 and 50
       Young between 17 and 25
       HomeOwners between 40 and 50
THEN   Rape between 1 and 10

RULE 2 :
IF     Population between 20k and 40k
       Minority between 10 and 20
       Unemployment between 0 and 4
       Single-Parent between 9 and 11
       Same-House between 45 and 50
       Young between 14 and 15
       HomeOwners between 40 and 50
THEN   Rape between 1 and 10

RULE 3 :
IF     Population between 20k and 40k
       Minority between 70 and 100
       Unemployment between 6 and 8
       Single-Parent between 5 and 7
       Same-House between 50 and 55
       Young between 13 and 14
```

```
      HomeOwners between 40 and 50
THEN  Rape between 1 and 10
```

Robbery (Cluster radius=0 2, frequency=30%, Pass=95%)

```
RULE 0 :
IF    Population between 60 and 80k
      Minority between 10 and 20
      Unemployment between 4 and 6
      Single-Parent between 7 and 9
      Same-House between 50 and 55
      Young between 15 and 17
      HomeOwners between 80 and 90
THEN  Robbery is 0 or between (10 and 100)

RULE 1 :
IF    Population between 20k and 40k
      Minority between 70 and 100
      Unemployment between 6 and 8
      Single-Parent between 5 and 7
      Same-House between 0 and 45
      Young between 0 and 12
      HomeOwners between 80 and 90
THEN  Robbery between 10 and 100

RULE 2 :
IF    Population between 20k and 40k
      Minority between 5 and 10
      Unemployment between 12 and 20
      Single-Parent between 20 and 40+
      Same-House between 65 and 80+
      Young between 25 and 39+
      HomeOwners between 60 and 70
THEN  Robbery is 0

RULE 3 :
IF    Population between 20k and 40k
      Minority between 20 and 40
      Unemployment between 4 and 6
      Single-Parent between 9 and 11
      Same-House between 50 and 55
      Young between 25 and 39+
      HomeOwners between 40 and 50
THEN  Robbery between 10 and 100

RULE 4 :
IF    Population between 20k and 40k
      Minority between 10 and 20
      Unemployment between 4 and 6
      Single-Parent between 9 and 11
      Same-House between 50 and 55
      Young between 25 and 39+
```

```
      HomeOwners between 40 and 50
THEN  Robbery between 10 and 100

RULE 5 :
IF    Population between 20k and 40k
      Minority between 5 and 10
      Unemployment between 0 and 4
      Single-Parent between 7 and 9
      Same-House between 50 and 55
      Young between 25 and 39+
      HomeOwners between 40 and 50
THEN  Robbery between 10 and 100

RULE 6 :
IF    Population between 20k and 40k
      Minority between 10 and 20
      Unemployment between 4 and 6
      Single-Parent between 20 and 40+
      Same-House between 60 and 65
      Young between 17 and 25
      HomeOwners between 40 and 50
THEN  Robbery between 10 and 100

RULE 7 :
IF    Population between 20k and 40k
      Minority between 5 and 10
      Unemployment between 8 and 12
      Single-Parent between 11 and 14
      Same-House between 50 and 55
      Young between 14 and 15
      HomeOwners between 40 and 50
THEN  Robbery between 10 and 100
```

Trends in Large Cities

Exıstıng

Murder (Cluster radıus=0 2, frequency=25%, Pass=95%)

```
RULE 0 :
IF    Population between 200k and 500k
      Minority between 20 and 40
      Unemployment between 8 and 12
      Single-Parent between 11 and 14
      Same-House between 50 and 55
      Young between 15 and 17
      HomeOwners between 50 and 60
THEN  Murder between 5 and 10
```

Rape (Cluster radius=0 2, frequency=25%, Pass=95%)

```
RULE 0 :
```

```
IF    Population between 200k and 500k
      Minority between 40 and 70
      Unemployment between 8 and 12
      Single-Parent between 14 and 20
      Same-House between 50 and 55
      Young between 15 and 17
      HomeOwners between 50 and 60
THEN  Rape between 10 and 70
```

Robbery (Cluster radius=0 2, frequency=30%, Pass=95%)

```
RULE 0 :
IF    Population between 160k and 200k
      Minority between 20 and 40
      Unemployment between 6 and 8
      Single-Parent between 11 and 14
      Same-House between 45 and 50
      Young between 15 and 17
      HomeOwners between 50 and 60
THEN  Robbery is 100+

RULE 1 :
IF    Population between 160k and 200k
      Minority between 20 and 40
      Unemployment between 6 and 8
      Single-Parent between 11 and 14
      Same-House between 45 and 50
      Young between 14 and 15
      HomeOwners between 50 and 60
THEN  Robbery is 100+

RULE 2 :
IF    Population between 130k and 160k
      Minority between 40 and 70
      Unemployment between 6 and 8
      Single-Parent between 11 and 14
      Same-House between 45 and 50
      Young between 17 and 25
      HomeOwners between 50 and 60
THEN  Robbery is 100+
```

Auto-Thefts (Cluster radius=0 2, frequency=20%, Pass=95%)

```
RULE 0 :
IF    Population between 100k and 130k
      Minority between 10 and 20
      Unemployment between 0 and 4
      Single-Parent between 5 and 7
      Same-House between 0 and 45
      Young between 0 and 12
      HomeOwners between 70 and 80
```

```
THEN  Auto-Thefts between 500 and 1000
```

Prediction

Murder (Cluster radius=0 2, frequency=25%, Pass=95%)

```
RULE 0 :
IF    Population between 200k and 500k
      Minority between 20 and 40
      Unemployment between 8 and 12
      Single-Parent between 11 and 14
      Same-House between 50 and 55
      Young between 15 and 17
      HomeOwners between 50 and 60
THEN  Murder between 5 and 10
```

Rape (Cluster radius=0 2, frequency=25%, Pass=100%)

```
RULE 0 :
IF    Population between 200k and 500k
      Minority between 70 and 100
      Unemployment between 6 and 8
      Single-Parent between 11 and 14
      Same-House between 0 and 45
      Young between 17 and 25
THEN  Rape is 70+

RULE 1 :
IF    Population between 200k and 500k
      Minority between 70 and 100
      Unemployment between 6 and 8
      Single-Parent between 11 and 14
      Same-House between 50 and 55
      Young between 13 and 14
THEN  Rape is 70+

RULE 2 :
IF    Population between 200k and 500k
      Minority between 70 and 100
      Unemployment between *20 and 40+*
      Single-Parent between 11 and 14
      Same-House between 50 and 55
      Young between 13 and 14
      HomeOwners between 60 and 70
THEN  Rape is 70+

RULE 3 :
IF    Population between 200k and 500k
      Minority between 70 and 100
      Unemployment between 0 and 4
```

```
       Single-Parent between 11 and 14
       Same-House between 50 and 55
       Young between 13 and 14
       HomeOwners between 60 and 70
THEN   Rape is 70+
```

Robbery (Cluster radius=0 2, frequency=30%, Pass=96%)

```
RULE 0 :
IF     Population between 100k and 130k
       Minority between 70 and 100
       Unemployment between 6 and 8
       Single-Parent between 11 and 14
       Same-House between 60 and 65
       Young between 17 and 25
       HomeOwners between 50 and 60
THEN   Robbery is 100+

RULE 1 :
IF     Population between 100k and 130k
       Minority between 70 and 100
       Unemployment between 6 and 8
       Single-Parent between 11 and 14
       Same-House between 45 and 50
       Young between 17 and 25
       HomeOwners between 50 and 60
THEN   Robbery is 100+

RULE 2 :
IF     Population between 100k and 130k
       Minority between 40 and 70
       Unemployment between 6 and 8
       Single-Parent between 11 and 14
       Same-House between 45 and 50
       Young between 17 and 25
       HomeOwners between 50 and 60
THEN   Robbery 100+

RULE 3 :
IF     Population between 200k and 500k
       Minority between 5 and 10
       Unemployment between 6 and 8
       Single-Parent between 11 and 14
       Same-House between 45 and 50
       Young between 17 and 25
       HomeOwners between 50 and 60
THEN   Robbery is 100+

RULE 4 :
IF     Population between 160k and 200k
       Minority between 5 and 10
       Unemployment between 6 and 8
```

```
       Single-Parent between 11 and 14
       Same-House between 45 and 50
       Young between 17 and 25
       HomeOwners between 50 and 60
THEN   Robbery is 100+

RULE 5 :
IF     Population between 100k and 130k
       Minority between 40 and 70
       Unemployment between 6 and 8
       Single-Parent between 11 and 14
       Same-House between 45 and 50
       Young between 17 and 25
       HomeOwners between 0 and 40
THEN   Robbery is 100+
```

Auto-Thefts (Cluster radius=0 2, frequency=20%, Pass=97%)

```
RULE 0 :
IF     Population between 200k and 500k
       Minority between 40 and 70
       Single-Parent between 11 and 14
       Same-House between 0 and 45
       Young between 17 and 25
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 1000+

RULE 1 :
IF     Population between 200k and 500k
       Minority between 40 and 70
       Unemployment between 8 and 12
       Single-Parent between 11 and 14
       Same-House between 0 and 45
       Young between 17 and 25
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 1000+

RULE 2 :
IF     Population between 500k+ and
       Minority between 0 and 5
       Unemployment between 8 and 12
       Single-Parent between 11 and 14
       Same-House between 0 and 45
       Young between 17 and 25
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 1000+

RULE 3 :
IF     Population between 100k and 130k
       Minority between 5 and 10
       Unemployment between 4 and 6
       Single-Parent between 14 and 20
       Same-House between 55 and 60
```

```
       Young between 13 and 14
       HomeOwners between 60 and 70
THEN   Auto-Thefts between 500 and 1000

RULE 4 :
IF     Population between 100k and 130k
       Minority between 5 and 10
       Unemployment between 4 and 6
       Single-Parent between 11 and 14
       Same-House between 45 and 50
       Young between 13 and 14
       HomeOwners between 60 and 70
THEN   Auto-Thefts between 500 and 1000

RULE 5 :
IF     Population between 200k and 500k
       Minority between 70 and 100
       Unemployment between 8 and 12
       Single-Parent between 11 and 14
       Same-House between 60 and 65
       Young between 0 and 12
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 1000+

RULE 6 :
IF     Population between 200k and 500k
       Minority between 5 and 10
       Unemployment between 0 and 4
       Single-Parent between 11 and 14
       Same-House between 60 and 65
       Young between 0 and 12
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 1000+

RULE 7 :
IF     Population between 200k and 500k
       Minority between 40 and 70
       Single-Parent between 11 and 14
       Same-House between 50 and 55
       Young between 0 and 12
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 1000+

RULE 8 :
IF     Population between 200k and 500k
       Minority between 40 and 70
       Unemployment between 12 and 20
       Single-Parent between 11 and 14
       Same-House between 50 and 55
       Young between 0 and 12
       HomeOwners between 60 and 70
THEN   Auto-Thefts is 1000+

RULE 9 :
```

```
IF    Population between 200k and 500k
      Minority between 70 and 100
      Single-Parent between 11 and 14
      Same-House between 0 and 45
      Young between 17 and 25
      HomeOwners between 40 and 50
THEN  Auto-Thefts is 1000+

RULE 10 :
IF    Population between 200k and 500k
      Minority between 70 and 100
      Unemployment between 12 and 20
      Single-Parent between 11 and 14
      Same-House between 0 and 45
      Young between 17 and 25
      HomeOwners between 40 and 50
THEN  Auto-Thefts is 1000+

RULE 11 :
IF    Population between 100k and 130k
      Minority between 70 and 100
      Unemployment between 8 and 12
      Single-Parent between 5 and 7
      Same-House between 50 and 55
      Young between 13 and 14
      HomeOwners between 40 and 50
THEN  Auto-Thefts between 500 and 1000

RULE 12 :
IF    Population between 200k and 500k
      Minority between 70 and 100
      Unemployment between 0 and 4
      Single-Parent between 11 and 14
      Same-House between 0 and 45
      Young between 0 and 12
      HomeOwners between 40 and 50
THEN  Auto-Thefts is 1000+

RULE 13 :
IF    Population between 100k and 130k
      Minority between 5 and 10
      Unemployment between 12 and 20
      Single-Parent between 9 and 11
      Same-House between 45 and 50
      Young between 13 and 14
      HomeOwners between 0 and 40
THEN  Auto-Thefts between 100 and 50
```

7.2 Computer Activity Database

Description

The computer activity database is a collection of a computer system activity measures The data was collected from a Sun Sparcstation 20/712 with 128 Mbytes of memory running in a multi-user university department Users would typically be doing a large variety of tasks ranging from accessing the internet, editing files or running CPU intensive programs Data was collected between 12 00 pm -18 00 pm which is when the machines would be busiest On both occasions, system activity was gathered every 5 seconds

The dataset was obtained from Delve , Department of computer science, University of Toronto, Canada The original dataset consisted of 8192 patterns We randomly selected 4000 patterns for our analysis The portion of time the CPU runs in user mode is correlated with the various measures of the system and predominant rules are extracted in this analysis

Data description and encoding:

Variable	*Description*
lread	Reads (transfers per second) between system memory and user memory
lwrite	writes (transfers per second) between system memory and user memory
scall	Number of system calls of all types per second
sread	Number of system read calls per second
swrite	Number of system write calls per second
fork	Number of system fork calls per second
exec	Number of system exec calls per second
rchar	Number of characters transferred per second by system read calls
wchar	Number of characters transfreed per second by system write calls
pgout	Number of page out requests per second
ppgout	Number of pages, paged out per second
pgfree	Number of pages per second placed on the free list
pgscan	Number of pages checked if they can be freed per second
atch	Number of page attaches (satisfying a page fault by reclaiming a page in memory) per second
pgin	Number of page in requests per second
ppgin	Number of pages paged in per second
pflt	Number of page faults caused by protection errors (copy on writes)
vflt	Number of page faults caused by address translation
runqsz	Process run queue size
freemem	Number of memory pages available to user processes
freeswap	Number of disk blocks available for page swapping
usr	Portion of time (%) that cpus run in user mode

Table 7 12 Data Description — Computer Active Database

Variable	No.of Nodes	Intervals
lread	4	[0-30],(30-60],(60-90],90+
lwrite	4	[0-30],(30-60],(60-90],90+
scall	6	<1k],(1k-2k],(2k-3k],(3k-4k],(5k-6k],6k+
sread	4	[0-300],(300-600],(600-900],900+
swrite	4	[0-300],(300-600],(600-900],900+
fork	4	[0-3],(3-6],(6-9],9+
exec	5	[0-3],(3-6],(6-9],(9-20],20+
rchar	4	[0-300],(300-600],(600-900],900+
wchar	4	[0-300],(300-600],(600-900],900+
pgout	3	[0-20],(20-40],(40-60]
ppgout	3	[0-20],(20-40],(40-60]
pgfree	4	<100],(100-200],(200-300],(300-400]
pgscan	4	<100],(100-200],(200-300],(300-400]
atch	4	[0-3],(3-6],(6-9],9+
pgin	3	[0-20],(20-40],(40-60]
ppgin	3	[0-20],(20-40],(40-60]
pflt	3	[0-200],(200-400],400+
vflt	3	[0-200],(200-400],400+
runqsz	3	[0-2],(2-4],4+
runocc	5	20,40,60,80,100
freemem	4	[0-2500],(2500-5000],(5000-7500],(5000-7501]
freeswap	4	[0-900],(900-1200],(1200-1500],(1500+]
usr	9	[55-60],(60-65],(65-70],(70-75],(75-80],(80-85],(85-90],(90-95],(95-100]

Table 7.13 Data Encoding — Computer Active Database

Network Architecture and Training Parameters:

Number of Input Nodes : 85
Number of Hidden Nodes : 60
Number of Output Nodes : 9
Error Tolerance : 0.001
Learning Rate : 0.4
MAX Cycles : 10000
Penalty Factor HI : 0.03
Penalty Factor OH : 0.01
Number of Training Patterns : 2001

Table 7.14 Network Architecture — Computer Active Database

Training and Pruning:

TEST ID :	CompActiv2
Number of Training Patterns :	2001
Percentage of Patterns filtered :	16%
Number of cycles of Training :	10,000
Accuracy Acieved :	96.41%
No. correctly recognized :	1702
No. pruned : (IH | HO)	963/5100 | 293/600
Nodes pruned :	None

Table 7.15 Training Results — Computer Active Database

Summary of Extraction Tests:

$$\text{Cluster radius bound, } |r_c| \leq \frac{\ln\left(\frac{1}{\rho} - 1\right)}{\sum v^*}$$

$\sum v^*$ calculated to be $\cong 15$

Sigmoid scaling factor, $a = 0.01$

Error Tolerance, $\rho = 0.01$

Therefore, The bound on the cluster radius was found to be $\cong 0.30$

No. of Patterns Extracted

Cluster Radius	*Frequency*	*Hidden Layer Activation Level %*	*Rules Extracted*
0.2	30%	100	28
0.2	30%	96	60
0.2	30%	95	89
0.1	30%	100	20
0.1	30%	96	28
0.1	30%	95	53
0.05	30%	100	0
0.05	30%	96	20
0.05	30%	95	20

Table 7.16 No. of Rules Extracted — Computer Active Database

Knowledge Discovery:

Exıstıng Rules

(Cluster radıus=0 2, frequency=30%, Pass=96%)

```
RULE 0 :
IF    lread between 0 and 30
      lwrite between 0 and 30
      scall between 3k and 4k
      sread between 0 and 300
      swrite between 0 and 300
      fork between 0 and 3
      exec between 0 and 3
      rchar is 900+
      wchar is 900+
      pgout between 0 and 20
      ppgout between 0 and 20
      pgfree between <100 and
      pgscan between <100 and
      atch between 0 and 3
      pgin between 0 and 20
      ppgin between 0 and 20
      pflt between 0 and 200
      vflt between 0 and 200
      runqsz between 2 and 4
      runocc between 100 and
      freemem between 2500 and 5000
      freeswap is 1500+
THEN  usr between 90 and 95

RULE 1 :
IF    lread between 0 and 30
      lwrite between 0 and 30
      scall between 2k and 3k
      sread between 0 and 300
      swrıte between 0 and 300
      fork between 0 and 3
      exec between 0 and 3
      rchar is 900+
      wchar is 900+
      pgout between 0 and 20
      ppgout between 0 and 20
      pgfree between <100 and
      pgscan between <100 and
      atch between 0 and 3
      pgin between 0 and 20
      ppgin between 0 and 20
      pflt between 0 and 200
      vflt between 0 and 200
      runqsz between 2 and 4
      runocc between 100 and
```

```
       freemem between 0 and 2500
       freeswap is 1500+
THEN   usr between 90 and 95

RULE 2 :
IF     lread between 0 and 30
       lwrite between 0 and 30
       scall between 2k and 3k
       sread between 0 and 300
       swrite between 0 and 300
       fork between 0 and 3
       exec between 0 and 3
       rchar is 900+
       wchar is 900+
       pgout between 0 and 20
       ppgout between 0 and 20
       pgfree between <100 and
       pgscan between <100 and
       atch between 0 and 3
       pgin between 0 and 20
       ppgin between 0 and 20
       pflt between 0 and 200
       vflt between 0 and 200
       runqsz between 0 and 2
       runocc between 100 and
       freemem between 0 and 2500
       freeswap is 1500+
THEN   usr between 90 and 95

RULE 3 :
IF     lread between 0 and 30
       lwrite between 0 and 30
       scall between 1k and 2k
       sread between 0 and 300
       swrite between 0 and 300
       fork between 0 and 3
       exec between 0 and 3
       rchar is 900+
       wchar is 900+
       pgout between 0 and 20
       ppgout between 0 and 20
       pgfree between <100 and
       pgscan between <100 and
       atch between 0 and 3
       pgin between 0 and 20
       ppgin between 0 and 20
       pflt between 0 and 200
       vflt between 0 and 200
       runqsz between 0 and 2
       runocc between 100 and
       freemem between 5000 and 7501
       freeswap is 1500+
THEN   usr between 90 and 95
```

```
RULE 4 :
IF    lread between 0 and 30
      lwrite between 0 and 30
      scall between 1k and 2k
      sread between 0 and 300
      swrite between 0 and 300
      fork between 0 and 3
      exec between 0 and 3
      rchar is 900+
      wchar is 900+
      pgout between 0 and 20
      ppgout between 0 and 20
      pgfree between <100 and
      pgscan between <100 and
      atch between 0 and 3
      pgin between 0 and 20
      ppgin between 0 and 20
      pflt between 0 and 200
      vflt between 0 and 200
      runqsz between 0 and 2
      runocc between 100 and
      freemem between 0 and 2500
      freeswap is 1500+
THEN  usr between 90 and 95

RULE 5 :
IF    lread between 0 and 30
      lwrite between 0 and 30
      scall between <1k and
      sread between 0 and 300
      swrite between 0 and 300
      fork between 0 and 3
      exec between 0 and 3
      rchar is 900+
      wchar is 900+
      pgout between 0 and 20
      ppgout between 0 and 20
      pgfree between <100 and
      pgscan between <100 and
      atch between 0 and 3
      pgin between 0 and 20
      ppgin between 0 and 20
      pflt between 0 and 200
      vflt between 0 and 200
      runqsz between 2 and 4
      runocc between 100 and
      freemem between 2500 and 5000
      freeswap is 1500+ and
```

```
THEN  usr between 90 and 95
```

7.3 Determinants of Plasma Retinol and Beta-Carotene Levels

Description:

In this analysis, the relationship that exists between personal characteristics and dietary factors, and plasma concentrations of retinol, beta-carotene, and other carotenoids are discovered by the neural network and significant trends are extracted based on the existing correlations. The dataset is obtained from study conducted by Nierenberg et al. in Determinants of plasma levels of beta-carotene and retinol , American Journal of Epidemiology 1989, consists of 315 observed cases, to investigate the relationship between personal characteristics and dietary factors, and plasma concentrations of retinol, beta-carotene and other carotenoids. Study subjects (N = 315) were patients who had an elective surgical procedure during a three-year period to biopsy or remove a lesion of the lung, colon, breast, skin, ovary or uterus that was found to be non-cancerous.

Data description and encoding:

Variable	*Description*
AGE	Age (years)
SEX	Sex (1=Male, 2=Female).
SMOKSTAT	Smoking status (1=Never, 2=Former, 3=Current Smoker)
QUETELET	Quetelet (weight/(height^2))
VITUSE	Vitamin Use (1=Yes, fairly often, 2=Yes, not often, 3=No)
CALORIES	Number of calories consumed per day.
FAT	Grams of fat consumed per day.
FIBER	Grams of fiber consumed per day.
ALCOHOL	Number of alcoholic drinks consumed per week.
CHOLESTEROL	Cholesterol consumed (mg per day).
BETADIET	Dietary beta-carotene consumed (mcg per day).
RETDIET	Dietary retinol consumed (mcg per day)
BETAPLASMA	Plasma beta-carotene (ng/ml)
RETPLASMA	Plasma Retinol (ng/ml)

Table 7.17 Data Description — Plasma Concentrations

Variable	*No. of Nodes*	*Intervals*
AGE	6	[20-30],(30-40],(40-50],(50-60],(60-70],(70-80]
SEX	2	1=Male, 2=Female
SMOKSTAT	3	1=Never, 2=Former, 3=Current Smoker
QUETELET	6	[15-20],(20-25],(25-30],(30-35],(35-40],40+
VITUSE	3	1=Yes, fairly often, 2=Yes, not often, 3=No
CALORIES	5	<1000],(1000-1500],(1500-2000],(2000-2500],2500+
FAT	4	[0-50],(50-100],(100-150],(150-200]
FIBER	4	[0-5],(5-10],(10-15],15+
ALCOHOL	6	[0-2],(2-4],(4-6],(6-10],(10-20],20+
CHOLESTEROL	4	[0-250],(250-500],(500-750],750+
BETADIET	4	[0-2000],(2000-4000],(4000-6000],(6000-8000]
RETDIET	4	[0-500],(500-1000],(1000-1500],1500+
BETAPLASMA	6	[0-100],(100-200],(200-300],(300-400],(400-800],800+
RETPLASMA	6	[0-200],(200-400],(400-600],(600-800],(800-1000],(1000-1200]

Table 7.18 Data Encoding — Plasma Concentrations

Network Architecture and Training Parameters:

Number of Input Nodes : 51
Number of Hidden Nodes : 40
Number of Output Nodes : 12
Error Tolerance : 0.01
Learning Rate : 0.4
MAX Cycles : 10000
Penalty Factor HI : 0.03
Penalty Factor OH : 0.01

Table 7.19 Network Architecture — Plasma Concentrations

Training and Pruning:

TEST ID :	PlasRet1
Number of Training Patterns :	352
Percentage of Patterns filtered :	7%
Number of cycles of Training :	10,000
Accuracy Acieved :	100%
No. correctly recognized :	292
No. pruned : (IH | HO)	41 | 249
Nodes pruned :	None

Table 7.20 Training Results — Plasma Concentrations

Summary of Extraction Tests:

$$\text{Cl:uster radius bound, } |r_c| \leq \frac{\ln\left(\frac{1}{\rho} - 1\right)}{\sum v^*}$$

$\sum v^*$ calculated to be $\cong$ 1100

Sigmoid scaling factor, $a = 0.01$

Error Tolerance, $\rho = 0.01$

Therefore, The bound on the cluster radius was found to be $\cong$ 0.4

No. of rules extracted:

Existing Rules				
Radius	*0.2*	*0.2*	*0.1*	*0.1*
Frequency	*30*	*25*	*30*	*25*
100% Pass	0	0	0	0
95% Pass	7	11	0	0

Table 7.21 No. of Rules Extracted — Plasma Concentrations

Knowledge Discovery:

Existing Rules:

(Cluster radius=0.2, frequency=30%, Pass=95%)

```
RULE 0 :
IF    AGE between 50 and 60
      SEX is F
      SMOKSTAT is Current
      QUETELET between 15 and 20
      VITUSE is No
      CALORIES between 1500 and 2000
      FAT between 50 and 100
      FIBER between 10 and 15
      ALCOHOL between 0 and 2
      CHOLESTEROL between 0 and 250
      BETADIET between 2000 and 4000
      RETDIET between 0 and 500
THEN  BETAPLASMA between 0 and 100
```

```
       RETPLASMA between 400 and 600


RULE 1 :
IF     AGE between 50 and 60
       SEX is F
       SMOKSTAT is Former
       QUETELET between 25 and 30
       VITUSE is No
       CALORIES between 1500 and 2000
       FAT between 50 and 100
       FIBER is 15+
       ALCOHOL between 10 and 20
       CHOLESTEROL between 0 and 250
       BETADIET between 2000 and 4000
       RETDIET is 1500+
THEN   BETAPLASMA between 100 and 200
       RETPLASMA between 600 and 800

RULE 2 :
IF     AGE between 50 and 60
       SEX is F
       SMOKSTAT is Never
       QUETELET between 25 and 30
       VITUSE is Yes:Often
       CALORIES between 1500 and 2000
       FAT between 50 and 100
       FIBER between 10 and 15
       ALCOHOL between 0 and 2
       CHOLESTEROL between 0 and 250
       BETADIET between 0 and 2000
       RETDIET is 1500+
THEN   BETAPLASMA between 0 and 100
       RETPLASMA between 800 and 1000

RULE 3 :
IF     AGE between 40 and 50
       SEX is F
       SMOKSTAT is Never
       QUETELET between 20 and 25
       VITUSE is Yes:Often
       CALORIES between 1500 and 2000
       FAT between 50 and 100
       FIBER between 10 and 15
       ALCOHOL between 0 and 2
       CHOLESTEROL between 0 and 250
       BETADIET between 4000 and 6000
       RETDIET between 0 and 500
THEN   BETAPLASMA between 100 and 200
       RETPLASMA between 400 and 600
```

```
RULE 4 :
IF    AGE between 30 and 40
      SEX is F
      SMOKSTAT is Current
      QUETELET between 20 and 25
      VITUSE is No
      CALORIES between 1000 and 1500
      FAT between 50 and 100
      FIBER between 5 and 10
      ALCOHOL between 0 and 2
      CHOLESTEROL between 0 and 250
      BETADIET between 0 and 2000
      RETDIET between 0 and 500
THEN  BETAPLASMA between 100 and 200
      RETPLASMA between 400 and 600

RULE 5 :
IF    AGE between 30 and 40
      SEX is F
      SMOKSTAT is Never
      QUETELET between 35 and 40
      VITUSE is Yes:NotOften
      CALORIES between 1500 and 2000
      FAT between 0 and 50
      FIBER between 10 and 15
      ALCOHOL between 0 and 2
      CHOLESTEROL between 0 and 250
      BETADIET between 2000 and 4000
      RETDIET between 0 and 500
THEN  BETAPLASMA between 0 and 100
      RETPLASMA between 600 and 800

RULE 6 :
IF    AGE between 30 and 40
      SEX is F
      SMOKSTAT is Never
      QUETELET between 20 and 25
      VITUSE is Yes:NotOften
      CALORIES between 2000 and 2500
      FAT between 50 and 100
      FIBER is 15+
      ALCOHOL between 0 and 2
      CHOLESTEROL between 0 and 250
      BETADIET between 0 and 2000
      RETDIET between 1000 and 1500
THEN  BETAPLASMA between 200 and 300
      RETPLASMA between 400 and 600
```

7.4 Body-Fat

Description

This dataset includes estimates of the percentage of body fat determined by weighing underwater and various body circumference measurements for 252 men. This dataset can be used to illustrate multiple regression techniques. Accurate measurement of body fat is inconvenient or expensive and it is desirable to have easy methods of estimating body fat that are more convenient and less expensive.

In this analysis, we try to find predominant factors, which contribute to body fat.

Data description and encoding

Variable	*Description*
Density	Density determined from underwater weighing
Age	Age (years)
Weight	Weight (lbs)
Height	Height (inches)
Neck	Neck circumference (cm)
Chest	Chest circumference (cm)
Abdomen	Abdomen 2 circumference (cm)
Hip	Hip circumference (cm)
Thigh	Thigh circumference (cm)
Knee	Knee circumference (cm)
Ankle	Ankle circumference (cm)
Biceps	Biceps (extended) circumference (cm)
Forearm	Forearm circumference (cm)
Wrist	Wrist circumference (cm)
PercentBodyFat	Percent body fat from Siri's (1956) equation

Table 7.22 Data Description — Body Fat Percentage

Variable	*No. of Nodes*	*Intervals*
Density	5	[0.900-1.0200],(1.020-1.040],(1.040-1.060],(1.060-1.080],(1.080-1.20]
Age	5	[20-30],(30-40],(40-50],(50-60],(60-82]
Weight	6	[117-130],(130-150],(150-170],(170-190],(190-210],210+
Height	6	[27-66],(66-69],(69-71],(71-73],(73-75],(75-78]
Neck	3	[30-38],(38-46],(46-52]
Chest	3	[79-95],(95-110],(110-140]
Ab	3	[69-85],(85-100],(100-115]
Hip	3	[80-95],(95-110],(110-147]
Thigh	3	[47-65],(65-80],(80-90]
Knee	3	[32-39],(39-45],(45-50]
Ankle	3	[18-22],(22-26],(26-30]
Biceps	3	[24-32],(32-38],(38-46]
Forearm	3	[20-25],(25-30],(30-35]
Wrist	3	[15-17],(17-19],(19-22]
PercentBodyFat	7	[0-3],(3-7],(7-12],(12-18],(18-22],(22-27],(27-50]

Table 7.23 Data Encoding — Body Fat Percentage

Network Architecture and Training Parameters

Number of Input Nodes : 52
Number of Hidden Nodes : 60
Number of Output Nodes : 7
Error Tolerance : 0.001
Learning Rate : 0.4
MAX Cycles : 10000
Penalty Factor HI : 0.03
Penalty Factor OH : 0.01

Table 7.24 Network Architecture — Body Fat Percentage

Training and Pruning

TEST ID :	BodyFat
Number of Training Patterns :	252
Percentage of Patterns filtered :	2%
Number of cycles of Training :	10,000
Accuracy Acieved :	100%
No. correctly recognized :	248
No. pruned : (IH | HO)	369 | 168
Nodes pruned :	None

Table 7.25 Training Results — Body Fat Percentage

Summary of Extraction Tests

$$\text{Cluster radius bound, } |r_c| \leq \frac{\ln\left(\frac{1}{\rho} - 1\right)}{\sum v^*}$$

$\sum v^*$ calculated to be ≅ 1200

Sigmoid scaling factor, $a = 0.01$

Error Tolerance, $\rho = 0.01$

Therefore, The bound on the cluster radius was found to be ≅ 0.38

No. of rules extracted:

Existing Rules						
Radius	*0.2*	*0.2*	*0.2*	*0.3*	*0.3*	*0.3*
Frequency	*30*	*25*	*20*	*30*	*25*	*20*
100% Pass	0	0	0	0	0	1
95% Pass	1	1	4	1	9	25

Table 7.26 No. of Rules Extracted — Body Fat Percentage

Knowledge Discovery

Existing Rules:

(Cluster radius=0.2, frequency=20%, Pass=95%)

```
RULE 0 :
IF    Density between 1.040 and 1.060
      Age between 60 and 82
      Weight between 150 and 170
      Height between 69 and 71
      Neck between 30 and 38
      Chest between 95 and 110
      Abdomen between 85 and 100
      Hip between 95 and 110
      Thigh between 47 and 65
      Knee between 32 and 39
      Ankle between 18 and 22
      Biceps between 24 and 32
      Forearm between 25 and 30
      Wrist between 17 and 19
THEN  PercentBodyFat between 18 and 22
```

```
RULE 1 :
IF    Density between 1.040 and 1.060
      Age between 60 and 82
      Weight between 190 and 210
      Height between 71 and 73
      Neck between 38 and 46
      Chest between 95 and 110
      Abdomen between 85 and 100
      Hip between 95 and 110
      Thigh between 47 and 65
      Knee between 32 and 39
      Ankle between 22 and 26
      Biceps between 32 and 38
      Forearm between 25 and 30
      Wrist between 17 and 19
THEN  PercentBodyFat between 18 and 22

RULE 2 :
IF    Density between 1.040 and 1.060
      Age between 30 and 40
      Weight between 150 and 170
      Height between 69 and 71
      Neck between 30 and 38
      Chest between 95 and 110
      Abdomen between 85 and 100
      Hip between 95 and 110
      Thigh between 47 and 65
      Knee between 32 and 39
      Ankle between 22 and 26
      Biceps between 32 and 38
      Forearm between 25 and 30
      Wrist between 17 and 19
THEN  PercentBodyFat between 18 and 22

RULE 3 :
IF    Density between 1.040 and 1.060
      Age between 40 and 50
      Weight between 150 and 170
      Height between 69 and 71
      Neck between 30 and 38
      Chest between 95 and 110
      Abdomen between 85 and 100
      Hip between 95 and 110
      Thigh between 47 and 65
      Knee between 32 and 39
      Ankle between 22 and 26
      Biceps between 32 and 38
      Forearm between 25 and 30
      Wrist between 17 and 19
THEN  PercentBodyFat between 22 and 27
```

7.5 Pollution

Description

In this dataset, analysis is performed to find correlations that tend to connect physical and demographic environmental factors to mortality rates. The source of this data is McDonald, G.C. and Schwing, R.C. (1973) 'Instabilities of regression estimates relating air pollution to mortality', Technometrics, vol.15. This dataset consists of 60 patterns.

Data description and encoding

Variable	*Description*
PREC	Average annual precipitation in inches
JANT	Average January temperature in degrees F
JULT	Same for July
OVR65	% of 1960 SMSA population aged 65 or older
POPN	Average household size
EDUC	Median school years completed by those over 22
HOUS	% of housing units which are sound & with all facilities
DENS	Population per sq. mile in urbanized areas, 1960
NONW	% non-white population in urbanized areas, 1960
WWDRK	% employed in white collar occupations
POOR	% of families with income < $3000
HC	Relative hydrocarbon pollution potential
NOX	Same for nitric oxides
SO@	Same for sulphur dioxide
HUMID	Annual average % relative humidity at 1pm
MORT	Total age-adjusted mortality rate per 100,000

Table 7.27 Data Description — Pollution

Variable	*No. of Nodes*	*Intervals*
PREC	3	[0-30],(30-40],(40-50]
JANT	4	[0-20],(20-40],(40-50],(50-60]
JULT	4	[0-60],(60-70],(70-80],80+
OVR65	5	[<8],(8-9],(9-10],(10-11],11+
POPN	4	[2.5-3.0],(3.0-3.2],(3.2-3.4],(3.4-3.6]
EDUC	4	[7-10],(10-11],(11-12],12+
HOUS	3	[50-70],(70-80],(80-90]
DENS	5	[0-2000],(200-4000],(4k-6k],(6k-8k],8k+
NONW	4	[0-10],(10-20],(20-30],(30-40]
WWDRK	4	[20-30],(30-40],(40-50],50+
POOR	3	[0-10],(10-20],(20-30]
HC	4	[0-20],(20-40],(40-100],100+
NOX	4	[0-20],(20-40],(40-100],100+
SO@	4	[0-40],(40-60],(60-100],100+
HUMID	3	[50-55],(55-60],(60-65]

MORT	5	[850-900],(900-950],(950-1000],(1000-1050],(1050-1150]

Table 7.28 Data Encoding — Pollution

Network Architecture and Training Parameters

Number of Input Nodes : 58
Number of Hidden Nodes : 40
Number of Output Nodes : 5
Error Tolerance : 0.01
Learning Rate : 0.4
MAX Cycles : 10000
Penalty Factor HI : 0.03
Penalty Factor OH : 0.01

Table 7.29 Network Architecture— Pollution

Training and Pruning

TEST ID :	Poll
Number of Training Patterns :	60
Percentage of Patterns filtered :	3%
Number of cycles of Training :	10,000
Accuracy Acieved :	100%
No. correctly recognized :	58
No. pruned : (IH | HO)	363 | 85
Nodes pruned :	7,36

Table 7.30 Training Results — Pollution

Summary of Extraction Tests

$$\text{Cluster radius bound, } |r_c| \leq \frac{\ln\left(\frac{1}{\rho} - 1\right)}{\sum v^*}$$

$\sum v^*$ calculated to be ≅ 850

Sigmoid scaling factor, $a = 0.01$

Error Tolerance, $\rho = 0.01$

Therefore, The bound on the cluster radius was found to be ≅ 0.54

Existing Rules						
Radius	*0.2*	*0.2*	*0.2*	*0.3*	*0.3*	*0.3*
Frequency	*30*	*25*	*20*	*30*	*25*	*20*
100% Pass	0	0	0	0	0	0
95% Pass	0	0	1	0	0	1

Table 7.31 No. of Rules Extracted — Pollution

Knowledge Discovery

Existing Rules:

(Cluster radius=0.2, frequency=20%, Pass=95%)

```
RULE 0 :
IF     PREC between 40 and 50
       JANT between 20 and 40
       JULT between 70 and 80
       OVR65 between 10 and 11
       POPN between 3.2 and 3.4
       EDUC between 7 and 10
       HOUS between 80 and 90+
       DENS between 4k and 6k
       NONW between 0 and 10
       WWDRK between 30 and 40
       POOR between 10 and 20
       HC between 0 and 20
       NOX between 0 and 20
       SO@ between 0 and 40
       HUMID between 50 and 55
THEN   MORT between 950 and 1000
```

CHAPTER 8. CONCLUSION

8.0 Overview

A framework for knowledge discovery / data mining using neural networks has been formulated and demonstrated with applications in this work A summary of the work is presented in this chapter and conclusions are drawn Section 8 1 discusses our process of knowledge discovery using neural networks, its significance and benefits Section 8 2 discusses the applications we have used to support the process

8.1 Process for Knowledge Discovery Using Neural Networks

The key accomplishments of the process are as follows

a A sound framework for knowledge discovery of existing trends in datasets of patterns has been formulated

b The process allows for prediction of trends based on the knowledge acquired from the dataset

c The process possesses flexibility capabilities through control parameters

1 Frequency — Percentage of the dataset that must support the discovered trends

2 Radius — Degree of accuracy and generalization allowed for the selection of trends

3 Activation Level — Percentage of internal activity required to support the discovered trends

d The process is able to provide generalized output through the adaptive clustering process which produces representative values for regions of activations

e A simple process for extraction of rules from neural networks is defined

The process can be used for knowledge discovery of various types of pattern datasets A wide range of applications such as financial, medical, security, and socioeconomics can benefit from this process

The process is briefly summarized as follows

Step 1 Encoding the patterns in the dataset into appropriate Binary Patterns

Step 2 Training the Neural Network using the encoded input-output pairs of patterns, inconsistent mappings are filtered out

Step 3 Pruning the neural network and Re-Training

Step 4 Adaptive Clustering the Hidden Unit Activation Values to create regions of activations having frequency counts and representative values

Step 5 Discovering existing and predicted trends by extraction of rules

The process is sound due to the following reasons

a The accuracy of the trained network reflects the accuracy of the model acquired from the dataset

b The filtering process removes inconsistent mappings which do not display any dominant trend

c The insignificant attributes are identified and removed by pruning the unnecessary input and output neurons

d The bound for the cluster radius ensures the soundness of the extracted trends In addition, by using representative values for the activations, we obtain generalized trends for close patterns

e The patterns must satisfy rigorous vigilance tests on frequency, radius, and activation level to be selected as a dominant trend

8.2 Applications and Analysis

We have shown the applicability and robustness of the process by applying the process to various real world datasets The process was used to discover trends in datasets containing various types of data such as demographic_crime, dietary factors_Plasma Retinol and Beta-Carotene concentrations, system measurements_CPU usage, body measurements_body fat percentage, pollution_mortality In addition, the process was able to predict trends from the demographics_crime dataset

In Discovering Crime Trends Across Cities in the United States (Chapter 7), we present existing and predicted trends of crimes across three categories of city population The existing trends represent the dominating demographic factors that lead to various types of crimes which exist in the data collected across 6100 US cities The predicted trends demonstrate the generalization capability of the neural network based on the knowledge learned from the existing patterns On a broad view, we can see that there are low rates of murder, rape, and robberies for small and medium cities, whereas there are higher rates of these crimes in large cities The auto-thefts are low in small towns, moderate in medium towns, and high in large cities Further relationship between other demographic factors and occurrences of these crimes are shown in the individual trends These trends dominate the data set Comments made by experts in the criminal justice field show that the existing and predicted trends are accurate

In Computer Active Database (Chapter 7), we extract existing trends for CPU usage Based on the discovered trends, we can see that the CPU usage for 90-95% dominate the dataset The correlation between the CPU usage of 90-95% and various system measures can be seen in all trends

Similarly in other applications, we can observe the dominating characteristics of patterns in the respective datasets These applications show the usefulness, flexibility, robustness, and simplicity of the knowledge discovery process

In this work, we have demonstrated the usefulness of the process for a few applications There are many applications in various fields where this process can be applied to extract significant trends and predict generalized trends Applications can vary widely across different fields such as medical, military, security, operations, business and financial An example of a financial application would be to extract trends that reflect the demographic and personal characteristics of individuals to their usage of credit cards for different activities

The knowledge discovery process does not impose restrictions on the complexity of the network architecture Hence, the neural network can have several hidden layers This enables the neural network to model highly non-linear applications more accurately

APPENDIX A. NEURAL NETWORK SOURCE CODE

myNetwork.h

```
#include <fstream>
#include <string>

using namespace std;

//function to round real values between 0 and 1 to 0 and 1
double myRound(double);
//function to prune the network
void prune(double [50][100],int [50][100],double [100][40],int
[100][40],int,int,int,double);
//funtion to prune a specific hidden node
void pruneHiddenNode(int,double [50][100],int [50][100],double [100][40],int
[100][40],int,int,int,double);
//functions to calcuate and update the node existence after pruning
void calcNodeExistence(int [],int [],int [],int [50][100],int [100][40],int,int,int);
void correctMatrices(int [],int [],int [],int [50][100],int [100][40],int,int,int);


class myNetwork {
public
        myNetwork(){},
        myNetwork(double,double,int,int,int,char*),
        myNetwork(char*,char*,char*),

private

        double input[100],              //input layer
        double hiddenNodeSum[100],      //hidden layer
        double outputNodeSum[40],       //output layer

        //to keep track of existing nodes after pruning
        int extInput[100],
        int extHidden[100],
        int extOutput[40],

        //to hold the error at respective layers
        double errorOutput[40],
        double errorHidden[100];
        double errorStorage[20],
        double errorSum,

        //to hold threshold values for hidden and output layer
        double hiddenThreshold[100],
        double outputThreshold[40],

        //weights of connections
        double weightInput2Hidden[100][100],
        double weightHidden2Output[100][40],

        //to keep track of existing connections
        int extInput2Hidden[100][100],
        int extHidden2Output[100][40],

        //other parameters — alpha is learning rate
        int negativeDeterminer,
        int interruptTraining,
        float errorThreshold,
        float alpha,

        double temp;

        double ScaleFactor,
        double MomentumFactor,
```

```
        //number of neurons on respectıve layers
        ınt numberOfInputNodes,
        ınt numberOfHıddenNodes,
        ınt numberOfOutputNodes,

        //sıze of the traınıng set
        ınt numberOfTraınıngSet,


        //mean squared error of the network
        double MeanSquareError,

        //testID of the network
        char* netID,

        //Centroıd actıvatıon layer
        CentroıdActıvatıonLayer* CAL,


public :

        //Network ınıtıalızatıon methods
        voıd loadNetwork(char*,char*,char*),
        voıd saveNetworkForLoadback(char*,char*,char*),
        voıd newNetwork(),

        //network traınıng methods
        voıd penaltyTraınıng(float[][100],float[][40],ınt,ınt,double,double),
        voıd fılteredtraınNetworkPenalty(float[][100],float[][40],ınt&,ınt,double,double);

        //network recall methods
        double recall(float[][100],float[][40],ınt&,char*),
        double recallCorrect(float[][100],float[][40],ınt&,char*),

        //clusterıng methods
        void myNetwork: computeConfCluster(double),
        voıd  addActıvatıonLayer(double,char*),

        //actıvatıon recall methods
        double GActıvatıonRecall(float[][100],float[][40],ınt&,ınt,char*),
        double myNetwork  GActıvatıonRecallPercentage(float [][100], float [][40],ınt& ,
ınt ,char* ,ınt ),
        double GActıvatıonRecallGeneralızed(short[][100],ınt&,ınt,char*,ınt),

        //fılterıng the traınıng set to remove ınconsıstent patterns
        ınt fılterPatSet(double,ınt,float[][100],float[][40],double[]),

        //penalty update and prunıng methods
        double penDerHıdInp(double,double,double),
        double penDerOutHıd(double,double,double),
        double penDerforWeıght(double),
        voıd nullıfyWeıghtsBelowLevel(double),
        voıd netprune(),
        voıd netpruneHıdden(ınt),
        voıd fıxNodes(),


},
```

netFunc.cpp

```
//network utility functions

#include "cluster h"
#include "clusterspace1dim h"
#include "CentroidActivationLayer h"
#include "myNetwork h"
#include <vector>
#include <fstream>
#include <iostream>
#include <stdlib h>
#include <stdio h>
#include <math h>
#include <string>

using namespace std,

//constructors
myNetwork::myNetwork(double eT, double a, int nI, int nH, int nO,char* nid) {

        errorThreshold=eT,
        alpha=a,

        ScaleFactor=10,

        numberOfInputNodes=nI,
        numberOfHiddenNodes=nH,
        numberOfOutputNodes=nO,

        netID=nid,

}

myNetwork::myNetwork(char* weightf, char* existf, char* thresholdf) {
        loadNetwork(weightf,existf,thresholdf),
}



// building a new network using architecture specifications
void myNetwork::newNetwork() {

// Input Array initialized to Zero
                for ( int i = 0, i < numberOfInputNodes, i++)
                {
                        input[i] = 0,
                        extInput[i]=1,
                }


// Random Weights from INPUT to HIDDEN (-1 and +1)
        for ( int i = 0; i < numberOfHiddenNodes, i++)
        {
                for ( int j = 0; j < numberOfInputNodes, j++)
                {
                        negativeDeterminer = rand() % 2,
                        weightInput2Hidden[j][i] = (double)(negativeDeterminer -
1 0*rand()/(RAND_MAX+1 0)),
                }
        }

// Random Weights from HIDDEN to OUTPUT (-1 to +1)
        for ( int i = 0, i < numberOfOutputNodes, i++ )
        {
                for ( int j = 0, j < numberOfHiddenNodes, j++ )
                {
                        negativeDeterminer = rand() % 2,
                        weightHidden2Output[j][i] = (double)(negativeDeterminer -
1 0*rand()/(RAND_MAX+1 0));
                }
        }

// Thresholds
        for ( int i = 0, i < numberOfHiddenNodes; i++)
        {
                hiddenThreshold[i] = 0,//(double)(1 0*rand()/(RAND_MAX+1 0)),
        }
```

```
        for ( int i = 0, i < numberOfOutputNodes, i++)
        {
                outputThreshold[i] = 0,//(double)(1 0*rand()/(RAND_MAX+1 0)),
        }


// SUMS initialized to Zero
        for ( int i = 0, i < numberOfHiddenNodes, i++ )
        {
                hiddenNodeSum[i] = 0,
                extHidden[i]=1,
        }

        for ( int i = 0, i < numberOfOutputNodes, i++ )
        {
                outputNodeSum[i] = 0,
                extOutput[i]=1,
        }

        for (int  i = 0, i < numberOfHiddenNodes, i++ )
        {
                errorHidden[i] = 0,
        }

        for ( int i = 0, i < numberOfOutputNodes, i++ )
        {
                errorOutput[i] = 0,
        }


//Initialize the existence matrix
        for ( int i = 0, i < numberOfHiddenNodes, i++)
        {
                for ( int j = 0; j < numberOfInputNodes, j++)
                {
                        extInput2Hidden[j][i] = 1,
                }
        }


        for ( int i = 0, i < numberOfOutputNodes, i++ )
        {
                for ( int j = 0; j < numberOfHiddenNodes, j++ )
                {
                        extHidden2Output[j][i] =1,
                }
        }

}


//loading a saved network
void myNetwork::loadNetwork(char *wfilename, char *efilename ,char* tfile) {

// reading the architecture from the file

        int tempFlush,
        std  ifstream fWLoad(wfilename), //Weights* file
        std  ifstream fELoad(efilename); //ext* file

        fWLoad>>numberOfInputNodes,
        fWLoad>>numberOfHiddenNodes,


        for ( int i = 0, i < numberOfInputNodes, i++)
        {
                for ( int j = 0, j < numberOfHiddenNodes, j++)
                {
                        fWLoad >> weightInput2Hidden[i][j],
                        fELoad >> extInput2Hidden[i][j],
                }
        }


        for ( int i = 0; i < numberOfHiddenNodes, i++ )
        {
                for ( int j = 0, j < numberOfOutputNodes, j++ )
                {
                        fWLoad >> weightHidden2Output[i][j],
```

```
                        fELoad >> extHidden2Output[i][j],
                }
        }

        for(int i=0,i<numberOfInputNodes,i++) {
                fELoad>>extInput[i],
                }

        for(int i=0,i<numberOfHiddenNodes,i++) {
                fELoad>>extHidden[i],
                }

        for(int i=0,i<numberOfOutputNodes,i++) {
                fELoad>>extOutput[i],
                }

        fWLoad close(),
        fELoad close();

// Input Array initialized to Zero
                for ( int i = 0, i < numberOfInputNodes, i++)
                {
                        input[i] = 0;
                }
// SUMS initialized to Zero
        for ( int i = 0, i < numberOfHiddenNodes, i++ )
        {
                hiddenNodeSum[i] = 0;
        }

        for ( int i = 0, i < numberOfOutputNodes, i++ )
        {
                outputNodeSum[i] = 0,
        }

        for ( int i = 0, i < numberOfHiddenNodes, i++ )
        {
                errorHidden[i] = 0,
        }

        for ( int i = 0, i < numberOfOutputNodes, i++ )
        {
                errorOutput[i] = 0,
        }

// Thresholds
        for ( int i = 0, i < numberOfHiddenNodes, i++)
        {
                hiddenThreshold[i] = 0,//(double)(1 0*rand()/(RAND_MAX+1.0)),
        }

        for ( int i = 0, i < numberOfOutputNodes, i++)
        {
                outputThreshold[i] = 0,//(double)(1 0*rand()/(RAND_MAX+1 0)),
        }


        cout<<"*Network Loaded and Initialized*  Architecture   "<<numberOfInputNodes<<"
"<<numberOfHiddenNodes<<" "<<numberOfOutputNodes<<endl,

}


//save a trained network to a file
void myNetwork::saveNetworkForLoadback(char* wFile, char* tFile, char* eFile) {
        //open weights file and save weights with sizes of R X C before each matrix
        std  ofstream foutsave(wFile), //Weights* file
        std  ofstream fesave(eFile), //ext* file

        foutsave<<numberOfInputNodes<<" "<<numberOfHiddenNodes<<endl,
        for ( int i = 0, i < numberOfInputNodes, i++)
        {
                for ( int j = 0, j < numberOfHiddenNodes, j++)
                {
                        foutsave << weightInput2Hidden[i][j] << " ",
                        fesave << extInput2Hidden[i][j] << " ",
                }
                foutsave<<endl,
                fesave<<endl,
```

```
        }

        foutsave<<numberOfHiddenNodes<<" "<<numberOfOutputNodes<<endl,
        for ( int i = 0, i < numberOfHiddenNodes, i++ )
        {
                for ( int j = 0, j < numberOfOutputNodes, j++ )
                {
                        foutsave << weightHidden2Output[i][j] << " ",
                        fesave << extHidden2Output[i][j] << " ",
                }
                foutsave<<endl,
                fesave<<endl;
        }


        fesave<<endl,

        for(int i=0,i<numberOfInputNodes,i++) {
                fesave<<extInput[i]<<" ",
                }
        fesave<<endl,

        for(int i=0,i<numberOfHiddenNodes,i++) {
                fesave<<extHidden[i]<<" ",
                }
        fesave<<endl,

        for(int i=0,i<numberOfOutputNodes,i++) {
                fesave<<extOutput[i]<<" ",
                }
        fesave<<endl,

        foutsave.close(),
        fesave close(),


        //open threshold file and save threshold values with sizes before each matrix
        std  ofstream foutTsave(tFile),

        foutTsave<<numberOfHiddenNodes<<endl,
        for(int i=0,i<numberOfHiddenNodes,i++) {
                foutTsave<<hiddenThreshold[i]<<" ",
                }
        foutTsave<<endl,

        foutTsave<<numberOfOutputNodes<<endl,
        for(int i=0,i<numberOfOutputNodes,i++) {
                foutTsave<<outputThreshold[i]<<" ",
                }
        foutTsave<<endl,

        foutTsave close(),

        cout<<"*Network Saved*"<<endl,
}


//pruning methods
void myNetwork::nullifyWeightsBelowLevel(double level) {

        for ( int i = 0, i < numberOfHiddenNodes, i++)
        {
                for ( int j = 0, j < numberOfInputNodes, j++)
                {
                        if(fabs(weightInput2Hidden[j][i])<level) {
                                weightInput2Hidden[j][i] = 0,
                                extInput2Hidden[j][i] = -1,
                                }
                }
        }


        for ( int i = 0, i < numberOfOutputNodes; i++ )
        {
                for ( int j = 0, j < numberOfHiddenNodes, j++ )
                {
                        if(fabs(weightHidden2Output[j][i])<level) {
                                weightHidden2Output[j][i] = 0,
                                extHidden2Output[j][i] = -1,
                                }
```

```
                }
        }

}


//pruning methods
void myNetwork::netprune() {
        prune(weightInput2Hidden,extInput2Hidden,weightHidden2Output,extHidden2Output,numb
erOfInputNodes,numberOfHiddenNodes,numberOfOutputNodes,0 49),
}

void myNetwork::netpruneHidden(int hid) {
        pruneHiddenNode(hid,weightInput2Hidden,extInput2Hidden,weightHidden2Output,extHidd
en2Output,numberOfInputNodes,numberOfHiddenNodes,numberOfOutputNodes,0 49),
}

void myNetwork::fixNodes() {
        calcNodeExistence(extInput,extHidden,extOutput,extInput2Hidden,extHidden2Output,nu
mberOfInputNodes,numberOfHiddenNodes,numberOfOutputNodes),
        correctMatrices(extInput,extHidden,extOutput,extInput2Hidden,extHidden2Output,numb
erOfInputNodes,numberOfHiddenNodes,numberOfOutputNodes),
}

//adding an centroid activation layer to the network
void myNetwork::addActivationLayer(double rad, char* actFile) {

        int numUnits=numberOfHiddenNodes,
        double radius=rad,

        //container for activation values - one for each node
        vector<double>* actValCont[100];

        //initialize the containers for each node
        for(int i=0,i<numUnits;i++) {
                actValCont[i]=new vector<double>(),
        }

        //read the file containing Hidden unit activation values, each line contains  activation for all nodes in 1 cycle
        //store the values in the respective containers
        ifstream readValues(actFile),
        double tempVal,
        int cont=0;

        while(true){
                readValues>>tempVal,
                if(readValues ios  eof()) {
                        break,
                        }
                else {
                        actValCont[0]->push_back(tempVal),

                        for(int node=1,node<numUnits,node++) {
                                readValues>>tempVal,
                                actValCont[node]->push_back(tempVal);
                                }

                        }
                }

        //create a new centroid activation layer and return the pointer back to it
        CAL=new CentroidActivationLayer(actValCont,numberOfHiddenNodes,rad),

}
```

APPENDIX B. NEURAL NETWORK TRAINING AND PRUNING PHASE SOURCE CODE

ApplTrainFilt.cpp

//main function for initializing the network and creating the workflow of training, filtering and pruning procedure

```
#include "cluster h"
#include "clusterspace1dim h"
#include "CentroidActivationLayer h"
#include "myNetwork h"
#include <fstream>
#include <stdlib h>
#include <iostream>
#include <stdio h>

using namespace std,

//initialize training buffer
int init_training_buffer(ifstream& ,int ,int , float [][100],float [][40]),

int main( int argc, char** argv ){

        //Reading Architecture
        //arg 1 = TestID
        char tempA[20]="Arch",
        char* File=strcat(tempA,argv[1]), //Arch*file

        //initialize architecture
        int numInp=-1,
        int numHidden=-1,
        ifstream archRead(File),

        archRead>>numInp,
        archRead>>numHidden,
        archRead>>numOut;

        archRead.close(),
        cout<<"*Architecture Initialized*  "<<numInp<<" "<<numHidden<<" "<<numOut<<endl;

        //reading Training Data Set
        //arg 2 = training file
        ifstream trainFile(argv[2]),

        float inpBuf[3100][100];
        float outBuf[3100][40],

        int numTSample=init_training_buffer(trainFile,numInp,numOut,inpBuf,outBuf),
        cout<<"*Traning buffer Initialized*"<<endl,
        trainFile.close(),


        //getting training parameters
        double learningRate,
        double decayHI,
        double decayOH,
        int cycles,
        int pruneCycles,

        cout<<"Enter Learning rate . ", cin>>learningRate, cout<<endl,
        cout<<"Enter Decay HI Factor   ", cin>>decayHI, cout<<endl,
        cout<<"Enter Decay OH Factor   "; cin>>decayOH, cout<<endl,
        cout<<"Enter Number of Cycles · ", cin>>cycles, cout<<endl,
        cout<<"Enter Number of Cycles after each pruning   ", cin>>pruneCycles,
cout<<endl,
```

```
//build network
myNetwork NetA(0 01,learningRate,numInp,numHidden,numOut,argv[5]),
NetA newNetwork(),
cout<<"*Network Built*"<<endl,


//display Info
cout<< endl,
cout<<"TEST ID   "<<argv[1]<<endl,
cout<<"Data File   "<<argv[2]<<endl,

int phase=0,
char tempphase[5],
int hid=0,

//filenames for Weights*, ext* and Threshold*
char tempw[20]="Weights",
char tempe[20]="ext",
char tempt[20]="Threshold",

char* wfile=strcat(tempw,argv[1]),
char* efile=strcat(tempe,argv[1]),
char* tfile=strcat(tempt,argv[1]),

cout<<"Number of Input Nodes   "<<numInp<<endl,
cout<<"Number of Hidden Nodes   "<<numHidden<<endl,
cout<<"Number of Output Nodes . "<<numOut<<endl,
cout<<"Error Tolerance    0 001"<<endl,
cout<<"Learning Rate   "<<learningRate<<endl,
cout<<"MAX Cycles   "<<cycles<<endl,
cout<<"Penalty Factor HI    "<<decayHI<<endl,
cout<<"Penalty Factor OH    "<<decayOH<<endl,
cout<<"Number of Training Patterns   "<<numTSample<<endl,


//start the training, filtering pruning workflow

while(true) {
        //initial training cycle
        if(phase>=1)
                cycles=pruneCycles,

//training the network with penalty and filtering
NetA filteredtrainNetworkPenalty(inpBuf,outBuf,numTSample,cycles,decayHI,decayOH),
        cout<<"*Training Complete* Phase "<<phase<<endl,


        cout<<"Network Calling .   "<<endl,
        double acu=NetA recall(inpBuf,outBuf,numTSample,argv[1]),
        cout<<"*Recall Complete*  Accuracy   "<<acu<<endl,

        //pruning the network, complete network at phase 0 and each hidden neuron thereafter
        cout<<"Pruning    "<<endl,
        if (phase==0) {
                NetA netprune(),
                NetA fixNodes();
                }
        else {
                NetA netpruneHidden(hid);
                NetA fixNodes();
                hid++,
         }

        acu=NetA recall(inpBuf,outBuf,numTSample,argv[1]),
        cout<<"*Recall Complete after pruning*  Accuracy   "<<acu<<endl,

        if (hid>=numHidden) {
//training the network without filtering after phase 0
NetA penaltyTraining(inpBuf,outBuf,numTSample,cycles,decayHI,decayOH),
                cout<<"*Training Complete* Phase "<<phase<<endl,
                cout<<"Final pruning . ."<<endl,
                NetA netprune(),
                NetA fixNodes(),
                double acu=NetA recall(inpBuf,outBuf,numTSample,argv[1]),
                cout<<"*Recall Complete after pruning*  Accuracy   "<<acu<<endl,

                break;
                }
```

```
            phase ++,
        }

            //save the network
            NetA saveNetworkForLoadback(wfile,tfile,efile),

}
```

penaltyfilttraining cpp

```
//method for training the network with penalty update every 2-5 cycles with filtering process
#include "cluster h"
#include "clusterspace1dim h"
#include "CentroidActivationLayer h"
#include "myNetwork h"
#include <fstream>
#include <iostream>
#include <stdlib h>
#include <math.h>

using namespace std,

void myNetwork::filteredtrainNetworkPenalty(float iBuffer[][100],float oBuffer[][40],
int& patSetSize, int cycles, double penaltyFactorOH, double penaltyFactorHI){

        int cycle=0,
        char tempfilename[20]="MSError";       //file for storing MSError
        ofstream errorSave(strcat(tempfilename,netID)),
        double ErrorAtCycle,

        //set number of cycles between each penalty update
        int penaltyCycle=5;
        double penUpdate=0,

        //cycles for filtering
        int cycle1=6999,cycle2=8999,

        //maximum number of cycles for training
        int MAXCYCLES=cycles,


        cout<<"Training Network (Filtered) Version3 0 With Penalty Update .   "<<endl;
        cout<<endl,

        if(patSetSize<=0) {
                cout<<"No training set"<<endl,
                errorSave close(),
                exit(1),
                }

        //array for holding  MSError values for all input patterns
        double mseArr[3100],
        for(int i=0,i<patSetSize,i++) {
                mseArr[i]=0,
                }


        //begin training cycles — each outer loop is 1 cycle
        while (true) {                  //Outer loop

        MeanSquareError=0;

                //pass each input through the network — each inner loop is I pattern
                for(int p=0,p<patSetSize,p++) {                 //inner loop

                        //initialize  input layer
                        for(int iDim=0;iDim<numberOfInputNodes,iDim++) {
                                input[iDim]=iBuffer[p][iDim],
                                }


                        // **** HIDDEN NODE SUMS ****
                                for (int i = 0; i < numberOfHiddenNodes, i++)
                                {
                                        hiddenNodeSum[i] = 0,
                                }
                                for (int i = 0; i < numberOfHiddenNodes, i++)
                                {
                                        for ( int j = 0, j < numberOfInputNodes, j++)
                                        {
                                                if(extInput2Hidden[j][i]'=-1)
                                                        hiddenNodeSum[i] += input[j] *
weightInput2Hidden[j][i],

                                        }
```

```
                                        hıddenNodeSum[ı] = 1 0 / (1 0 + exp(- 010 *
(hıddenNodeSum[ı] - hıddenThreshold[ı]))),
                                }

                                // **** OUTPUT NODE SUMS ****
                                for (ınt ı = 0, ı < numberOfOutputNodes, ı++)
                                {
                                        outputNodeSum[ı] = 0,
                                }
                                for ( ınt ı = 0, ı < numberOfOutputNodes; ı++)
                                {
                                        for ( ınt ȷ = 0, ȷ < numberOfHıddenNodes, ȷ++)
                                        {
                                                ıf(extHıdden2Output[ȷ][ı]!=-1)
                                                        outputNodeSum[ı] += hıddenNodeSum[ȷ]
* weıghtHıdden2Output[ȷ][ı],
                                        }
                                        outputNodeSum[ı] = 1 0 / (1 0 + exp(- 010 *
(outputNodeSum[ı] - outputThreshold[ı]))),
                                }


                        // **** ERROR CALCULATIONS ****

                        // Inıtıalıze Error to Zero
                                for ( ınt ı = 0, ı < numberOfHıddenNodes, ı++ )
                                {
                                        errorHıdden[ı] = 0,
                                }

                                for ( ınt ı = 0, ı < numberOfOutputNodes, ı++ )
                                {
                                        errorOutput[ı] = 0,
                                }


                // Error Calculatıon for Output Nodes

                ErrorAtCycle=0,
                //error for pattern p   sıgma k=1 to o {(dk-Ok)^2}
                for(ınt k=0, k<numberOfOutputNodes,k++) {
                        errorOutput[k]=oBuffer[p][k] - outputNodeSum[k]; //d(p,k)-o(p,k)
                        ErrorAtCycle=ErrorAtCycle+(errorOutput[k]*errorOutput[k]);
                        ıf(cycle==4999) {
                                mseArr[p]=ErrorAtCycle,
                                }
                        //delta(p,k)=err(p,k)*o(p,k)*(1-o(p,k))
                        errorOutput[k]=errorOutput[k]*outputNodeSum[k]*(1-
outputNodeSum[k]);
                        }

                //add the squared pattern error to the sum
                MeanSquareError+=ErrorAtCycle,

                                // Error Calculatıon for Hıdden Nodes
                                for ( ınt ı = 0, ı < numberOfHıddenNodes, ı++)
                                {
                                        temp = 0,
                                        for ( ınt ȷ = 0, ȷ < numberOfOutputNodes, ȷ++)
                                        {
                                                ıf(extHıdden2Output[ı][ȷ]!=-1)
                                                        temp += errorOutput[ȷ] *
weıghtHıdden2Output[ı][ȷ];
                                        }
                                        errorHıdden[ı] = hıddenNodeSum[ı] * (1 -
hıddenNodeSum[ı]) * temp, //delta(p,ȷ)
                                }


                        // **** WEIGHTS UPDATE ****

                        //      Weıght Change between Hıdden and Output
                                for ( ınt ı = 0, ı < numberOfOutputNodes, ı++ )
                                {
                                        for ( ınt ȷ = 0, ȷ < numberOfHıddenNodes, ȷ++ )
                                        {

                                                ıf(extInput2Hıdden[ȷ][ı]!=-1) {
```

```
                                                weıghtHıdden2Output[j][ı] += alpha *
hıddenNodeSum[j] * errorOutput[ı], //deltaW(o,h)=alpha*error(o)*s'(o)*ınput(o)
                                                ıf(cycle%penaltyCycle==0) {

        penUpdate=penaltyFactorOH*penDerforWeıght(weıghtHıdden2Output[j][ı]),
                                                        weıghtHıdden2Output[j][ı]-=penUpdate,
                                                        }
                                                }

                                        }
                                }


                        // Weıght Changıng between Input and Hıdden
                                for ( ınt ı = 0, ı < numberOfHıddenNodes, ı++ )
                                {
                                        for ( ınt j = 0, j < numberOfInputNodes, j++ )
                                        {
                                                ıf(extHıdden2Output[j][ı]!=-1) {
                                                weıghtInput2Hıdden[j][ı] += alpha * ınput[j]
* errorHıdden[ı],

                                                ıf(cycle%penaltyCycle==0) {

        penUpdate=penaltyFactorHI*penDerforWeıght(weıghtInput2Hıdden[j][ı]),
                                                        weıghtInput2Hıdden[j][ı]-=penUpdate;
                                                        }

                                                }
                                        }
                                }

                        //**** THRESHOLD CHANGE ****
                         //Threshold Change Output Layer
                                for(ınt ı = 0, ı < numberOfOutputNodes, ı++)
                                {
                                        outputThreshold[ı] += ( 1 * errorOutput[ı]),
                                }

                                for(ınt ı = 0, ı < numberOfOutputNodes, ı++)
                                {
                                        outputThreshold[ı] = 1 0 / ( 1.0 + exp(-
1*outputThreshold[ı])),
                                }

                        //Theshold Change Hıdden Layer
                                for(ınt ı=0; ı<numberOfHıddenNodes, ı++)
                                {
                                        hıddenThreshold[ı] += ( 1 * errorHıdden[ı]),
                                }

                                for(ınt ı=0, ı<numberOfHıddenNodes, ı++)
                                {
                                        hıddenThreshold[ı] = 1 0 / ( 1 0 + exp(-
1*errorHıdden[ı])),
                                }

                } //end ınner loop

                MeanSquareError=(MeanSquareError/patSetSıze),
                errorSave<<MeanSquareError<<endl,

                //fındıng ıf any errors have varıance wrt to the errors of all patterns
                //remove patterns wıth hıgh error
                ıf(cycle==cycle1) {
                double cV=MeanSquareError*2,
                cout<<endl<<"Cycle   "<<cycle<<"  Fınıdıng patterns wıth hıgh error eCycle
greater than eq · "<<cV<<endl,
                patSetSıze=fılterPatSet(cV,patSetSıze,ıBuffer,oBuffer,mseArr),
                }

                ıf(cycle==cycle2) {
                ıf(MeanSquareError>0 1) {
                        double cV=MeanSquareError*2,
                        cout<<endl<<"Cycle   "<<cycle<<"  Fınıdıng patterns wıth hıgh error
eCycle greater than eq   "<<cV<<endl,
                        patSetSıze=fılterPatSet(cV,patSetSıze,ıBuffer,oBuffer,mseArr),
                        }
                }
```

```
                if(cycle%1000==0)
                        cout<<"** Network Mean Error @ "<<cycle<<"    "<<MeanSquareError<<"
**"<<endl,
                cycle++,
                if((MeanSquareError < errorThreshold) || (cycle > MAXCYCLES)) {
                        cout<<endl<<"Training completed at cycle    " <<cycle<<endl,
                        break;
                        }


                } //end outer loop

                errorSave close(),


}               //end training

//filtering the dataset
int myNetwork::filterPatSet(double checkVal, int pSize, float iBuffer[][100], float
oBuffer[][40], double mseArr[]) {

                //check to see if the Error of the pattern is greater than desired value
                for(int z=0,z<pSize,z++) {
                        if(mseArr[z] > checkVal) {
                                cout<<"Removing Pattern "<<z<<endl,

                                //copy the pattern at the end of the array to the current location and reduce the array size
by 1
                                for(int l=0,l<numberOfInputNodes;l++) {
                                        iBuffer[z][l]=iBuffer[pSize-1][l],
                                        iBuffer[pSize-1][l]=0,
                                        }
                                for(int l=0,l<numberOfOutputNodes,l++) {
                                        oBuffer[z][l]=oBuffer[pSize-1][l],
                                        oBuffer[pSize-1][l]=0,
                                        }

                                pSize--,

                                }
                        }

                cout<<endl,

                alpha++,
                return pSize,
}
```

penaltyTraining.cpp

```
//method for training the network with penalty updates (no filtering)
#include "cluster h"
#include "clusterspace1dim h"
#include "CentroidActivationLayer h"
#include "myNetwork h"
#include <fstream>
#include <iostream>
#include <stdlib h>
#include <math h>


using namespace std,

void myNetwork::penaltyTraining(float iBuffer[][100],float oBuffer[][40], int patSetSize,
int cycles, double penaltyFactorOH, double penaltyFactorHI){

        int cycle=0,
        char tempfilename[20]="MSError",
        ofstream errorSave(strcat(tempfilename,netID)),
        double ErrorAtCycle,

        int MAXCYCLES=cycles,

        double penUpdate=0,

        cout<<endl,
        cout<<"Training Network (Non-Filtered) Version2 0 With Penalty Update     "<<endl;
        cout<<endl,

        if(patSetSize<=0) {
                cout<<"No training set"<<endl,
                errorSave close(),
                exit(1),
                }


        while (true) {                  //Outer loop

        MeanSquareError=0,

                for(int p=0,p<patSetSize,p++) {                 //inner loop

                        //initialize  input layer
                        for(int iDim=0,iDim<numberOfInputNodes,iDim++) {
                                input[iDim]=iBuffer[p][iDim],
                                }

                                // **** HIDDEN NODE SUMS ****
                                for (int i = 0, i < numberOfHiddenNodes, i++)
                                {
                                        hiddenNodeSum[i] = 0,
                                }
                                for (int i = 0; i < numberOfHiddenNodes, i++)
                                {
                                        for ( int j = 0, j < numberOfInputNodes, j++)
                                        {
                                                if(extInput2Hidden[j][i]'=-1)
                                                        hiddenNodeSum[i] += input[j] *
weightInput2Hidden[j][i],

                                        }
                                        hiddenNodeSum[i] = 1.0 / (1 0 + exp(- 010 *
(hiddenNodeSum[i] - hiddenThreshold[i]))),
                                }

                                // **** OUTPUT NODE SUMS ****
                                for (int i = 0; i < numberOfOutputNodes; i++)
                                {
                                        outputNodeSum[i] = 0;
                                }
                                for ( int i = 0, i < numberOfOutputNodes, i++)
                                {
                                        for ( int j = 0, j < numberOfHiddenNodes, j++)
                                        {
                                                if(extHidden2Output[j][i]'=-1)
                                                        outputNodeSum[i] += hiddenNodeSum[j]
* weightHidden2Output[j][i],
```

```
                                        }
                                        outputNodeSum[i] = 1.0 / (1.0 + exp(- 0.10 *
(outputNodeSum[i] - outputThreshold[i]))),
                                }

                                // **** ERROR CALCULATIONS ****
                                // Initialise Error to Zero
                                for ( int i = 0, i < numberOfHiddenNodes, i++ )
                                {
                                        errorHidden[i] = 0,
                                }

                                for ( int i = 0, i < numberOfOutputNodes, i++ )
                                {
                                        errorOutput[i] = 0,
                                }

                        // Error Calculation for Output Nodes
                        ErrorAtCycle=0,
                        //error for pattern p   sigma k=1 to o {(dk-Ok)^2}
                        for(int k=0; k<numberOfOutputNodes,k++) {
                                errorOutput[k]=oBuffer[p][k] - outputNodeSum[k], //d(p,k)-o(p,k)
                                ErrorAtCycle=ErrorAtCycle+(errorOutput[k]*errorOutput[k]),
                                errorOutput[k]=errorOutput[k]*outputNodeSum[k]*(1-
outputNodeSum[k]), //delta(p,k)=err(p,k)*o(p,k)*(1-o(p,k))
                                }

                        //add the squared pattern error to the sum
                        MeanSquareError+=ErrorAtCycle,

                        // Error Calculation for Hidden Nodes
                                        for ( int i = 0, i < numberOfHiddenNodes, i++)
                                        {
                                                temp = 0,
                                                for ( int j = 0, j < numberOfOutputNodes, j++)
                                                {
                                                        if(extHidden2Output[i][j]!=-1)
                                                                temp += errorOutput[j] *
weightHidden2Output[i][j],
                                                }
                                                errorHidden[i] = hiddenNodeSum[i] * (1 -
hiddenNodeSum[i]) * temp, //delta(p,j)
                                        }

                                //**** WEIGHTS CHANGE ****
                                //      Weight Change between Hidden and Output
                                        for ( int i = 0, i < numberOfOutputNodes, i++ )
                                        {
                                                for ( int j = 0, j < numberOfHiddenNodes, j++ )
                                                {

                                                        if(extHidden2Output[j][i]!=-1) {

        penUpdate=penaltyFactorOH*penDerforWeight(weightHidden2Output[j][i]),
                                                        weightHidden2Output[j][i] += (alpha *
hiddenNodeSum[j] * errorOutput[i]), //deltaW(o,h)=alpha*error(o)*s'(o)*input(o)
                                                        weightHidden2Output[j][i]-=penUpdate;
                                                        }
                                                }
                                        }



                                // Weight Changing between Input and Hidden
                                        for ( int i = 0, i < numberOfHiddenNodes, i++ )
                                        {
                                                for ( int j = 0, j < numberOfInputNodes; j++ )
                                                {
                                                        if(extInput2Hidden[j][i]!=-1) {

        penUpdate=penaltyFactorHI*penDerforWeight(weightInput2Hidden[j][i]),
                                                        weightInput2Hidden[j][i] += alpha * input[j]
* errorHidden[i],
                                                        weightInput2Hidden[j][i]-=penUpdate;
                                                        }
                                                }
                                        }

                                //**** THRESHOLD CHANGE ****
```

```
                        //Threshold Change Output Layer
                                for(int i = 0, i < numberOfOutputNodes, i++)
                                {
                                        outputThreshold[i] += ( 1 * errorOutput[i]),
                                }

                                for(int i = 0, i < numberOfOutputNodes, i++)
                                {
                                        outputThreshold[i] = 1 0 / ( 1 0 + exp(-
1*outputThreshold[i])),
                                }


                         //Theshold Change Hidden Layer
                                for(int i=0, i<numberOfHiddenNodes, i++)
                                {
                                        hiddenThreshold[i] += ( 1 * errorHidden[i]),
                                }

                                for(int i=0; i<numberOfHiddenNodes, i++)
                                {
                                        hiddenThreshold[i] = 1 0 / ( 1 0 + exp(-
1*errorHidden[i]));
                                }


                }   // end inner loop

                MeanSquareError=(MeanSquareError/patSetSize),
                errorSave<<MeanSquareError<<endl,

                if(cycle%1000==0)
                        cout<<"** Network Mean Error @ "<<cycle<<" . "<<MeanSquareError<<"
**"<<endl,
                cycle++,

                if((MeanSquareError < errorThreshold) || (cycle > MAXCYCLES)) {
                        cout<<endl<<"Training completed at cycle   " <<cycle<<endl,
                        break,
                        }
                } //end outer loop

                errorSave close();
}
```

penaltyFunc cpp

```
//functions for calculating penalty values for connection weights based
#include "cluster h"
#include "clusterspace1dim h"
#include "CentroidActivationLayer h"
#include "myNetwork h"
#include <math h>

using namespace std,

double myNetwork::penDerHidInp(double ep1,double ep2,double beta) {
        double tempWFac1=0,
        double tempWFac2=0,

        for(int i=0,i<numberOfHiddenNodes,i++) {
                for(int j=0;j<numberOfInputNodes,j++) {
                        //w/(1+bw^2)^2
                        tempWFac1+=weightInput2Hidden[j][i] / pow(1 +
beta*pow(weightInput2Hidden[j][i],2),2),
                        //w
                        tempWFac2+=weightInput2Hidden[j][i],
                        }
                }
        return ((2*ep1*beta*tempWFac1) + (2*ep2*tempWFac2)),
}

double myNetwork::penDerOutHid(double ep1,double ep2,double beta) {
        double tempWFac1=0,
        double tempWFac2=0,

        for(int i=0,i<numberOfOutputNodes,i++) {
                for(int j=0,j<numberOfHiddenNodes;j++) {
                        //w/(1+bw^2)^2
                        tempWFac1+=weightHidden2Output[j][i] / pow(1 +
beta*pow(weightHidden2Output[j][i],2),2),
                        //w
                        tempWFac2+=weightHidden2Output[j][i],
                        }
                }
        return ((2*ep1*beta*tempWFac1) + (2*ep2*tempWFac2)),
}

double myNetwork  penDerforWeight(double weight) {
        double tempWFac1=0,
        double tempWFac2=0,

        double ep1=0 1,
        double ep2=0 00001,
        double beta=10,
                        //w/(1+bw^2)^2
                        tempWFac1=weight / pow(1 + beta*pow(weight,2),2),
                        //w
                        tempWFac2=weight,
        return ((2*ep1*beta*tempWFac1) + (2*ep2*tempWFac2));
}
```

pruneMatrıx cpp

```
//methods for pruning the connections and removing unnecessary nodes
#include<fstream>
#include <iostream>
#include<math h>

using namespace std,

void prune(double wIH[100][100], int extIH[100][100], double wHO[100][40], int
extHO[100][40],int ni,int nh,int no,double n2) {
        double vMax[100];

        //initialize the vMax array to zero
        for(int i=0;i<100,i++) {
                vMax[i]=0;
                }

        //set vMax[h] to the max value of the connection Who connecting hidden unit h to output unit o for all o
        cout<<"Vmax : "<<" ",
        for(int i=0;i<nh;i++) {
                for(int j=0;j<no,j++){
                        if(fabs(wHO[i][j])>vMax[i]) {
                                vMax[i]=fabs(wHO[i][j]),
                                }
                        }
                        cout<<vMax[i]<<" ",
                }
                cout<<endl,

        int IHpruned=0,

        //for all the elements of wIH, prune the weights if the |WihXvMax[h]|<4n2, but setting the value of wIH[i][h] to 0 and
extIH[i][h] to -1
        for(int i=0,i<nh,i++) {
                for(int j=0,j<ni,j++){
                        if((fabs(wIH[j][i])*vMax[i])<(4*n2)) {
                                wIH[j][i]=0,
                                extIH[j][i]=-1,
                                IHpruned++,
                                }
                        }
                }

        cout<<"IH pruned   "<<IHpruned<<endl,


        int HOpruned=0,
        //for all elemets of wOH prune all the weights which are < 4n2
        for(int i=0,i<nh,i++) {
                for(int j=0,j<no,j++){
                        if(fabs(wHO[i][j])<(4*n2)) {
                                wHO[i][j]=0,
                                extHO[i][j]=-1,
                                HOpruned++,
                                }
                        }
                }
        cout<<"OH pruned   "<<HOpruned<<endl,

}

void pruneHiddenNode(int h,double wIH[100][100], int extIH[100][100], double
wHO[100][40], int extHO[100][40],int ni,int nh,int no,double n2) {

        if(h>(nh-1)) {
                cout<<"Error pruning Hidden unit "<<h<<" Segmentation"<<endl,
                return,
                }

        double vMax=0,


        cout<<"Hid unit "<<h,

        //set vMax[h] to the max value of the connection Who connecting hidden unit h to output unit o for all o
        cout<<" Vmax   "<<" ",
                for(int j=0,j<no,j++){
                        if(fabs(wHO[h][j])>vMax) {
```

```
                                vMax=fabs(wHO[h][j]),
                                }
                        }
                        //cout<<vMax<<" ",


        int IHpruned=0,

        //for all the elements of wIH, prune the weights if the |WithXvMax[h]|<4n2, but setting the value of wIH[i][h] to 0 and
extIH[i][h] to -1
                for(int j=0,j<ni,j++){
                        if((fabs(wIH[j][h])*vMax)<(4*n2)) {
                                wIH[j][h]=0,
                                extIH[j][h]=-1;
                                IHpruned++,
                                }
                        }

        cout<<"IH pruned   "<<IHpruned,


        int HOpruned=0,

        //for all elemets of wOH prune all the weights which are < 4n2
                for(int j=0,j<no,j++){
                        if(fabs(wHO[h][j])<(4*n2)) {
                                wHO[h][j]=0;
                                extHO[h][j]=-1;
                                HOpruned++,
                                }
                        }
        cout<<" OH pruned   "<<HOpruned<<endl,

}

void calcNodeExistence(int eInp[], int eHid[], int eOut[],int extIH[100][100],int
extHO[100][40],int ni,int nh,int no) {

        //eliminate Input Nodes
        for(int i=0,i<ni,i++) {
                int state=0,
                if(extIH[i][0]==-1)
                        state=1,
                else
                        continue,

                for(int j=1,j<nh,j++) {
                        if(extIH[i][j]==-1)
                                continue,
                        else {
                                state=2,
                                break,
                                }
                }

                if(state==1) {
                        eInp[i]=-1,
                        cout<<"Input Node   "<<i<<" Eliminated"<<endl,
                        }
                }

        //eliminate Hidden nodes 1

        for(int i=0,i<nh;i++) {
                int state=0,
                if(extIH[0][i]==-1)
                        state=1,
                else
                        continue,

                for(int j=1,j<ni,j++) {
                        if(extIH[j][i]==-1)
                                continue,
                        else {
                                state=2,
                                break,
                                }
                }

                if(state==1) {
```

```
                        eHid[i]=-1,
                        cout<<"Hidden Node   "<<i<<" Eliminated"<<endl,
                        }
                }
```

//eliminate Hidden nodes 2

```
        for(int i=0,i<nh,i++) {
                int state=0,
                if(extHO[i][0]==-1)
                        state=1,
                else
                        continue,

                for(int j=1,j<no;j++) {
                        if(extHO[i][j]==-1)
                                continue,
                        else {
                                state=2,
                                break,
                                }
                }

                if(state==1) {
                        eHid[i]=-1,
                        cout<<"Hidden Node   "<<i<<" Eliminated"<<endl,
                        }
                }

        //eliminate Output nodes

        for(int i=0,i<no,i++) {
                int state=0;
                if(extHO[0][i]==-1)
                        state=1,
                else
                        continue,

                for(int j=1;j<nh,j++) {
                        if(extHO[j][i]==-1)
                                continue,
                        else {
                                state=2;
                                break,
                                }
                }

                if(state==1) {
                        eOut[i]=-1,
                        cout<<"Output Node   "<<i<<" Eliminated"<<endl,
                        }
        }

}

void correctMatrices(int eInp[], int eHid[], int eOut[],int extIH[100][100],int
extHO[100][40],int ni,int nh,int no) {

        for(int i=0,i<ni,i++) {
                if(eInp[i]==-1) {
                        for(int j=0,j<nh,j++) {
                                extIH[i][j]=-1;
                                }
                }
        }

        for(int i=0;i<nh,i++) {
                if(eHid[i]==-1) {
                        for(int j=0,j<ni,j++) {
                                extIH[j][i]=-1,
                                }
                        for(int k=0,k<no,k++) {
                                extHO[i][k]=-1,
                        }
                }
        }

        for(int i=0;i<no,i++) {
                if(eOut[i]==-1) {
                        for(int j=0;j<nh,j++) {
```

```
                                extIH[j][1]=-1,
                                }
                        }
                }
        }
}
```

APPENDIX C. CLUSTERING SOURCE CODE

cluster.h

```
using namespace std,

class cluster {
        private
                int freq,
                double centroid,
                double radius,

                double confidenceR,
                int freqConf,

        public :
                cluster(double r),
                cluster(double elem, double r),

                //accessor methods
                int getfreq(),
                double getG();
                double getr();
                double getcr(),
                int getfreqConf(),

                //set methods
                void setr(double r),
                void setConfR(double cr),

                //add methods
                double addElem(double e), //returns centroid

                //other
                bool inCluster(double e), //returns true, if an element belongs to the
cluster
                bool inConfCluster(double e), //returns true, if an element belongs to the
cluster
                //bool clusterIntersect(cluster& c), //returns true if this cluster
intersects cluster c
                double distFromG(double e),// returns the distance from the centroid
                void updateConfFreq(),//increments the confidence freq

};
```

clusterspace1dim.h

```
//#ınclude "cluster h"
#ınclude <vector>

class clusterspace1dim {
        prıvate
                vector<cluster> CSP,
                double clusterRadıus,
                double confR,
                vector<double>* valueSet,

                voıd addToSpace(double e),

        publıc
                clusterspace1dım(double r, vector<double>* vs),

                voıd add(double e),
                cluster* fındBestCluster(double e),
                voıd showSpace(),
                cluster* actıvateRadıus(double e),
                cluster* actıvateConfRadıus(double e),
                voıd calcConfıdenceFreq(double cr),

};
```

CentroidActivationLayer.h

```
//#ınclude "clusterspace1dım h"

class CentroidActivationLayer {
        prıvate
                clusterspace1dım* ActNodes[100],
                ınt sıze,
        publıc
                CentroıdActıvatıonLayer(vector<double>* ActVal[], ınt s, double radıus),

                voıd actıvateCR(double layer[]),
                voıd actıvateCRFreq(double [],ınt),
                voıd calcConfClusters(double cr),

                bool fılterActıvatıon(double layer[],ınt ext[],ınt s),
                bool fılterActıvatıonPercentage(double layer[],ınt ext[],ınt s,ınt perc),
};
```

cluster.cpp

```
#ınclude<math h>
#ınclude "cluster h"

usıng namespace std,

cluster  cluster(double r) {
        centroıd=0 0,
        radıus=fabs(r),
        freq=0,
        confıdenceR=-1,
        freqConf=0,
}

cluster  cluster(double elem, double r) {
        addElem(elem),
        radıus=fabs(r);
}

double cluster··addElem(double e) {
        ıf(freq==0) {
                centroıd=e,
                freq++,
                }
        else {
                centroıd=(centroıd*freq+e)/(freq+1),
                freq++,
                }

        return centroıd,
}

ınt cluster  getfreq() {
        return freq,
        }

double cluster  getr() {
        return radıus,
        }

double cluster  getcr() {
        return confıdenceR,
        }

double cluster  getG() {
        return centroıd,
        }

voıd cluster  setr(double r) {
        radıus=r,
        }

voıd cluster  setConfR(double cr) {
        confidenceR=cr,
        }

ınt cluster  getfreqConf() {
        return freqConf,
        }

double cluster  dıstFromG(double e) {
        return (fabs(centroıd-e)),
}

bool cluster  ınCluster(double e) {
        double temp=dıstFromG(e),
        ıf(temp<=radıus)
                return true,
        else
                return false,
}

bool cluster  ınConfCluster(double e) {
        double temp=dıstFromG(e),
        ıf(temp<=confıdenceR)
                return true,
        else
                return false,
}
```

```
void cluster  updateConfFreq() {
        freqConf++,
        }
```

clusterspace1dim.cpp

```
#include "cluster h"
#include "clusterspace1dim h"
#include<math h>
#include <iostream>

using namespace std,

clusterspace1dim::clusterspace1dim(double r, vector<double>* vs) {
        clusterRadius=fabs(r),
        confR=-1,
        valueSet=vs,

        std  vector<double>  iterator it,
        int idx=0,
        for(it=valueSet->begin(),it'=valueSet->end(),it++,idx++) {
                addToSpace((*valueSet)[idx]),
        }
        //cout<<"Cluster Space Constructed"<<endl,
}

void clusterspace1dim::addToSpace(double e) {
        cluster* tempCl=findBestCluster(e),
        if(tempCl && tempCl->inCluster(e)) {
                        (*tempCl) addElem(e),
        }
        else {
                cluster newCl(clusterRadius),
                newCl addElem(e),
                CSP push_back(newCl),
        }
}


void clusterspace1dim::showSpace() {
        std  vector<cluster>  iterator itr,

        cout<<"Cluster Radius . "<<clusterRadius<<endl,
        if(confR>=0) {
                cout<<"Confidence Radius   "<<confR<<endl,
                }

        int idx=0,
        for(itr=CSP begin(),itr'=CSP end(),itr++,idx++) {
                cout<<"Cluster "<<idx<<" : ",
                cout<<"G="<<CSP[idx] getG()<<" | ",
                cout<<"Freq="<<CSP[idx] getfreq(),
                if(confR>=0) {
                        cout<<" | Conf Freq="<<CSP[idx] getfreqConf(),
                                }
                cout<<endl,
                }
}


cluster* clusterspace1dim::findBestCluster(double e) {
        cluster *retr=0,
        std ·vector<cluster>· iterator itr,

        int idx=0,
        double foundR,

        for(itr=CSP begin(),itr'=CSP end(),itr++,idx++) {
                        double tempr=CSP[idx] distFromG(e),
                        if(itr==CSP begin()) {
                                foundR=tempr,
                                retr=&CSP[idx],
                                }
                        else {
                                if (tempr<foundR) {
```

```
                                        foundR=tempr,
                                        retr=&CSP[idx],
                                        }
                                }
                }

        /*
        if(retr) {
                if(retr->inCluster(e))
                        return retr,
                else {
                        cout<<retr->getG()<<" "<<e<<endl,
                        return 0;
                        }
                }
        else {
                cout<<e<<endl,
                return 0,
                }
        */

        return retr,
}

void clusterspace1dim::add(double e) {
        valueSet->push_back(e);
        addToSpace(e),
}

void clusterspace1dim::calcConfidenceFreq(double cr) {
        //use once only, to re compute conf clusters, we need to reinitialize the clusters
        //if used more than once, the update freq will not keep on incrementing frequencies
        confR=cr,
        std  vector<cluster>  iterator itc,

        int idx=0,
        for(itc=CSP begin(),itc!=CSP end(),itc++,idx++) {
                CSP[idx] setConfR(cr),
        }

        std  vector<double>  iterator it,
        idx=0,
        for(it=valueSet->begin(),it!=valueSet->end(),it++,idx++) {
                cluster* temp=findBestCluster((*valueSet)[idx]),
                if(temp==0)
                        cout<<"Some error updating Confidence freq"<<endl;
                else{
                        if(temp->inConfCluster((*valueSet)[idx]))
                                temp->updateConfFreq(),
                        }
                }
}

cluster* clusterspace1dim::activateRadius(double e) {
        cluster* retrieved=0,
        retrieved=findBestCluster(e),

        if(retrieved && retrieved->inCluster(e))
                return retrieved,
        else
                return 0,

}

cluster* clusterspace1dim::activateConfRadius(double e) {
        if(confR<0) {
                cout<<"Confidence Frequencies not computed   Cluster Space cannot be
activated"<<endl,
                return 0,
                }
        cluster* retrieved=0,
        retrieved=findBestCluster(e),

        if(retrieved && retrieved->inConfCluster(e))
                return retrieved,
        else
                return 0,

}
```

CentroidActivationLayer.cpp

```
#ınclude <ıostream>
#ınclude "cluster h"
#ınclude "clusterspace1dım h"
#ınclude "CentroıdActıvatıonLayer h"
#ınclude<vector>

CentroidActivationLayer::CentroidActivationLayer(vector<double>* ActVal[],int s, double
radius) {
        sıze=s;
        for(ınt ı=0,ı<sıze,ı++) {
                clusterspace1dım* tempCl=new clusterspace1dım(radıus, ActVal[ı]);
                ActNodes[ı]=tempCl,
                }
        cout<<"Centroıd Actıvatıon Layer Created   "<<sıze<<"Nodes"<<endl,
}

void CentroidActivationLayer::activateCR(double layer[]) {
        for(ınt ı=0,ı<sıze,ı++) {
                cluster* tempC=ActNodes[ı]->actıvateConfRadıus(layer[ı]);
                ıf(tempC){
                        layer[ı]=tempC->getG(),
                        cout<<layer[ı]<<" ",
                        }
                else {
                        layer[ı]=0,
                        cout<<layer[ı]<<" ",
                        }
                }
                cout<<endl,
}

void CentroidActivationLayer::activateCRFreq(double layer[],int filtFreq) {
        for(ınt ı=0,ı<sıze,ı++) {
                cluster* tempC=ActNodes[ı]->actıvateConfRadıus(layer[ı]),
                ıf(tempC && ((tempC->getfreqConf())>=fılt Freq)){
                        layer[ı]=tempC->getG(),
                        //cout<<layer[ı]<<" ",
                        }
                else {
                        layer[ı]=0;
                        //cout<<layer[ı]<<" ",
                        }
                }
                //cout<<endl;
}

void CentroidActivationLayer::calcConfClusters(double cr) {
        for(ınt ı=0;ı<sıze,ı++) {
                ActNodes[ı]->calcConfıdenceFreq(cr),
                }
}

bool CentroidActivationLayer::filterActivation(double layer[],int ext[],int s) {
        bool ınvalıdAct=false,
        ınt checkActıve=0;
        ınt checkDeAct=0;

        for(ınt ı=0,ı<s,ı++) {
                ıf(layer[ı]==0) {
                        ıf(ext[ı]==1) {
                                ınvalıdAct=true,
                                break,
                                }
                        }
                else {
                        checkActıve++,
                        }

                }


        ıf(ınvalıdAct) {
        //cout<<"Deactıvatıng Layer"<<endl,
                for (ınt ı=0,ı<s,ı++) {
                        layer[ı]=0;
```

```
                }
        cout<<"-",
        return false,
        }
        else {
                cout<<"*"<<checkActive;
                return true,
                }
}

bool CentroidActivationLayer::filterActivationPercentage(double layer[],int ext[],int
s,int perc) {
        int active=0,
        int totExt=0,
        //static int totHardPass=0,

        for(int i=0,i<s,i++) {
                if(ext[i]==1) {
                        totExt++,
                        if(layer[i]==0) {
                                continue,
                                }
                        else {
                                active++;
                                continue,
                                }
                        }
                else if(ext[i]==-1) {
                        continue,
                        }
                }

        /*
        if(active==totExt)
                totHardPass++,
        */

        double passFilter=((double)active/(double)totExt)*100,

        if(passFilter<perc) {
                //cout<<"Deactivating Layer   Active="<<pass<<endl,
                for (int i=0,i<s,i++) {
                        layer[i]=0,
                }
                //cout<<"-",
                return false,
        }
        else {
                //cout<<"*"<<active,
                //cout<<" Active   "<<active<<" Tot EXT : "<<totExt<<"perc   "<<perc<<"
pass : "<<passFilter/*<<" TotHardPass · "<<totHardPass*/<<endl,

                return true,
                }
}
```

APPENDIX D. NEURAL NETWORK RULE EXTRACTION AND PREDICTION SOURCE CODE

testNetworkGAct.cpp

```
//maın funtıon to load a specıfic network and to carry out workflow of extractıng rules from exıstıng dataset
#ınclude "cluster h"
#ınclude "clusterspace1dım h"
#ınclude "CentroıdActıvatıonLayer h"
#ınclude "myNetwork h"
#ınclude <fstream>
#ınclude <ıostream>
#ınclude <stdlıb h>
#ınclude <stdıo h>

usıng namespace std,

ınt ınıt_traınıng_buffer(ıfstream& ,ınt ,ınt , float [][100],float [][40]),

int main( int argc, char** argv ){
        //Readıng Archıtecture
        //arg 1 = TestID
        char tempA[20]="Arch",
        char* Afıle=strcat(tempA,argv[1]),
        ınt numInp=-1,
        ınt numOut=-1,
        ınt numHıdden=-1,
        ıfstream archRead(Afıle),
        archRead>>numInp,
        archRead>>numHıdden,
        archRead>>numOut,

        archRead close(),

        //readıng Traınıng Data Set
        //arg 2 = traınıng file
        ıfstream traınFıle(argv[2]),

        float ınpBuf[3100][100],
        float outBuf[3100][40],

        ınt numTSample=ınıt_traınıng_buffer(traınFıle,numInp,numOut,ınpBuf,outBuf),
        cout<<"Extractıng usıng G Actıvated Recall"<<endl,
        cout<<"Test ID   "<<argv[1]<<endl,
        cout<<"*Tranıng buffer Inıtıalızed*  "<<numTSample<<" Patterns"<<endl,
        traınFıle close(),

        //filenames
        char tempw[20]="Weıghts",
        char tempe[20]="ext",
        char tempt[20]="Threshold",
        char tempHA[20]="HıdAct",

        char* wfıle=strcat(tempw,argv[1]),
        char* efıle=strcat(tempe,argv[1]),
        char* tfıle=strcat(tempt,argv[1]);
        char* HAfıle=strcat(tempHA,argv[1]),

        //build the network
        myNetwork NetA(wfıle,efıle,tfıle),
        cout<<endl,

        cout<< endl,
        cout<<"Data Fıle   "<<argv[2]<<endl,
        cout<<"Weıght Fıle   "<<wfıle<<endl,
        cout<<"Exıstence Fıle   "<<efıle<<endl,
        cout<<"Threshold Fıle   "<<tfıle<<endl,
```

```
        //build the Centroid Activation Layer
        double rad,confRad,
        int filterFreq,
        cout<<"Hid unit act values   "<<HAfile<<endl,
        cout<<"Enter Cluster Radius   ",
        //assign radius and calculate confidence radius
        cin>>rad,
        cout<<endl,
        NetA addActivationLayer(rad,HAfile),

        confRad=0 5*rad,
        NetA computeConfCluster(confRad),

        //dynmically recall rules for different frequencies
        while(true) {
        cout<<"Conf Radius . "<<confRad<<" Enter Confidence Frequency Filter Percentage
",
        cin>>filterFreq,
        cout<<endl,
        int confFreqNum=(int)(((double)filterFreq/100 00)*numTSample),
        cout<<"Number of Samples selectd as filter frequency   "<<confFreqNum<<endl,

        cout<<"Network Recalling with G activations    "<<endl,

        //set the activation level as the percentage of hidden layer neurons
        double acu=-1,
        int filtLayerPerc,
        cout<<"Enter the Percentage of hidden layer nodes which need to be activated for
the pattern to pass   ",
        cin>>filtLayerPerc,

        acu=NetA GActivationRecallPercentage(inpBuf,outBuf,numTSample,filterFreq,argv[1],f
iltLayerPerc),
        }
        cout<<"*Recall With G Activations Complete*  Accuracy · "<<acu<<endl,
        cout<<endl,
        }


}
```

testPredict.cpp

//main function which extracts predicted trends from an input dataset

//the input dataset is all the combinations of the input attributes of a particular dataset, hence this maybe large This function takes the automatically takes input in batches of 20000 and extracts predicted rules from the loaded network To ensure this, the data must be split in to files using the UNIX Split —l command into batches of less than 20,000 The file name format will be like <filename>aa, etc The input to the program should be just <filename> The sequence is generated automatically similar to the split command

//the hidden unit activation values are loaded from the file which contains activation values for the training data set in the HidAct file This is done automatically*

```
#include "cluster h"
#include "clusterspace1dim h"
#include "CentroidActivationLayer h"
#include "myNetwork h"
#include <fstream>
#include <iostream>
#include <stdlib h>
#include <stdio h>

using namespace std,

int init_test_buffer(ifstream& ,int, short [][100]),
//function to generate the sequence similar to the split command
char* generateNextSequence(char*),

int main( int argc, char** argv ){
        cout<<"*******************************************************"<<endl,
        cout<<"*Extracting Generalized Rules using G Activated Recall*"<<endl,
        cout<<"*******************************************************"<<endl,
        cout<<endl,

        //Reading Architecture
        //arg 1 = TestID
        char tempA[20]="Arch";
        char* Afile=strcat(tempA,argv[1]),

        int numInp=-1;
        int numOut=-1,
        int numHidden=-1,
        ifstream archRead(Afile),

        archRead>>numInp,
        archRead>>numHidden,
        archRead>>numOut;

        archRead close(),
        cout<<"*Architecture Initialized*  "<<numInp<<" "<<numHidden<<" "<<numOut<<endl,

        //filenames
        char tempw[20]="Weights",
        char tempe[20]="ext",
        char tempt[20]="Threshold",
        char tempHA[20]="HidAct",

        char* wfile=strcat(tempw,argv[1]),
        char* efile=strcat(tempe,argv[1]);
        char* tfile=strcat(tempt,argv[1]);
        char* HAfile=strcat(tempHA,argv[1]),


        //build the network
        myNetwork NetA(wfile,efile,tfile),
        cout<<endl,

        cout<< endl,
        cout<<"Test ID   "<<argv[1]<<endl,
        cout<<"Data File   "<<argv[2]<<endl,
        cout<<"Weight File . "<<wfile<<endl,
        cout<<"Existence File . "<<efile<<endl,
        cout<<"Threshold File · "<<tfile<<endl,
        cout<<endl,

        //build the Centriod Activation Layer
                double rad,confRad;
```

```
                int filterFreq,
                int filtLayerPerc,
                cout<<"Hid unit act values   "<<HAfile<<endl,
                cout<<"Enter Cluster Radius   ",
                cin>>rad,
                confRad=0 5*rad,

                NetA addActivationLayer(rad,HAfile),
                NetA computeConfCluster(confRad),

        //set the frequency and activation level
                cout<<"Conf Radius   "<<confRad<<" Enter Confidence Frequency Filter
Percentage  ",
                cin>>filterFreq,

                cout<<"Enter the Percentage of hidden layer nodes which need to be
activated for the pattern to pass   ",
                cin>>filtLayerPerc,

                cout<<endl,

        //extract predicted rules in batches
        int pass=0,
        while(true) {
                short inpBuf[20000][100],
                char suffix[3]="",
                char currentfilename[30]="",

                pass++,

                strcpy(currentfilename,argv[2]),
                generateNextSequence(suffix),
                strcat(currentfilename,suffix),

                cout<<"Opening File  "<<currentfilename<<"    ",
                ifstream trainFile(currentfilename),

                if(!trainFile) {
                        cout<<"*No more file - Finished Prediction Process'!*"<<endl,
                        break,
                }

                int numTSample=init_test_buffer(trainFile,numInp,inpBuf),
                cout<<" Pass"<<pass<<" *Buffer Initialized "<<numTSample<<" Patterns*";
                trainFile close(),

                cout<<"   Recalling    ",
                double acu=-1;

        acu=NetA GActivationRecallGeneralized(inpBuf,numTSample,filterFreq,argv[1],filtLay
erPerc),

                cout<<" *Complete*  Accuracy   "<<acu,
                cout<<endl,
        }
}
```

APPENDIX E. NEURAL NETWORK RECALL SOURCE CODE

recall.cpp

```
//methods for recalling the network --simple and through activation layer
#include "cluster h"
#include "clusterspace1dim h"
#include "CentroidActivationLayer h"
#include "myNetwork h"
#include <fstream>
#include <iostream>
#include <stdlib h>
#include <math.h>

using namespace std,

//simple recall
double myNetwork::recall(float iBuffer[][100], float oBuffer[][40],int& testSetSize,char*
testID){

        char temp[20]="testOutput", //testOutput* file for output
        char tempAct[20]="HidAct", //HidAct* file for hidden unit activation values

        ofstream fout(strcat(temp,testID)),
        ofstream foutAct(strcat(tempAct,testID)),

                //data to determine accuracy percentage
                int corrCntr=0,

                for(int p=0,p<testSetSize,p++) {                //inner loop

                        fout<<"p"<<p<<"    ",

                        //input
                        for(int iDim=0,iDim<numberOfInputNodes,iDim++) {
                                input[iDim]=iBuffer[p][iDim],
                                fout<<input[iDim];
                                fout<<" ",
                                }

                        fout<<"    ",

                        //expected output
                        for(int iDim=0;iDim<numberOfOutputNodes,iDim++) {
                                fout<<oBuffer[p][iDim],
                                fout<<" ",
                                }

                        fout<<" · ",


                        for (int i = 0, i < numberOfHiddenNodes; i++)
                        {
                                hiddenNodeSum[i] = 0,
                        }

                        for (int i = 0, i < numberOfOutputNodes, i++)
                        {
                                outputNodeSum[i] = 0,
                        }




// Run Input Through WeightsAtoB To Hidden Layer
                        for ( int i = 0, i < numberOfHiddenNodes, i++)
                        {
                                if(extHidden[i]==-1) {
```

```
                                        hıddenNodeSum[ı]=0,
                                }
                                else {
                                        for (ınt  ȷ = 0, ȷ < numberOfInputNodes, ȷ++)
                                        {
                                                ıf(extInput2Hıdden[ȷ][ı]'=-1)
                                                        hıddenNodeSum[ı] += ınput[ȷ] *
weıghtInput2Hıdden[ȷ][ı],
                                        }
                                        hıddenNodeSum[ı] = 1 0 / (1 0 + exp(- 01 *
hıddenNodeSum[ı])),
                                }
                                foutAct<<hıddenNodeSum[ı]<<" ",
                        }
                        foutAct<<endl,

// Run Sıgnal From Hıdden Layer to Output Layer
                        for ( ınt ı = 0, ı < numberOfOutputNodes, ı++)
                        {
                                ıf(extOutput[ı]==-1) {
                                        outputNodeSum[ı]=-1,
                                }
                                else {
                                        for ( ınt ȷ = 0, ȷ < numberOfHıddenNodes, ȷ++)
                                        {
                                                ıf(extHıdden2Output[ȷ][ı]'=-1)
                                                        outputNodeSum[ı] += hıddenNodeSum[ȷ]
* weıghtHıdden2Output[ȷ][ı],
                                        }

                                        //double tempSto,
                                        //tempSto=outputNodeSum[ı],
                                        outputNodeSum[ı] = (1 0 / (1.0 + exp(- 01 *
(outputNodeSum[ı] + outputThreshold[ı])))),
                                        //cout<<outputNodeSum[ı]<<" | ",
                                        //ıf(outputNodeSum[ı] > 1.0)
                                        //      cout<<"Some Error   "<<tempSto<<endl,
                                }
                                fout<<outputNodeSum[ı]<<" ",
                        }


//check ıf the pattern ıs classıfied correctly

        bool corr=true,
        for(ınt ı=0,ı<numberOfOutputNodes,ı++) {
                ıf(extOutput[ı]==-1) {
                        contınue,
                        }
                ıf((myRound(outputNodeSum[ı]))==oBuffer[p][ı]) {
                        contınue,
                        }
                else {
                        ıf(extOutput[ı]==-1)
                                contınue,
                        else {
                                corr=false;
                                break,
                                }
                        }
        }

        ıf(corr==true) {
                fout<<"   CORRECT",
                corrCntr++,
                }
        else {
                fout<<"   INCORRECT",
                }

                fout<<endl,

        } // end ınner loop

        fout<<"No  of Patterns   "<<testSetSıze<<endl,
        fout<<"No  correctly recognızed   "<<corrCntr<<endl,

        fout.close();
        foutAct close(),
```

```
        double accuracy=(double)corrCntr/(double)testSetSize,

        return accuracy,
}

//rounding function

double myRound(double x) {
        if(x>=0 5)
                return 1 0,
        else
                return 0 0,
}


//clustering methods
void myNetwork::computeConfCluster(double confRad) {
        //compute Confidence Frequencies
        CAL->calcConfClusters(confRad),
}


//recall using centroid activations
//use only after computing conf clusters
double myNetwork::GActivationRecall(float iBuffer[][100], float oBuffer[][40],int&
testSetSize, int freqFilter,char* testID){

        char temp[20]="GAOutput",
        char temp1[20]="GAExt",

        ofstream fout(strcat(temp,testID)),
        ofstream foute(strcat(temp1,testID)),

        int deactivated=0,

                //data to determine accuracy percentage
                int corrCntr=0,

                for(int p=0,p<testSetSize,p++) {                //inner loop

                        fout<<"p"<<p<<"   ",

                        //input
                        for(int iDim=0,iDim<numberOfInputNodes,iDim++) {
                                input[iDim]=iBuffer[p][iDim],
                                fout<<input[iDim];
                                fout<<" ",
                                }

                        fout<<"   ",

                        //expected output
                        for(int iDim=0,iDim<numberOfOutputNodes,iDim++) {
                                fout<<oBuffer[p][iDim],
                                fout<<" ",
                                }

                        fout<<"   ",


                        for (int i = 0; i < numberOfHiddenNodes, i++)
                        {
                                hiddenNodeSum[i] = 0;
                        }

                        for (int i = 0, i < numberOfOutputNodes; i++)
                        {
                                outputNodeSum[i] = 0,
                        }


// Run Input Through WeightsAtoB To Hidden Layer
                        for ( int i = 0, i < numberOfHiddenNodes, i++)
                        {
                        if(extHidden[i]==-1) {
                                hiddenNodeSum[i]=0;
                        }
                        else {
                                for (int  j = 0, j < numberOfInputNodes, j++)
```

```
                                {
                                        if(extInput2Hidden[j][i]'=-1) {
                                                hiddenNodeSum[i] += input[j] *
weightInput2Hidden[j][i],
                                                }
                                }
                                hiddenNodeSum[i] = 1 0 / (1 0 + exp(- 01 *
hiddenNodeSum[i])),
                        }
                        }
```

//activate the centroids of the clusters for the hidden layer activations

```
                CAL->activateCRFreq(hiddenNodeSum,freqFilter),
```

//Filter the activation values

```
                bool actSuccess=false;
                actSuccess=CAL-
>filterActivation(hiddenNodeSum,extHidden,numberOfHiddenNodes),

                if('actSuccess) {
                        for(int i=0,i<numberOfOutputNodes,i++) {
                                fout<<"- ",
                                }
                        fout<<"   Layer Not Activated   No Output"<<endl,
                        deactivated++;
                        continue,
                }
```

// Run Signal From Hidden Layer to Output Layer

```
                        for ( int i = 0, i < numberOfOutputNodes, i++)
                        {
                        if(extOutput[i]==-1) {
                                outputNodeSum[i]=-1,
                        }
                        else {
                                for ( int j = 0, j < numberOfHiddenNodes, j++)
                                {

                                        if(extHidden2Output[j][i]'=-1) {
                                                outputNodeSum[i] += hiddenNodeSum[j] *
weightHidden2Output[j][i],
                                        }
                                }

                                //double tempSto,
                                //tempSto=outputNodeSum[i],
                                outputNodeSum[i] = (1 0 / (1 0 + exp(- 01 *
(outputNodeSum[i] + outputThreshold[i])))),
                                //cout<<outputNodeSum[i]<<" | ",
                                //if(outputNodeSum[i] > 1 0)
                                //      cout<<"Some Error   "<<tempSto<<endl,
                        }

                                fout<<outputNodeSum[i]<<" ";
                        }
```

//check if the pattern is classified correctly

```
        bool corr=true,
        for(int i=0,i<numberOfOutputNodes,i++) {
                if(extOutput[i]==-1)
                        continue,
                if(((myRound(outputNodeSum[i]))==oBuffer[p][i]))
                        continue,
                else {
                        if(extOutput[i]==-1)
                                continue;
                        else {
                                corr=false,
                                break,
                                }
                        }
        }

        if(corr==true) {
                fout<<" . CORRECT",
                corrCntr++,
```

```
                foute<<"Pattern "<<p<<" ",

                        //input
                        for(int iDim=0,iDim<numberOfInputNodes,iDim++) {
                                input[iDim]=iBuffer[p][iDim],
                                foute<<input[iDim],
                                foute<<" ",
                                }

                        foute<<"    ",

                        //expected output
                        for(int iDim=0,iDim<numberOfOutputNodes,iDim++) {
                                foute<<oBuffer[p][iDim],
                                foute<<" ",
                                }

                        foute<<" : ",

                        for ( int i = 0; i < numberOfOutputNodes, i++)
                        {
                                foute<<outputNodeSum[i]<<" ";
                        }

                        foute<<endl,
                }
        else {
                fout<<" · INCORRECT";
                }

                fout<<endl,

        } //inner loop

        cout<<endl,

        fout<<"No  of Patterns    "<<testSetSize<<endl,
        fout<<"No  correctly recognized    "<<corrCntr<<endl,

        cout<<"Number Correctly recognized . "<<corrCntr<<endl,
        cout<<"Number Deactivated    "<<deactivated<<endl,

        fout close(),
        foute close(),

        double accuracy=(double)corrCntr/(double)testSetSize,

        return accuracy,
}
```

//recall using centroid activations with capability of choosing activation level of hidden layer, i e percentage of hidden layer neurons which pass the frequency test for the defined radius
//use only after computing conf clusters

```
double myNetwork::GActivationRecallPercentage(float iBuffer[][100], float
oBuffer[][40],int& testSetSize, int freqFilter,char* testID,int HidPercPass){

        char temp[20]="GAOutput",
        char temp1[20]="GAExt";

        ofstream fout(strcat(temp,testID)),
        ofstream foute(strcat(temp1,testID)),

        int deactivated=0,

                //data to determine accuracy percentage
                int corrCntr=0,

                for(int p=0,p<testSetSize,p++) {                //inner loop

                        fout<<"p"<<p<<"    ",

                        //input
                        for(int iDim=0,iDim<numberOfInputNodes,iDim++) {
                                input[iDim]=iBuffer[p][iDim],
                                fout<<input[iDim],
                                fout<<" ";
                                }

                        fout<<" . ",
```

```
                        //expected output
                        for(int iDim=0,iDim<numberOfOutputNodes,iDim++) {
                                fout<<oBuffer[p][iDim],
                                fout<<" ",
                                }
                        fout<<"   ",


                        for (int i = 0, i < numberOfHiddenNodes, i++)
                        {
                                hiddenNodeSum[i] = 0,
                        }

                        for (int i = 0, i < numberOfOutputNodes, i++)
                        {
                                outputNodeSum[i] = 0,
                        }


// Run Input Through WeightsAtoB To Hidden Layer
                        for ( int i = 0, i < numberOfHiddenNodes, i++)
                        {
                        if(extHidden[i]==-1) {
                                hiddenNodeSum[i]=0,
                        }
                        else {
                                for (int  j = 0, j < numberOfInputNodes; j++)
                                {
                                        if(extInput2Hidden[j][i]'=-1) {
                                                hiddenNodeSum[i] += input[j] *
weightInput2Hidden[j][i],
                                                }
                                }
                                hiddenNodeSum[i] = 1 0 / (1 0 + exp(- 01 *
hiddenNodeSum[i])),
                        }
                        }

//activate the centroids of the clusters for the hidden layer activations

                CAL->activateCRFreq(hiddenNodeSum,freqFilter),

//Filter the activation values

                bool actSuccess=false,
                actSuccess=CAL-
>filterActivationPercentage(hiddenNodeSum,extHidden,numberOfHiddenNodes,HidPercPass),

                if('actSuccess) {
                        for(int i=0,i<numberOfOutputNodes,i++) {
                                fout<<"- ",
                                }
                        fout<<"   Layer Not Activated . No Output"<<endl,
                        deactivated++;
                        continue;
                }

// Run Signal From Hidden Layer to Output Layer
                        for ( int i = 0, i < numberOfOutputNodes; i++)
                        {
                        if(extOutput[i]==-1) {
                                outputNodeSum[i]=-1;
                        }
                        else {
                                for ( int j = 0, j < numberOfHiddenNodes, j++)
                                {

                                        if(extHidden2Output[j][i]'=-1) {
                                                outputNodeSum[i] += hiddenNodeSum[j] *
weightHidden2Output[j][i],
                                        }
                                }

                                //double tempSto,
                                //tempSto=outputNodeSum[i],
                                outputNodeSum[i] = (1 0 / (1 0 + exp(-.01 *
(outputNodeSum[i] + outputThreshold[i])))),
                                //cout<<outputNodeSum[i]<<" | ";
                                //if(outputNodeSum[i] > 1.0)
```

```
                        //      cout<<"Some Error    "<<tempSto<<endl,
                }

                        fout<<outputNodeSum[i]<<" ",
                }
```

//output the extracted rule

```
        foute<<"Rule "<<p<<" ",

                //input
                for(int iDim=0,iDim<numberOfInputNodes,iDim++) {
                        input[iDim]=iBuffer[p][iDim],
                        foute<<input[iDim],
                        foute<<" ",
                        }

                foute<<"    ",


                for ( int i = 0, i < numberOfOutputNodes, i++)
                {
                        foute<<myRound(outputNodeSum[i])<<" ",
                }

                foute<<endl,
```

//check if the pattern is classified correctly

```
        bool corr=true;
        for(int i=0,i<numberOfOutputNodes,i++) {
                if(extOutput[i]==-1)
                        continue,
                if(((myRound(outputNodeSum[i]))==oBuffer[p][i]))
                        continue,
                else {
                        if(extOutput[i]==-1)
                                continue,
                        else {
                                corr=false,
                                break,
                                }
                        }
        }

        if(corr==true) {
                fout<<"   CORRECT";
                corrCntr++;
                }
        else {
                fout<<" · INCORRECT";
                }

                fout<<endl;

        } // end inner loop

        cout<<endl,

        fout<<"No. of Patterns    "<<testSetSize<<endl;
        fout<<"No  correctly recognized . "<<corrCntr<<endl,

        cout<<"Number Correctly recognized    "<<corrCntr<<endl,
        cout<<"Number Deactivated    "<<deactivated<<endl;

        fout close(),
        foute close();

        double accuracy=(double)corrCntr/(double)testSetSize;

        return accuracy,
}
```

//recall method for outputing only the correctly classified patterns

```
double myNetwork::recallCorrect(float iBuffer[][100], float oBuffer[][40],int&
testSetSize,char* testID){

        char tempf[20]="FiltData", //FiltData*file
```

```
        char tempAct[20]="HıdAct",

        ofstream foutAct(strcat(tempAct,testID)),
        ofstream foutfd(strcat(tempf,testID)),

                //data to determıne accuracy percentage
                ınt corrCntr=0,

                for(ınt p=0,p<testSetSıze,p++) {              //ınner loop


                        //ınput
                        for(ınt ıDım=0,ıDım<numberOfInputNodes,ıDım++) {
                                ınput[ıDım]=ıBuffer[p][ıDım],
                                }


                        //expected output
                        for(ınt ıDım=0,ıDım<numberOfOutputNodes,ıDım++) {
                                }


                        for (ınt ı = 0, ı < numberOfHıddenNodes, ı++)
                        {
                                hıddenNodeSum[ı] = 0;
                        }

                        for (ınt ı = 0, ı < numberOfOutputNodes, ı++)
                        {
                                outputNodeSum[ı] = 0,
                        }


// Run Input Through WeıghtsAtoB To Hıdden Layer
                        for ( ınt ı = 0, ı < numberOfHıddenNodes, ı++)
                        {
                        ıf(extHıdden[ı]==-1) {
                                hıddenNodeSum[ı]=0,
                        }
                        else {
                                for (ınt  ȷ = 0, ȷ < numberOfInputNodes, ȷ++)
                                {
                                        ıf(extInput2Hıdden[ȷ][ı]'=-1)
                                                hıddenNodeSum[ı] += ınput[ȷ] *
weıghtInput2Hıdden[ȷ][ı],
                                }
                                hıddenNodeSum[ı] = 1 0 / (1 0 + exp(- 01 *
hıddenNodeSum[ı])),
                        }

                        }

// Run Sıgnal From Hıdden Layer to Output Layer
                        for ( ınt ı = 0, ı < numberOfOutputNodes, ı++)
                        {
                        ıf(extOutput[ı]==-1) {
                                outputNodeSum[ı]=-1;
                        }
                        else {
                                for ( ınt ȷ = 0, ȷ < numberOfHıddenNodes, ȷ++)
                                {
                                        ıf(extHıdden2Output[ȷ][ı]'=-1)
                                                outputNodeSum[ı] += hıddenNodeSum[ȷ] *
weıghtHıdden2Output[ȷ][ı];
                                }

                                //double tempSto,
                                //tempSto=outputNodeSum[ı];
                                outputNodeSum[ı] = (1 0 / (1.0 + exp(- 01 *
(outputNodeSum[ı] + outputThreshold[ı])))),
                                //cout<<outputNodeSum[ı]<<" | ",
                                //ıf(outputNodeSum[ı] > 1.0)
                                //      cout<<"Some Error   "<<tempSto<<endl,
                        }

                        }


//check ıf the pattern ıs classıfied correctly
```

```
        bool corr=true,
        for(int i=0,i<numberOfOutputNodes,i++) {
                if(extOutput[i]==-1)
                        continue,
                if((myRound(outputNodeSum[i]))==oBuffer[p][i])
                        continue,
                else {
                        if(extOutput[i]==-1)
                                continue,
                        else {
                                corr=false,
                                break,
                                }
                        }
        }

        if(corr==true) {
                        //fout<<"p"<<p<<"   ",

                        //input
                        for(int iDim=0,iDim<numberOfInputNodes,iDim++) {
                                //fout<<input[iDim],
                                foutfd<<input[iDim],
                                //fout<<" ",
                                foutfd<<" ",
                                }

                        //fout<<"   ",
                        foutfd<<"  ",

                        //expected output
                        for(int iDim=0,iDim<numberOfOutputNodes,iDim++) {
                                //fout<<oBuffer[p][iDim],
                                foutfd<<oBuffer[p][iDim],
                                //fout<<" ",
                                foutfd<<" ",
                                }

                        //fout<<"   ",
                        foutfd<<endl,

                        for(int i=0,i<numberOfOutputNodes,i++) {
                                //fout<<outputNodeSum[i]<<" ";
                        }

                        for ( int i = 0, i < numberOfHiddenNodes, i++)
                        {
                                foutAct<<hiddenNodeSum[i]<<" ";
                        }
                        foutAct<<endl,

                //fout<<"   CORRECT",
                corrCntr++,
                }
        else {
                //fout<<"   INCORRECT",
                }

                //fout<<endl;

        } //inner loop

        //fout<<"No  of Patterns . "<<testSetSize<<endl,
        //fout<<"No  correctly recognized   "<<corrCntr<<endl,


        //fout close(),
        foutAct close(),
        foutfd close(),

        double accuracy=(double)corrCntr/(double)testSetSize,

        return accuracy,
}
```

//recall method for prediction — uses only input of the patterns
//use only after computing conf clusters

```
double myNetwork::GActivationRecallGeneralized(short iBuffer[][100],int& testSetSize, int
freqFilter,char* testID,int HidPercPass){
```

```
        char temp1[20]="GAExtPredıcted",
        ofstream foute(strcat(temp1,testID),ıos· app),

        ınt deactıvated=0,

                //data to determıne accuracy percentage
                ınt corrCntr=0,

                for(ınt p=0,p<testSetSıze,p++) {              //ınner loop


                        //ınput
                        for(ınt ıDım=0,ıDım<numberOfInputNodes,ıDım++) {
                                ınput[ıDım]=ıBuffer[p][ıDım],
                                }


                        for (ınt ı = 0, ı < numberOfHıddenNodes, ı++)
                        {
                                hıddenNodeSum[ı] = 0,
                        }

                        for (ınt ı = 0, ı < numberOfOutputNodes, ı++)
                        {
                                outputNodeSum[ı] = 0,
                        }
//ınsert code here

// Run Input Through WeıghtsAtoB To Hıdden Layer
                        for ( ınt ı = 0; ı < numberOfHıddenNodes, ı++)
                        {
                        ıf(extHıdden[ı]==-1) {
                                hıddenNodeSum[ı]=0,
                        }
                        else {
                                for (ınt  ȷ = 0, ȷ < numberOfInputNodes, ȷ++)
                                {
                                        ıf(extInput2Hıdden[ȷ][ı]'=-1) {
                                                hıddenNodeSum[ı] += ınput[ȷ] *
weıghtInput2Hıdden[ȷ][ı],
                                                }
                                }
                                hıddenNodeSum[ı] = 1 0 / (1 0 + exp(- 01 *
hıddenNodeSum[ı])),
                        }
                        }

//actıvate the centroıds of the clusters for the hıdden layer actıvatıons

                CAL->actıvateCRFreq(hıddenNodeSum,freqFılter);

//Fılter the actıvatıon values

                bool actSuccess=false,
                actSuccess=CAL-
>fılterActıvatıonPercentage(hıddenNodeSum,extHıdden,numberOfHıddenNodes,HıdPercPass);

                ıf('actSuccess) {
                        for(ınt ı=0,ı<numberOfOutputNodes,ı++) {
                                //fout<<"- ",
                                }
                        //fout<<"   Layer Not Actıvated   No Output"<<endl,
                        deactıvated++;
                        contınue,
                }

// Run Sıgnal From Hıdden Layer to Output Layer
                        for ( ınt ı = 0, ı < numberOfOutputNodes, ı++)
                        {
                        ıf(extOutput[ı]==-1) {
                                outputNodeSum[ı]=-1;
                        }
                        else {
                                for ( ınt ȷ = 0, ȷ < numberOfHıddenNodes; ȷ++)
                                {

                                        ıf(extHıdden2Output[ȷ][ı]'=-1) {
```

```
                                                outputNodeSum[ı] += hıddenNodeSum[j] *
weıghtHıdden2Output[j][ı],
                                        }
                                }

                                //double tempSto,
                                //tempSto=outputNodeSum[ı],
                                outputNodeSum[ı] = (1 0 / (1 0 + exp(- 01 *
(outputNodeSum[ı] + outputThreshold[ı])))),
                                //cout<<outputNodeSum[ı]<<" | ",
                                //ıf(outputNodeSum[ı] > 1 0)
                                //      cout<<"Some Error   "<<tempSto<<endl,
                        }

                                //fout<<outputNodeSum[ı]<<" ",
                        }
```

//extract the pattern whıch passes

```
                corrCntr++,
                foute<<"P "<<p<<"   ",
                        for(ınt ıDım=0,ıDım<numberOfInputNodes,ıDım++) {
                                ıf(extInput[ıDım]==-1)
                                        foute<<"* ";
                                else {
                                        foute<<ınput[ıDım],
                                        foute<<" ",
                                        }
                                }
                foute<<"   ",
                        for ( ınt ı = 0, ı < numberOfOutputNodes, ı++) {
                                ıf(extOutput[ı]==-1) {
                                        foute<<"* ",
                                }
                                else {
                                        foute<<myRound(outputNodeSum[ı])<<" ",
                                }
                        }
                foute<<endl,
```

} // end ınner loop

```
        //cout<<endl,

        //foute<<"No. of Patterns   "<<testSetSıze<<endl,
        //foute<<"No  Actıvated    "<<corrCntr<<endl,
        //foute<<"Number Deactıvated   "<<deactıvated<<endl,

        cout<<" No  Actıvated   "<<corrCntr,

        foute.close(),

        double accuracy=(double)corrCntr/(double)testSetSıze,

        return accuracy,
}
```

APPENDIX F. OTHER FUNCTIONS SOURCE CODE

bufferIO2.cpp

```
//function to load the input-output pairs of the data into the training buffer
#include <vector>
#include <fstream>
#include <iostream>

using namespace std;

int init_training_buffer(ifstream& inputfile,int nInp,int nOut,float
inpBuffer[][100],float outBuffer[][40]){
        int idx=-1;

        while(true) {
                float tempVal;
                inputfile>>tempVal;
                if(inputfile ios ·eof()) {
                        break;
                        }

                idx++;

                inpBuffer[idx][0]=tempVal;
                //cout<<"Pattern   "<<idx<<" being read"<<endl;

                for(int i=1,i<nInp,i++) {
                        inputfile>>inpBuffer[idx][i];
                }
                for(int i=0,i<nOut,i++) {
                        inputfile>>outBuffer[idx][i];
                }

        }

        return idx+1;
}

//function to load the input of the patterns into the test buffer — for prediction
int init_test_buffer(ifstream& inputfile,int nInp,short inpBuffer[][100]){
        int idx=-1;

        while(true) {
                short tempVal;
                inputfile>>tempVal;
                if(inputfile ios  eof()) {
                        break;
                        }

                idx++;

                inpBuffer[idx][0]=tempVal;
                //cout<<"Pattern   "<<idx<<" being read"<<endl;

                for(int i=1,i<nInp,i++) {
                        inputfile>>inpBuffer[idx][i];
                }

        }

        return idx+1;
```

netCorrOut.cpp

```
//function to extract only the correctly classified pairs — for generating Filt* file
#include "cluster h"
```

```
#ınclude "clusterspace1dım h"
#ınclude "CentroıdActıvatıonLayer h"
#ınclude "myNetwork h"
#ınclude <fstream>
#ınclude <ıostream>
#ınclude <stdlıb h>
#ınclude <stdıo h>

usıng namespace std,

ınt ınıt_traınıng_buffer(ıfstream& ,ınt ,ınt , float [][100],float [][40]),

int main( int argc, char** argv ){
        //Readıng Archıtecture
        //arg 1 = TestID
        char tempA[20]="Arch",
        char* Afıle=strcat(tempA,argv[1]),

        ınt numInp=-1,
        ınt numOut=-1,
        ınt numHıdden=-1,
        ıfstream archRead(Afıle),

        archRead>>numInp,
        archRead>>numHıdden,
        archRead>>numOut,

        archRead close(),

        //readıng Traınıng Data Set
        //arg 2 = traınıng fıle
        ıfstream traınFıle(argv[2]),

        float ınpBuf[3100][100],
        float outBuf[3100][40],

        ınt numTSample=ınıt_traınıng_buffer(traınFıle,numInp,numOut,ınpBuf,outBuf),
        cout<<"Correct Recall"<<endl,
        cout<<"Test ID   "<<argv[1]<<endl,
        cout<<"*Tranıng buffer Inıtıalızed*  "<<numTSample<<" Patterns"<<endl,
        traınFıle close(),

        //fılenames
        char tempw[20]="Weıghts",
        char tempe[20]="ext",
        char tempt[20]="Threshold",
        char tempHA[20]="HıdAct",

        char* wfıle=strcat(tempw,argv[1]),
        char* efıle=strcat(tempe,argv[1]),
        char* tfıle=strcat(tempt,argv[1]),
        char* HAfile=strcat(tempHA,argv[1]),

        //buıld the network
        myNetwork NetA(wfıle,efıle,tfıle),
        cout<<endl,

        cout<< endl,
        cout<<"Data Fıle   "<<argv[2]<<endl,
        cout<<"Weıght Fıle   "<<wfile<<endl,
        cout<<"Exıstence Fıle . "<<efile<<endl,
        cout<<"Threshold Fıle   "<<tfile<<endl,

        cout<<"Network Recallıng Correct Patterns    "<<endl;
        double acu=NetA.recallCorrect(ınpBuf,outBuf,numTSample,argv[1]),
        cout<<"*Recallıng Correct Patterns Complete *  Accuracy   "<<acu<<endl;
}
```

seqGenerator cpp

```
#ınclude <stdlıb h>
#ınclude <strıng h>
#ınclude <ıostream>

usıng namespace std,

void incrChar(char* pos2,char* pos1) {
        //wıll only work tıll bz
        ınt last=122,
```

```
        int test=pos1[0],
        test++,

        if(test>last) {
                strcpy(pos1,"a"),

                char sto[2],
                int temp=pos2[0],
                sto[0]=++temp,
                sto[1]='\0',
                strcpy(pos2,sto),

                //cout<<" pos2   "<<pos2<<" pos1   "<<pos1,
        }
        else {
                char sto[2],
                sto[0]=test,
                sto[1]='\0',
                //cout<<" test   "<<test<<" sto   "<<sto,
                strcpy(pos1,sto),
        }
}


char* generateNextSequence(char* temp) {
        static char c1[2]="a",
        static char c0[2]="a",

        static int turn=0,

        if(turn==0) {
                //do nothing
        }
        else {
                incrChar(c1,c0),
        }

        strcpy(temp,c1),
        strcat(temp,c0),

        turn++,

        return temp,
}
```

generateRule cpp

```
//tool to generate rules in a readable form using description for the data
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string h>
#include <vector>

using namespace std,

int main (int argc,char** argv) {

        //loading description
        ifstream descFile(argv[1]),
        if('descFile) {
                cout<<"Error   Desc File Not Opened"<<endl;
                exit(1),
                }

        int catSize, inpSize,
        vector<string> categ,
        vector<string> intvl,

        descFile>>catSize,
        descFile>>inpSize,

        for(int i=0,i<catSize,i++) {
                string temp,
```

```
        descFile>>temp,
        categ push_back(temp),
}

while(true) {
        string temp,
        descFile>>temp,

        if(descFile ios  eof()) {
                break,
        }
        else {
                intvl push_back(temp),
        }
}

/*
for(int i=0,i<categ size(),i++) {
        cout<<categ[i]<<endl,
}

cout<<endl,

for(int i=0,i<intvl size(),i++) {
        cout<<intvl[i]<<endl,
}
*/

//loading data and mapping it to the interval name
ifstream dataFile(argv[2]),
if('dataFile) {
        cout<<"Error . Data File Not Opened"<<endl,
        exit(1),
        }

vector<vector<string>*> jar,

while(true) {
        short temp,
        dataFile>>temp,

        if(dataFile ios· eof()) {
                break,
        }

        vector<string>* pouch=new vector<string>(),
        jar push_back(pouch);

        if(temp==1) {
                (*pouch) push_back(intvl[0]),
        }

        for(int i=1,i<intvl size(),i++) {
                dataFile>>temp,
                if(temp==1) {
                        (*pouch) push_back(intvl[i]),
                }
        }
}


//generate the rules
ofstream outF(argv[3]),

for (int i=0, i<jar.size();i++) {
        outF<<"IF <",
        int catIDX=0,

        for(int x=0,x<inpSize,x++) {
                if(x'=0){
                        outF<<" AND ",
                        }
                outF<<"("<<categ[catIDX]<<" = "<<(*(jar[i]))[catIDX]<<")",
                catIDX++,
        }

        outF<<"> THEN <",

        for(int y=0,y<(catSize-inpSize),y++) {
                if(y'=0){
                        outF<<" AND ";
```

```
                        }
                outF<<"("<<categ[catIDX]<<" = "<<(*(jar[1]))[catIDX]<<")",
                catIDX++,
        }

        outF<<">"<<endl,

}

}
```

REFERENCES

R Andrews, J Diederich, and A B Tickle, Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks , *Knowledge-Based Systems*, vol 8, no 6, 1995

Joseph P Bigus, *Data Mining With Neural Networks: Solving Business Problems from Application Development to Decision Support,* McGraw-Hill, NY, 1996

Chuang, W and Yang, J, Extracting Sentence Segments for Text Summarization A Machine Learning Approach , *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'00)*, Athens, Greece pp 152-159, 2000

Mark W Craven and Jude W Shavlik, Using Neural Networks for Data Mining , *Future Generation Computer Systems special issue on Data Mining,* 1998

David J Hand, Heikki Mannila, Padhraic Smyth, *Principles of Data Mining (Adaptive Computation and Machine Learning),* Aug 2001

Deogun, J S , Raghavan, V V , Sarkar, A and Sever, H, "Data mining trends in research and developments", In Lin, T Y and Cercone, N (eds), *Rough Sets and Data Mining: Analysis of Imprecise Data*, Kluwer Academic Publishers, 1998

Amit Gupta, Generalized Analytic Rule Extraction for Feedforward Neural Networks , *IEEE transactions on knowledge and data engineering,* 1999

Jıawei Han, Mıcheline Kamber, *Data Mining Concepts and Techniques,* Morgan Kaufmann, San Francisco, California, 2001

Tony Kai, Yun Chan, Eng Chong Tan and Neeraj Haralalka, A Novel Neural Network for Data Mining , *8th International Conference on Neural Information Processing Proceedings Vol 2*, 2001

Wen-Syan Li and Chris Clifton, "Semantic integration in heterogeneous databases using neural networks", In *Proceedings of the 20th International Conference on Very Large Data Bases*, Chile, 1994

Kishan Mehrotra, Chilukuri K Mohan, Sanjay Ranka, *Elements of Artificial Neural Networks (Complex Adaptive Systems),* Cambridge, MA MIT Press, 1997

R O Rogers, A framework for parallel data mining using neural networks , *Technical Report 97-413*, Queen's University, Department of Computing and Information Science, November 1997 [Setiono1996] Rudy Setiono, Extracting Rules from Pruned Neural Networks for Breast Cancer Diagnosis , *Artificial Intelligence in Medicine*, 1996

R Setiono and H Liu, Analysis of hidden representations by greedy clustering , *Connection Science*, 10(1), 1998

Jason T L Wang, Qicheng Ma, Dennis Shasha and Cathy H Wu, Application of Neural Networks to Biological Data Mining A Case Study in Protein Sequence Classification , *The Sixth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining,* August 20-23, 2000 Boston, MA, USA

Chengqı Zhang, Schıchao Zhang, Shıchao Zhang, Berno Eugene Heymer, *Associatıon Rule Mınıng Models and Algorıthms (Lecture Notes ın Artıfıcıal Intelligence)*, Vol 2307, Apr 2002

Dataset Sources

McDonald, G C and Schwıng, R C 'Instabılıtıes of regressıon estımates relatıng aır pollutıon to mortalıty', *Technometrıcs*, vol 15, 463-482, 1973

Nıerenberg DW, Stukel TA, Baron JA, Daın BJ, and Greenberg ER, Determınants of plasma levels of beta-carotene and retınol *, Amerıcan Journal of Epıdemıology*, 1989,130 511-521

K W Penrose, A G Nelson, and A G Fısher, "Generalızed body composıtıon predıctıon equatıon for men usıng sımple measurement technıques", *Medıcıne and Scıence ın Sports and Exercıse*, vol 17, no 2, Aprıl 1985, p 189

Jıhong Zhao, Matthew C Scheıder, Quınt Thurman, Fundıng communıty polıcıng to reduce crıme Have cops grants made a dıfference? , *Crımınology & Publıc Polıcy,* Nov 2002 Vol 2

VITA

Sandesh Doddameti was born in Bangalore, India on December 20, 1975, the son of Dr. Ashok Doddameti and Dr. Sunanda Doddameti. He obtained his Bachelor s Degree in Architecture from R V College of Engineering, Bangalore Univerity, Bangalore, India, in 1998. During the following year he was employed by an Urban Planning and Township Planning Frim in Bangalore, India. He entered the Graduate College at the Texas State University at San Marcos, Texas to pursue graduate studies in Computer Science in 2001. During the course of his study he was employed as a research assistant to manage and analyze the data with Texas Statewide Tobacco Education and Prevention in San Marcos, Texas.

Permanent Address:

984, KB Sandra, RT Nagar Post

Bangalore, Karnataka — 560032

India

This thesis was typed by Sandesh Doddameti.