3-1-2000

DETECTION OF RARE EVENTS AND RULE EXTRACITION BY NEURAL NETWORKS AND DECISION TREES

Wooyoung Choe
Purdue University School of ECE

Okan K. Ersoy
Purdue University School of ECE

DETECTION OF RARE EVENTS AND RULE EXTRACTION BY NEURAL NETWORKS AND DECISION TREES

WOOYOUNG CHOE
OKAN K. ERSOY

TR-ECE 00-3
MARCH 2000

SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

DETECTION OF RARE EVENTS AND RULE EXTRACTION BY NEURAL NETWORKS AND DECISION TREES

by

Wooyoung Choe, Okan K. Ersoy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Choe, Wooyoung. Ph.D., Purdue University, December 1999, Detection of Rare Events and Rule Extraction by Neural Networks and Decision Trees. Major Professor: Okan K. Ersoy.

Sample stratification is a technique for making each class in a sample have equal influence on decision making. For classification with neural networks, it is often preferable to use a stratified sample including an equal number of examples from each class;. However, it is not always possible to have a stratified sample because of unavailability of examples (referred to as rare event cases). For rare event detection, we develop two sample stratification schemes in this research. The first scheme stratifies a sample by adding up the weighted sum of the derivatives during the backward pass of training. The second scheme uses bootstrap aggregating. After training neural networks with multiple sets of bootstrapped examples of the rare event classes and subsampled examples of common event classes, we perform multiple voting for classification. The second part of the research is on the development of rule extraction algorithms from neural networks. As a method of overcoming the common criticism of black box approach of neural networks, we propose rule extraction algorithms by neural networks and decision trees based on parallel self-organizing hierarchical neural network. The first system solves the limitation of default rules which do not provide any interpretation about the data. By using another neural network for the input data that cannot be covered by extracted rules from the stage, more rules are generated and they are helpful in decision-making. The second system includes decision trees in rule-extracting process and gives more efficient and accurate results. This adds explanation capability to neural networks.

1. INTRODUCTION

Artificial neural networks are a class of very powerful, general–purpose tools applied across broad range of industries in pattern recognition, prediction, optimization, control, and digital signal processing. Their ability to generalize and learn from data imitates the human's ability to learn from experience. Although conventional approaches have been proposed for solving many problems, neural networks provide exciting alternatives. Many applications could benefit from using these alternative schemes. Neural networks consist of a set of interconnected nodes. The output of each neural node is determined by the following equation:

$$y = f(\sum W_i X_i + \theta) \tag{1.1}$$

where X_i is the input from the i-th input node and W, is a weight factor associated with the link between the i-th input node and an output node. The function f is a nonlinear transfer function, and θ is a threshold value.

Data mining is one such application arena for neural networks. In its narrow sense, data mining can be viewed as a preliminary step in connection with knowledge discovery using data from large databases. A broader and more general definition of data mining is the automated discovery of new, non-obvious and potential useful information frorn large amounts of data. Data mining is inherently interdisciplinsuy in nature and typical data mining problems require a comprehensive approach including statistics, machine learning, decision trees, fuzzy logic, and neural networks. In particular, neural networks are powerful data mining tools because they have a proven track record, and novel types of problems and challenges for neural networks arise in many data mining and decision-support applications. Apparently, most data mining problems require an interdisciplinary effort, and a comprehensive approach might yield the ultimate solution to complex problems.

Another application arena for neural networks can be found in the "Human Genome Project" which involves sequencing the DNA in human chromosomes. Data interpretation is complex phenomena and poses a greatest challenge. Data interpretation can only be achieved using systems of adequate complexity. In most cases, the analytical construction of tractable models is out of the reach in present day theory. An alternative approach is to inductively construct complex, but tractable models using machine learning methods. One of the most successful techniques is the use of artificial neural networks to identify the behavior or the properties of processes for which complete theories are lacking. The abilities of these systems include those of any general purpose information processing system and are used for tasks such as pattern association, feature extraction or the approximation of any static or dynamic system. Given the reasonable constraints to the neural network model, a learning algorithm can develop a system that matches the underlying processes to any desired degree so that the neural model can be used to make predictions that would be very hard to obtain by other modeling methods.

In human genome project, a widespread use of neural networks seems to be inevitable, due to the exponentially increasing amount of experimental data that has to be interpreted. Neural networks can learn useful descriptions of genetic concepts when given only instances, rather than explicit definitions of those concepts. However, successful applications of inductive learning methods may only be expected if sufficient characteristic data for the adaptation and verification of the system is available.

A particular class of biological problems-the regulation of genes- has profoundly affected the choice of tools and the development of algorithms. In addition to genes, chromosomal DNA contains sequences that serve as signals for turning on and off gene expression. These signals are thought to be distributed as clusters in the regulatory regions of genes[1]. These segments which control gene expression can be anywhere in the genome, including in noncoding sequences (introns) of genes and in areas that might be many kilobases upstream or downstream from the transcription initiation sites. In addition, because the control elements are relatively short, they occur not only in regulatory regions but also elsewhere in the DNA sequence, probably by chance. Hence, locating the regulatory regions in the DNA is complex problem. This problem is one driving factor of our development of neural network schemes.

Our research addresses the question of how to use neural networks as well as decision trees, in conjunction with neural networks for interpreting genomic sequence data.. More specifically, we investigate how to locate the 5' end of genes in genomic sequences and explain the characteristics of the leader regions preceding the translation initiation site. We will also evaluate the strengths and weaknesses of our approaches, and identify research directions in neural networks for data mining applications. Fig.1.1 shows the framework for our schemes.

First stage of our approach to the problem is to detect leader regions and repetitive DNA sequences of Alu type. These goals are achieved by two kinds of neural network classifier systems. For recognition of repetitive sequences, we use a new type of neural network as a model for analyzing data that are rare and hard to obtain. This is based on detection of rare events by neural networks. For leader sequence detection, hybrid neural network-decision tree system is used because we can obtain not only higher accuracy but also underlying rules.

As mentioned before, explaining how we make a decision is an important issue and generating underlying rules is one of main topics in data mining, but neural networks cannot easily provide such information. This is the biggest criticism directed at neural networks. We can make up for the weakness. For that purpose, we extract rules from the trained neural networks and finally provide underlying rules about decision-malung process. That means we can add the explanation capability to neural networks and we can make a complete data mining system.

The thesis is organized into 6 chapters. Chapter 2 provides a brief description of human genome project and neural networks. In Chapter 3, we propose new learning algorithms for detecting rare events. The proposed methods are compared with the back propagation networks and networks using importance sampling. In Chapter 4, a rule extraction scheme with neural networks is proposed. Based on parallel self-organizing hierarchical neural networks, we provide its further development as rule extracting PSHNN. In Chapter 5, hybrid neural network and decision tree system for rule extraction with higher classification accuracy is presented. Chapter 6 covers conclusions and possible future research.

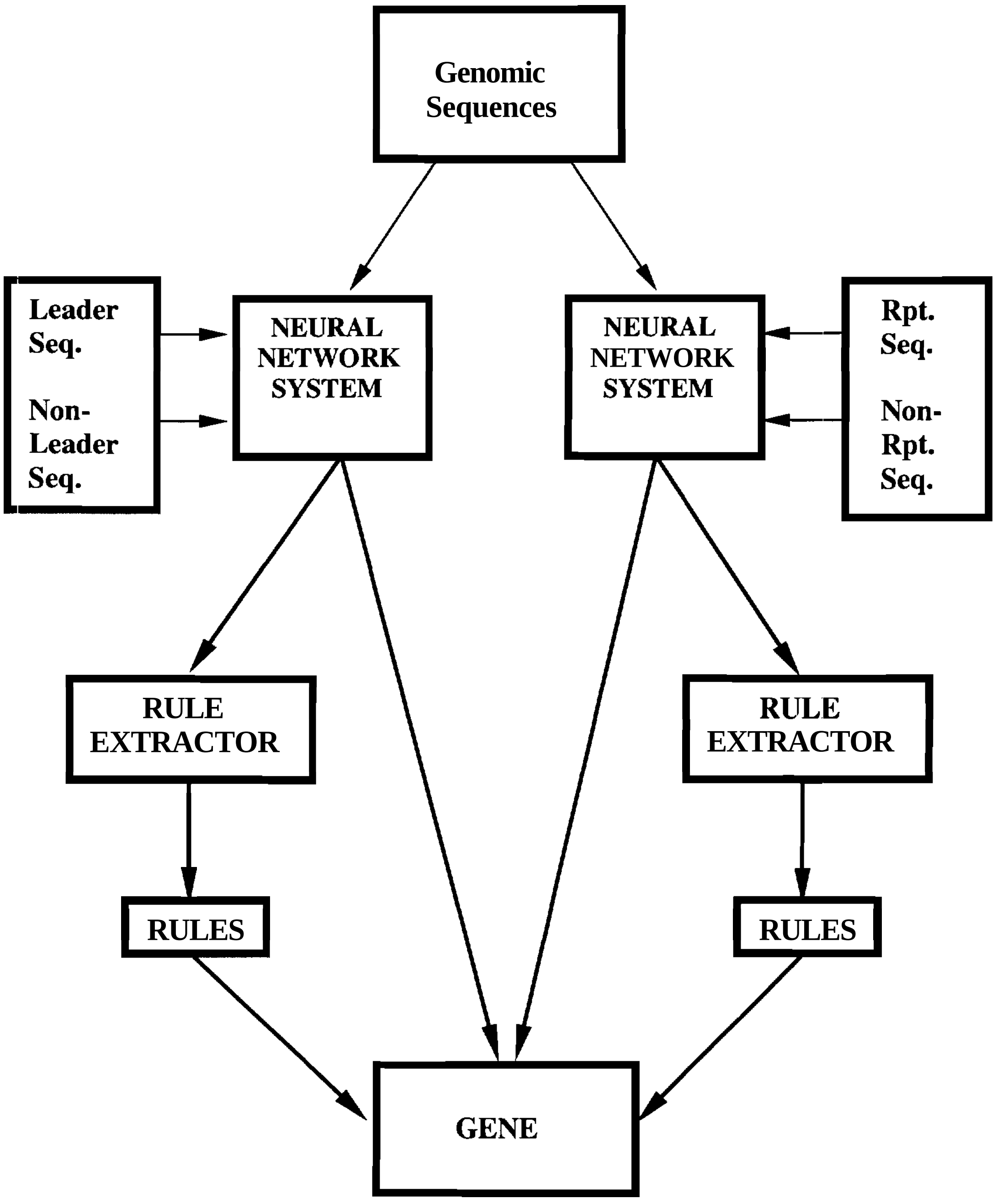


Fig.1.1 Framework for detection of classes of sequences by neural networks and subsequent rule extraction

2. BACKGROUND

2.1 :NeuralNetworks

In this section we describe the basic concepts of neural networks, neural network architecture, and learning algorithms.

2.1.1 Basic concepts

A neural network is an information-processing device consisting of a large number of highly interconnected processing elements to be called *units* (neurons). Fig. 2.1 shows a general version of a unit. Each unit *i* performs only very simple computations resulting in a single activation-value $a_i(t)$. The activation value is transformed into an output signal $o_i(t) = f_i(a_i(t))$ using an output function f_i. This function is sometimes the identity function

$$o_i(t) = a_i(t) \qquad (2.1)$$

or a step function $o_i(t) = 1$, if $a_i(t) > \theta_i$, $o_i(t) = 0$ else, with threshold θ_i for each unit. The output signals are propagated on weighted interconnections between the processing elernents called links. The signals going into one unit are combined into a net-input value $net_i(t)$ using a propagation rule. This rule normally calculates the weighted sum of the output signals leading into unit *i* as

$$net_i(t) = \Sigma_j \, w_{ji} o_j(t) \qquad (2.2)$$

where w_{ji} is the weight on the link from unit j to unit i

2.1.2 Multilayer neural networks

Although a single neuron can perform certain simple functions, the power of neural computation comes from connecting neurons into networks. The simplest network is a group of neurons arranged in a layer. Larger, more complex networlts generally offer computational capabilities. Multilayer neural networks have been proven to have capabilities beyond those of a single layer. Multilayer networks may be formed by simply cascading a group of single layers. The output of one layer provides the input to the subsequent layer. The layers between the input layer and the output layer is called the hidden layers.

2.1.3 Backpropagation

Several neural network models have been proposed since Rosenblatt [1] introduced the perceptron in 1958. The perceptron is a two-layer neural network which has the ability to learn and recognize simple pattern. Rosenblatt proved that if the input data were linearly separable, the training procedure of the perceptron would converge and the perceptron could discriminate the data. However, the input data. are usually not separable, and the decision boundaries may oscillate continuously when the perceptron algorithm is applied. A modification of the perceptron is a two-layer delta rule.

The delta rule, developed by Widrow and Hoff [2] in the early 1960's, is a supervised training approach in which error correction is done with a least mean square algorithm (LMS). The delta rule is so named because it changes weights in proportion to the difference between actual and desired output responses. The delta rule neural network has two layers and can be used to discriminate linearly separable data. The delta rule has been extended to include three or more layers. The extension is called feedforward multilayer network (FMLN). By applying neural network with three or more layers, arbitrarily shaped decision regions can be formed.

In contrast to the delta rule, the backpropagation algorithm **[3]** with which a FMLN is trained is a neural network algorithm which can be used to 'discriminate data that are not linearly separable. But a problem with backpropagation is that its training process can be computationally very complex and convergence may be slow because the training process is iterative. This is a serious drawback, especially when the dimensionality of the data is very high.

Rumelhart ***et*** al. [3] added a momentum term to the backpropagation algorithm in order to speed up training. Adding a momentum term has the advantage:that it filters out high frequency variations in the weight space. The momentum term causes an upper bound on how large an adjustment can be made to a weight. However, the sign of the momentum term may also cause a weight to be adjusted up the gradient of the error surface, instead of down the gradient as desired.

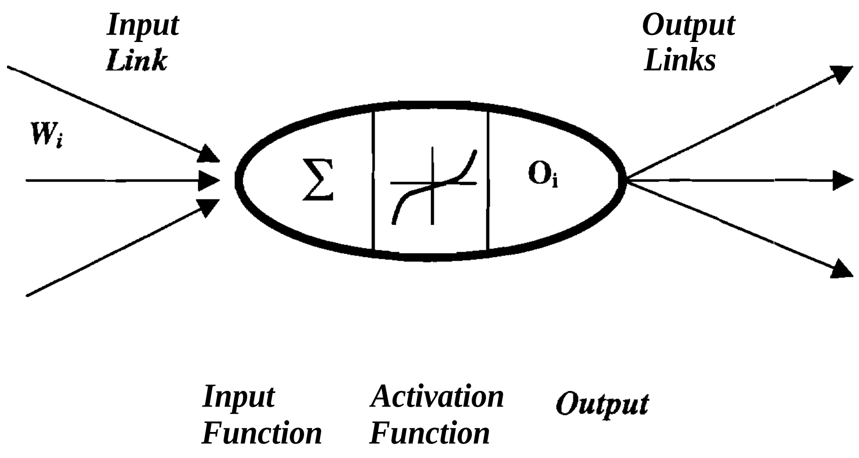


Fig.2.1 A general model of an artificial neuron.

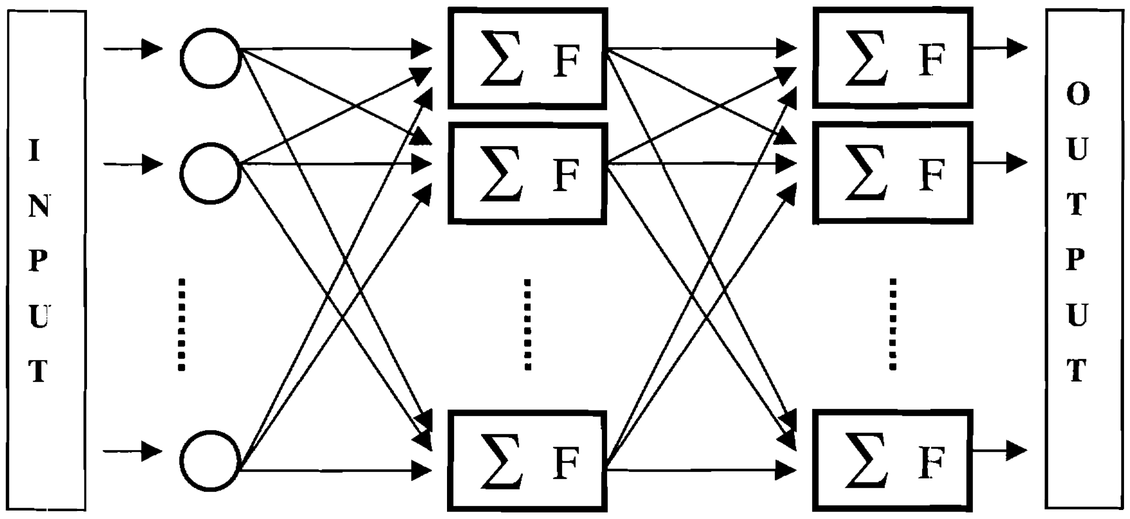


Fig.2.2 **A** neural network with a hidden layer.

2.2 Human Genome Project

Although the Human Genome Project (HGP) is well recognizecl as the first "big science" project in biology [5], it is less well known as a major project in computer technology and information management. The HGP is an international undertaking with the goal of obtaining a fully connected genetic and physical map of the human chromosomes and defining complete nucleotide sequence of human DNA. The HGP can be considered to be the creation of the most amazing data containing instructions for building people.

Although the computational challenges associated with the project have been described, some engineers have expressed concerns about its complexity because the HGP is rapidly becoming a data-rich area with extensive computaitional needs [6]. Engineers with a variety of research interests are looking for ways to address those needs. The sheer volume of data poses a serious challenge in storing and retrieving biological information, and the rate of growth is exponential. For example, the volume of DNA and protein sequence data is currently doubling every 22 months. This growth is driven partly by :increased interest in genetic studies and partly by technological developments that have reduced costs of gathering data. Moreover, as our understanding of fundamental biological processes increases, so does the need to store and access data that go beyond DNA and protein sequences-for instances, mapping data from the analysis of gene positions on chromosomes. Linking the heterogeneous data libraries of HGP, organizing its diverse and interrelated data sets, and developing effective query options for its databases are all areas for cross-fertilization between HGP and engineering. However, even the apparently simple task of analyzing a single sequence of DNA requires complex collaboration. We believe that there are no monolithic solutions to the computational challenges of the work being done. By addressing the particular biological considerations of a. problem, engineers with a variety of skills can make important contributions.

2.3 .Applicationsof Neural Networks in HGP [6]

In the area of HGP, researchers in various areas are taking the first steps toward knowing the functions and locations of all the genes and regulatory sites in the genomes of several organisms. As these researchers determine the nucleotide sequences of large stretches of human or other DNA, they are producing great volumes of sequence data. Direct laboratory analysis of this data is difficult and expensive, making computational techniques essential. But the variation, complexity, and incompletely understood nature of genes make it impractical to hand-code the algorithms. Several researchers are exploring the increase in the diversity and efficiency of mathematical and informatics methods to predict and analyze functional sites on nucleic acids. For problems related to well defined sites, like those for the recognition of restriction enzymes, the technique of constructing consensus sequence patterns and comparing them automatically with newly sequenced DNA provides a reasonable solution. A further avenue for the development of more sophisticated pattern recognition tools will be opened by the application of neural networks. Beyond the obvious advantages that such tools are much easier to develop than statistical and/or rule based methods and that once the networks have been trained, neural networks can be easily distributed and employed to search on new targets. We may also expect that neural networks with a complex architecture will make it possible to learn higher order correlations between sequences and functions than it is possible with traditional methods. In fact, there is extensive genetic evidence for context-dependent and/or compensatory mutations in regulatory genomic sequences. Their adequate analysis and understanding requires methods that accept them in their entirety and make use all of their information, not only position by position, but also cross-correlating every position with every other one.

The most important DNA pattern recognition problems, to which several neural networks studies were devoted in past years, include the distinction of ribosome binding sites, the analysis of promoter recognition, and the differentiation between coding and non-coding (exon/intron) regions in eucaryotic genomes. Reczko and Suhai *[6]*provided an in-depth review of applications of neural networks in this area.

The first application of an artificial neural networks for a DNA sequence analysis problem was the use of the perceptron algorithm to predict ribosome binding sites on mRNAs by Stormo et al. [7].

Lukashin et al. [8] tackled the analysis of promoter regions in E. coli by using a specific neural architecture consisting of separated blocks of three-layer networks that were trained independently. Demeler and Zhou [9] employed a simple backpropagating network for E.coli promoter prediction. They paid special attention to the design of training and test sets, to different forms of input representation, to the number of hidden units, and to the error level to be achieved during training.

Stormo et al. [7] applied first the perceptron algorithm to the identification of coding regions. Although they obtained good training results, the reliability of their networks was too low for sequences outside of the training set. Kudo et al. [10] constructed a finite state automation as a recognizer of 5' splice sites but arrived only at 50-55% prediction accuracy on test genes. Nakata et al. [11] combined the output of two two-layer perceptrons with physicochemical and base composition properties in the framework of a discriminant analysis. Brunak et al. [12] constructed the first backpropagation neural network to realize a joint scheme where prediction of transition regions between exons and introns regulates a cut-off level for splice:-site assignment. Uberbacher and Mural [13] designed a 'coding recognition module' consisting of a backpropagation neural network whose seven input nodes received their information frorn seven specific 'sensors'. A significant increase in the prediction accuracy could be achieved for the exon/intron localization problem by Farber et al. [14] who made use of the 'mutual information' of spatially separated codons in exons and in introns.

One of the most prominent applications of neural networks trained with the backpropagation algorithm is the prediction of secondary structures using only a local context of the amino acid sequences done by Qian and Sejnowski [15]. The secondary structure prediction networks developed by Bohr et al. [16] were also used to define a homology measure between sequences. A major improvement was achieved by Rost et al. [17] using a primary sequence representation that contains much more evolutionary information than the bare sequence.

Hirst and Sternberg [18] compared the performance of a two-layer perceptron with a statistical method for the prediction of ATP/GTP-binding motifs in protein sequences. A successful application of an unsupervised learning algorithm uses the self-organizing Kohonen feature map for clustering proteins into families based on their sequence similarity [19].

There are many successful applications of neural networks for genome research with performances superior to comparable statistical methods. It has to be emphasized that the theory for simple neural network architecture such as the perceptron is very well developed and has shown that these models can be viewed as variants of statistical classifiers or approximators. The most severe problem of all these supervised learning methods is overfitting which is not well understood in some applications where the real performance on test set is not emphasized sufficiently. More advanced networks such as recurrent networks or complex hybrid of different neural networks modules have no comparable statistical model as counterparts and are thus tools to solve new classes of problems that could not be tackled successfully previously. With the increase of neurobiological knowledge this situation will shift much more in the direction where more and more complex and realistic models of parts of biological neural networks may be built and applied to difficult pattern processing tasks. These models may only be coarsely analyzed using complex system theory. Yet these new forms of information processing algorithm most probably will be shown to be of enormous use as a first step in solving extremely hard problems in molecular biology.

2.4 Input Representations of Genomic Sequences for Neural Networks

In the context of genomic sequences, one starts by providing a suitable encoding of the symbols "A," "T," "G," and "C" into a string of bits, i.e., 0's and 1's. For example, each symbol can be represented by four bits, and these four bits/symbol are concatenated to provide a bit string representing a given length of genomic sequence (see Fig.2.3). If a codon representation instead of a base representation were used, each symbol (codon) would be represented by sixty-four bits. The line of circle represents input neurons, which take on values of 0 or 1. The output neuron state is determined by the input neurons' states according to a set of connection weights. After training is complete, it should attain a value near 1.0 if the input example comes from the "true" class; near 0.0 otherwise.

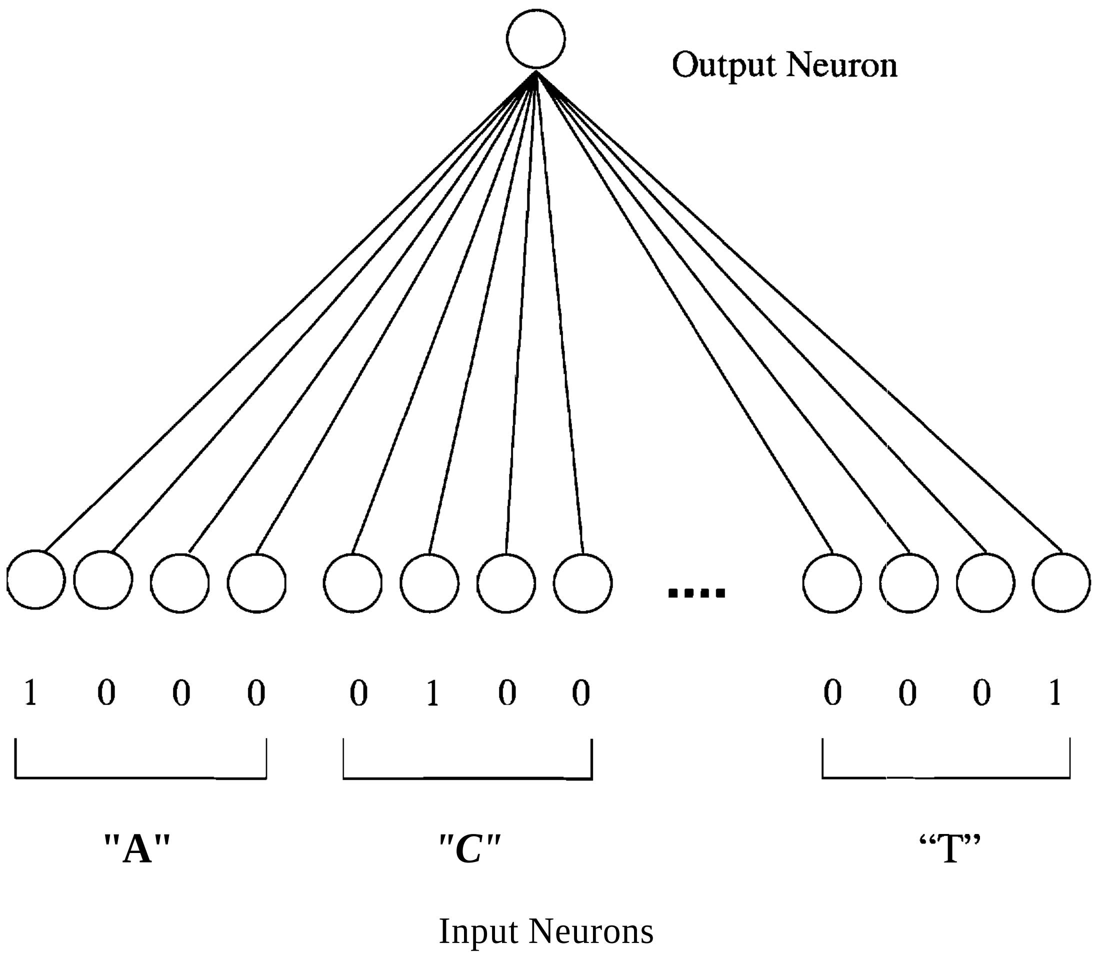


Fig.2.3 Input representation of genomic sequences for neural networks

2.5 Data Mining

Data mining is a problem-solving methodology that finds a logical or mathematical description, eventually of a complex nature, of patterns and regularities in a set of data. As laboratories around the world produce ever-greater volumes of data, fast and efficient computational analysis techniques are becoming essential. The development of the science of statistics has produced a number of methods for data analysis, and statistical methods have been widely applied to analyzing data with the goal of finding con-elations and dependencies among sets of data. These data, however, might be underutilized. That is, there is a potential wealth of information dormant in the sets of data which can show patterns and rules. The key issue is the use of new techniques to extract new, potentially useful information from the huge amounts of data that have been produced in modern times. Data mining reflects the rapid transition from the information age to the knowledge age and is a rapidly growing area.

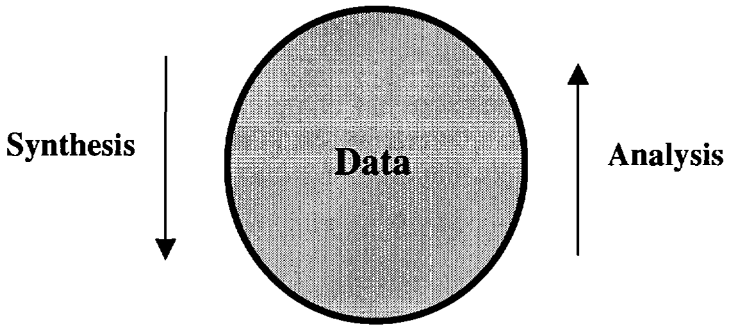


Fig. 2.4 Representation of data mining as a dual process

There are two basic ways of performing data mining and data analysis. The first makes use of supervised learning, and the other is based on unsupervised learning. In supervised learning, patterns are found exploring a number of known cases that show or imply well defined patterns: based on those cases, generalizations are formed. With unsupervised learning, data patterns are found starting from some logical characterization of the regularities. We can represent data mining as a dual process that can be either synthetisizing, generalizing from known cases, or analytic, expanding some high-level implicit description, as depicted in Fig. **2.4.**

Supervised and unsupervised learning have other meanings in other contexts. It is therefore appropriate to remark that, in the context of data mining, machine learning is a set of techniques that perform two fundamental tasks:

1. they generalize from a set of known examples;
2. they detail structure from its descriptions.

Data mining allows to produce scientific knowledge in areas that are not yet suitable for the traditional approach of mathematical physics. Learning and optimization algorithms can be used to produce optimal modeling of experimental data in the absence of previous theoretical explanations. Knowledge thus acquired has been applied to a number of different domains and used for engineering purposes. Data mining is a building block of these engineering methods as it provides basic knowledge on which exploratory search and design optimization can be based (Fig. **2.5).**

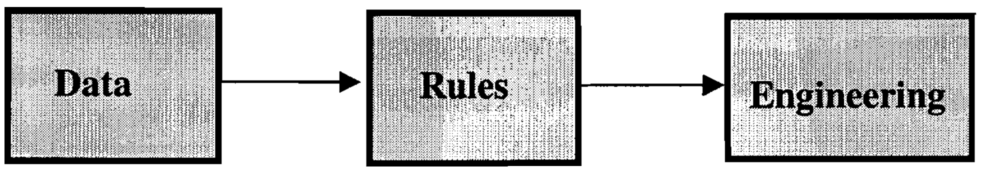


Fig. **2.5** Data leading to rules used in engineering.

2.6 Decision Trees [40]

A decision tree is a series of decisions to determine the class label for a given example. It consists of many nodes and leaves. At each internal node there is a test, and a branch corresponding to each of the possible outcomes of the test, and at each leaf node there is a class label. Decision trees have been used for classification and other tasks since the 1960s [59]. In the 1980's, Breiman et al.'s book on CART [49] and Quinlan's work on ID3 [60] paved the way for this research arena. Some decision tree algorithms also involve other machine learning paradigms such as neural networks [50].

The advent of major scientific projects such as the human genome project has resulted in enormous amounts of data on a daily basis, and fast, efficient, automated machine learning techniques are becoming essential. These techniques analyze, filter and classify those data before presenting them in easy form to end users. Decision trees are a particularly useful tool in this context because they perform classification by a sequence of simple, easy-to-understand tests whose meaning is intuitively clear to end users.

The algorithm used here for inducing decision trees is referred to as top-down induction of decision trees and is based on the following steps [60].

1. Divide a set of examples called the training set S into two or more subsets using a test on one or more features. If every subset from S is pure(all examples in the subset are from one class), then stop.
2. If not, investigate all tests that split S into subsets. Rank each test according to how well it divides the examples.
3. Select the test that ranks first.
4. Do this operation recursively on each subset.

The result of this algorithm is a decision tree with test nodes and leaf nodes. The leaf nocles contain class labels.

2.6.1 Selection of a test

How decision trees choose a test is crucial, and we use the information gain splitting rule for this purpose [59]. This is an information theoretic metric that chooses the feature to split on, based on the gain in information obtained by that split and measured in bits. If S is the set of training data, and C_j is a class in the set, the expected information for deciding a given class in S is given by

$$Info(S) = -\sum_{j=1}^{K} \log_2\left(\frac{|C_j,S|}{|S|}\right)\cdot\frac{|C_j,S|}{|S|} \qquad (5.7)$$

where K is the number of classes, $|C_j,S|$ is the number of examples of class C_j in S, and $|S|$ is the number of examples in the set. From information theory, info(S) is the same as the estimate of entropy. When a feature, X, with possible t values, has been chosen as a test feature, then the expected information for deciding a class under the test is:

$$info_X(S) = \sum_{i=1}^{t}\frac{|S_i|}{|S|}\cdot info(S_i) \qquad (5.8)$$

where S_i is the subset of S with examples of value i for feature X , $info_X(S)$ is the same as the estimate of conditional entropy of information theory. The infonnation gain is the difference between the expected information for deciding a class with and without the tesl: on feature X:

$$gain(S) = info(S) - info_X(S) \qquad (5.9)$$

where gain(S) is the same as the estimate of mutual information of information theory.

Thus, the feature producing the highest information gain is chosen as the current split.

Consider the training set of Table as a concrete description. There are two classes, ten examples belonging to non-leader and five to leader, so

$$Info(S) = -10/15 \times \log_2(10/15) - 5/15 \times \log_2(5/15) = 0.9182 \text{ bits}$$

After using A1 (the first input of a neural network) to divide S into four subsets, the results is given by

$$\begin{aligned} Info_X(S) = & \\ & 8/15 \times (-3/8 \times \log_2(3/8) - 5/8 \times \log_2(5/8)) \\ & + 3/15 \times (-1/3 \times \log_2(1/3) - 2/3 \times \log_2(2/3)) \\ & + 2/15 \times (-2/2 \times \log_2(2/2)) \\ & + 2/15 \times (-1/2 \times \log_2(1/2) - 1/2 \times \log_2(1/2)) \\ & = 0.6976 \text{ bits} \end{aligned}$$

$$gain(S) = 0.9182 - 0.6976 = 0.2206 \text{ bits}$$

We can test other attributes. Suppose that we had divided it on the attribute A62. This would have given four subsets, and similar computation is

$$\begin{aligned} Info_X(S) = & \\ & 3/15 \times (-1/3 \times \log_2(1/3) - 2/3 \times \log_2(2/3)) \\ & + 4/15 \times (-4/4 \times \log_2(4/4)) \\ & + 6/15 \times (-3/6 \times \log_2(3/6) - 3/6 \times \log_2(3/6)) \\ & + 2/15 \times (-1/2 \times \log_2(1/2) - 1/2 \times \log_2(1/2)) \end{aligned}$$

$$gain(S) = 0.9182 - 0.5886 = 0.3296 \text{ bits}$$

This gain is more than the gain from the previous test. The gain criterion would then prefer the test on A62 over the first test on A1. The feature which has maximum gain is chosen in this way. For each feature value A,C,G,T of the feature, the same procedure is repeated until pure class is obtained.

Table 2.1 An example of training set.

A1	A2	A3	A4	A5	....	A62	A63	A64	Class
A	C	G	G	C	....	G	C	T	Leader
T	T	G	C	C	....	C	T	G	Non-leader
A	C	T	A	C	....	G	C	T	Leader
C	C	G	G	C	....	A	T	A	Non-leader
A	A	A	T	C	....	C	C	T	Non-leader
A	T	G	G	T	....	T	C	A	Non-leader
G	C	G	T	C	....	G	A	T	Non-leader
C	C	C	G	T	....	A	C	C	Leader
T	C	G	T	C	....	G	C	T	Leader
A	T	C	C	C	....	G	T	C	Non-leader
A	C	T	A	C	....	T	C	T	Leader
C	T	G	G	C	....	G	A	A	Non-leader
A	C	A	T	C	....	C	A	T	Non-leader
A	A	G	G	T	....	C	C	A	Non-leader
G	G	G	T	C	....	A	A	T	Non-leader

2.6.2 Tests on continuous attributes

It might seem that tests on continuous attributes would be difficult to formulate, since they contain arbitrary thresholds. This is not so: The method for finding such tests can be described very closely. This algorithm for finding such tests can be described very concisely. This algorithm for finding appropriate thresholds against which to compare the values of continuous attributes appears in Breiman *et* al. [49].

The training cases S are first sorted on the values of the attribute **A** being considered. There are only a finite number of these values, so let us denote them in order as { x_1, x_2, ..., x_m }. Any threshold value lying between x_i and x_{i+1} will have the same effect of dividing the cases into those whose value of the attribute A lies in { x_1, x_2, ..., x_i } and those whose value is in { x_{i+1}, x_{i+2}, ..., x_m }. There are thus only m-1 possible splits on A, all of which are examined. It is usual to choose the midpoint of each interval as the representative threshold.

3. DETECTION OF RARE EVENT BY NEURAL NETWORKS

3.1 Introduction

Training neural networks to recognize events which occur with low probability is significant in many applications. This is a difficult and challenging problem. When we investigated the use of neural networks to identify regulatory regions in human genomic sequences, we realized there are rare events in the sequences. That is, Alu regions which is a kind of repetitive DNA sequences, represent small portions of entire human DNA sequences, most being non-Alu regions. This results in the shortage of examples. We propose two schemes to solve these problems by neural networks and sample stratification. The experimental performance of the two schemes using human DNA as well as two other data sets indicates that proposed schemes have the potential of significantly improving accuracy of neural networks to recognize rare events.

3.2 Importance Sampling

Previous work in digital communication systems has shown the potential of dramatically lowering the computational burden of simulations by utilizing importance sampling [20]. Importance sampling is a technique that can be applied to the simulations of low probability events without incurring the computational costs usually associated with such simulations. With importance sampling, one can modify the probability distribution of the underlying random process in order to make the rare events occur more frequently. In order to compensate for this modification, each event is weighted by a factor that is a function of only the state of the input and independent of the process itself. Applying the basic idea behind importance sampling technique to neural networks, Monro et al. [21] developed a likelihood ratio weighting function (LRWF) which leads to learning with weighted least squares. This weighting function allows neural networks to be trained utilizing a data set in which the events occur with high probability, but also successfully classify data during testing in which the events occur with much lower probability. This also leads to the reduction of computational burden associated with training neural networks in order to recognize low probability events.

The algorithms presented next in this chapter are more concerned with highly accurate detection of rare events.

3.3 Sample Stratification

Sample stratification is a technique for making each class in a sample have equal influence on decision making. For classification with neural networks, it is often preferable to use a stratified sample including an equal number of examples from each class. However, it is not always possible to have a stratified sarnple because of unavailability of examples (referred to as rare event cases). For rare event detection, we develop two sample stratification schemes. The first scheme stratifies a sample by adding up the weighted sum of the derivatives during the backward pass of training. The second scheme uses bootstrap aggregating. After training neural networks with multiple sets of bootstrapped examples of the rare event classes and subsampled examples of common event classes, we perform multiple voting for classification. These two schemes make rare event classes have a better chance of being represented in the sample used for training neural networks and thus improve the classification accuracy for rare event detection.

If a sample is drawn such that every example in the population occurs according to its probability of occurrence, then the sample is termed *representative.* It is often preferable to use a representative sample. However, for classification in which the problem is assigning an example to a class or category, it is better to include an equal number of examples from each class during training, although the probability of occurrence is different from class to class. In that case, the sample is not representative, and is termed ***stratified.*** With a stratified sample, examples from small classes have a better chance of being included than those from large classes.

Consider, for example, a sample containing sequences of type A (5%) and non-type A (95%), and suppose this sample contains 100 examples. If a representative sample is drawn, non-type A examples will most likely occur 95 times, and type A examples will most likely occur 5 times. On the other hand, the sample can be stratified so as to have 50 examples of each class. Which sample will produce the best detection accuracy for the rare events? We anticipate that the stratified sample will produce the best accuracy. Increasing the number of examples of type A from 5 to 50 produces a big improvement in accuracy of classification of type A, whereas decreasing the number of examples of non-type A from 95 to 50 produces only a small decline in accuracy of classification of the non-type A. Thus, the stratified sample is better because the improvement on the rare events is greater than the loss of accuracy on the common events, even when the test is made on a representative sample.

Next, suppose that sample size in the data set is increased from 100 to 1,000. We know that increasing sample size makes the model more accurate. The problem is that we cannot maintain equal numbers of examples for both events: we woulld need 500 type A's., but we have only 50. In this situation, we can just use a sample that includes all of the examples of type A (rare events), and non-type (common events) of which we have many examples. This produces a sample with 50 type A and 950 non-type A. Then there are 19 times as many non-type A as type A in the sample. This may cause a problem in training neural networks because the data is unbalanced. To alleviate this, it is necessary to give the type A's 19 times as much weight as the non-type A's.

3.4 Stratifying Coefficients and Derivation of a Modified Hackpropagation Algorithm

We can make a stratified sample by modifying the backward pass through neural networks using backpropagation, where we accumulate the derivatives of the error with respect to each weight. During the backward pass, as we accumulate the derivatives, we add up not the sum of the derivatives but their weighted sum. For example, in the previous case, when the example is a type A, we add 19 times the denivative, but when the example is a non-type A, we add just the derivative. At the end of tht: epoch, when we change the weights, each type A will have 19 times as much impact as each non-type A.

We derive a modified form of the backpropagation algorithm which includes a term to be called stratifying coefficient (SC), and to be denoted by c(x). It is included in the computation of the error terms associated with the final output layer of the neural network. The SC is similar to LRWF, but the basic idea is different. SC adds more weight to the rare event to make a sample stratified while LRWF gives less weight to the rare event to make a sample representative.

Consider a specific weighted error, E_p, due to the presentation of the input vector x_p as

$$E_p = \frac{1}{2}\sum_j [D_{pj} - Z^N_{pj}(x_p, w)]^2 \, c(x_p) \tag{3.1}$$

where D_{pj} is the jth component of the desired output vector due to the presentation of input vector x_p. The output of node j of the output layer, which is the Nth layer, is denoted as $Z^N_{pj}(x_p, w)$. The SC, $c(x_p)$ evaluated at the present input vector, equals the ratio of the probability of the class of x_p to the probability of the rare event. The dependence of Z^N_{pj}

on the present input vector x_p and the weights denoted by w will be suppressed in the following notation.

The output of node j in the m^th layer due to the presentation of the input vector x_p is defined as

$$Z_{pj}^{m} = f\,(Y_{pj}^{m}) \tag{3.2}$$

where f(.) is a continuously differentiable, non-decreasing, nonlinear activation function such as a sigmoid. Furthermore, the input to node j of the m^th layer due to the presentation of the input vector x_p is defined as

$$Y_{pj}^{m} = \sum w_{ji}^{m}\, Z_{pi}^{m-1} \tag{3.3}$$

where w^m denotes the weight matrix between the m^th and the (m-1)^th layer of the networks.

The backpropagation algorithm applies a correction $A\,w_{ji}$ to synaptic weight w_{ji}, which is proportional to the instantaneous gradient $\frac{\partial E_p}{\partial w_{pj}^{m}}$. According to the chain rule, we may express this gradient as follows:

$$\frac{\partial E_p}{\partial w_{pj}^{m}} = \frac{\partial E_p}{\partial Y_{pj}^{m}} \frac{\partial Y_{pj}^{m}}{\partial w_{pj}^{m}} \tag{3.4}$$

$$\frac{\partial Y_{pj}^{m}}{\partial w_{pj}^{m}} = Z_{pj}^{m} \tag{3.5}$$

The negative of the gradient vector components of the error E_p with respect to Y^m_{pj} are given by

$$\delta^m_{pj} = -\frac{\partial E_p}{\partial Y^m_{pj}} \tag{3.6}$$

Applying the chain rule allows this partial derivative to be written as

$$\delta^m_{pj} = -\frac{\partial E_p}{\partial Y^m_{pj}} = -\frac{\partial E_p}{\partial Z^m_{pj}} \frac{\partial Z^m_{pj}}{\partial Y^m_{pj}} \tag{3.7}$$

The second factor can be easily computed from **Eq. (3.3)** as

$$\frac{\partial Z^m_{pj}}{\partial Y^m_{pj}} = f'(Y^m_{pj}) \tag{3.8}$$

which is simply the first derivative of the activation function evaluated at the present input to that particular node.

In order to compute the first term, consider two cases. The first case is when the error signal is developed at the output layer N. This can be computed from **Eq. (3.1)** as

$$\frac{\partial E_p}{\partial Z^m_{pj}} = -[D_{pj} - Z^N_{pj}]\ c(x_p) \tag{3.9}$$

Substituting Eqs. **(3.8)** and **(3.9)** into **Eq. (3.7)** yields

$$\delta_{pj}^{N} = [D_{pj} - Z_{pj}^{N}]\,c(x_p)\, f'(Y_{pj}^{N}) \qquad (3.10)$$

For the second case, when computing the error terms for some layer other than the output layer, the δ_{pj}'s can be computed recursively from those associated with the output layer as

$$\begin{aligned}
\frac{\partial E_p}{\partial Z_{pj}^{m}} &= \sum_k \left[\frac{\partial E_p}{\partial Y_{pk}^{m+1}} \frac{\partial Y_{pk}^{m+1}}{\partial Z_{pj}^{m}}\right] \\
&= \sum_k \left[\frac{\partial E_p}{\partial Y_{pk}^{m+1}} \frac{\partial}{\partial Z_{pj}^{m}} \sum_i w_{ki}^{m+1} Z_{pi}^{m}\right] \\
&= \sum_k \frac{\partial E_p}{\partial Y_{pk}^{m+1}} w_{kj}^{m+1} \\
&= -\sum_k \delta_{pk}^{m+1} w_{kj}^{m+1} \qquad (3.11)
\end{aligned}$$

Combining this result with Eq. (3.6) gives

$$\delta_{pj}^{m} = f'(Y_{pj}^{m}) \sum_k \delta_{pk}^{m+1} w_{kj}^{m+1} \qquad (3.12)$$

These results can be summarized in three equations. First, an input vector x_p is propagated through the network until an output is computed for each of the output nodes of the output layer. These values are denoted as Y^{N}_{pj}. Next, the error terms associated with the output layer are computed by Eq. (3.10). The error terms associated with each of the other m-1 layers of the networks are computed by Eq. (3.12). Finally, the weights are updated as

$$\Delta_p w_{ji}^{m} = \eta\, \delta_{pj}^{m} Z_{pi}^{m-1} \qquad (3.13)$$

where η represents the learning rate of the networks. Usually η is chosen to be some nominal value such as 0.01.

As compared to the regular backpropagation algorithm [3], the only change is the inclusion of the stratifying coefficient in Eq. (3.10). All other steps of the backpropagation algorithm remain the same.

3.4.1 Approximation of a posteriori probabilities

Neural networks can provide outputs to approximate a posteriori probabilities that can be used for higher level decision making. Conventional statistical methods like Parzen density estimation can also be used for this purpose, but these methods are are less reliable with high dimensional inputs.

We can estimate how the networks with the stratifying coefficient scheme approximate a posteriori probabilities. We will assume a 2-class problem, but the results can be simply generalized to more number of classes. With a squared-enor cost function and without stratifying coefficient, the network parameters are chosen to minimize the following cost function:

$$E_a = \int \sum_{j=1}^{2} \left\{ \sum_{i=1}^{2} [Z_i(X) - D_i]^2 \right\} f(X, C_j)\, dx \qquad (3.14)$$

The above equation represents a sum of squared errors, with two errors appearing for each input-class pair. For a particular pair of input X and class C_j, each error, $Z_i(X) - D_i$ is simply the difference of the actual network output $Z_i(X)$ and the corresponding desired output D_i. The two errors are squared, summed, and weighted by the joint probability f (X,C_j) of the particular input-class pair.

Substituting $f(X, C_j) = f(C_j \mid X) f(X)$ in (14) yields

$$E_a = \int \left[\sum_{j=1}^{2} \sum_{i=1}^{2} [Z_i(X) - D_i]^2 f(C_j \mid X) \right] f(X)\, dx \qquad (3.15)$$

In the backpropagation algorithm with stratifying coefficient, the new weighted error function to be minimized is given by

$$E_a' = \int \left[\sum_{j=1}^{2} \sum_{i=1}^{2} [Z_i(X) - D_i]^2 \, c(X) \cdot f(C_j \mid X) \right] f(X)\, dx \qquad (3.16)$$

$$= \int \left[\sum_{j=1}^{2} \sum_{i=1}^{2} [Z_i(X) - D_i]^2 f(C_j \mid X) \right] c(X) \cdot f(X)\, dx \qquad (3.17)$$

This result can be written as

$$E_a' = \int \left[\sum_{j=1}^{2} \sum_{i=1}^{2} [Z_i(X) - D_i]^2 f(C_j \mid X) \right] \cdot f^*(X)\, dx \qquad (3.18)$$

where $f^*(X) = c(X) f(X)$ (3.19)

Expanding the bracketed expression in Eq. (3.18) yields

$$E_a' = \int \left\{ \sum_{j=1}^{2} \sum_{i=1}^{2} \left[Z_i^2(X) f(C_j \mid X) - 2 Z_j(X) D_i f(C_j \mid X) + D_i^2 f(C_j \mid X) \right] \right\}$$
$$\bullet f^*(X)\ dx$$

(3.20)

Exploiting the fact that $Z_i^2(X)$ is a function only of X and $\sum_{j=1}^{2} f(C_j \mid X) = 1$ allows Eq. *(3.20)* to be expressed as

$$E_a' = \int \left\{ \sum_{i=1}^{2} \left[Z_i^2(X) - 2 Z_i(X) \sum_{j=1}^{2} D_i f(C_j \mid X) + \sum_{j=1}^{2} D_i^2 f(C_j \mid X) \right] \right\} f^*(X)\ dx$$

(3.21)

For a two-class problem, D_i equals 1 if input X belongs to class C_i and 0 otherwise. Therefore, $\sum_{j=1}^{2} D_i f(C_j \mid X) = f(C_i \mid X)$, and

$$E_a' = \int \left\{ \sum_{i=1}^{2} \left[Z_i^2(X) - 2 Z_i(X) f(C_i \mid X) + f(C_i \mid X) \right] \right\} f^*(X)\ dx \qquad (3.22)$$

Addling and subtracting $\sum_{i=1}^{2} f^2(C_i \mid \mathrm{X})$ in Eq. *(3.22)* allows it to be written in the following form:

$$E_a' = \int \left\{ \sum_{i=1}^{2} \left[Z_i^2(X) - 2 Z_i(X) f(C_i \mid X) + f^2(C_i \mid X) + f(C_i \mid X) - f^2(C_i \mid X) \right] \right\}$$
$$\bullet f^*(X)\ dx$$

(3.23)

$$= \int \left\{ \sum_{i=1}^{2} \left[Z_i(X) - f(C_i \mid X) \right]^2 + \sum_{i=1}^{2} \left[f(C_i \mid X) - f^2(C_i \mid X) \right] \right\} f^*(X)\ dx$$

(3.24)

$$= \int \left\{ \sum_{i=1}^{2} [Z_i(X) - f(C_i \mid X)]^2 \right\} f^*(X)\, dx + \int \left\{ \sum_{i=1}^{2} \left[f(C_i \mid X) - f^2(C_i \mid X) \right] \right\} f^*(X)\, dx \tag{3.25}$$

Since the second term in Eq. **(3.25)** is independent of the networks outputs, minimization of E_a' is achieved by choosing network parameters to minimize the first term.

Eq. **(3.25)** can be interpreted as

$$E_a' = E_{X^*} \left\{ \sum_{i=1}^{2} [Z_i(X) - f(C_i \mid X)]^2 \right\} + \Delta \tag{3.26}$$

where E_{X^*} [.] is the expected value with respect to the modified distribution $f^*(X)$. Minimization of E,' is achieved by minimizing the first term in Eq. **(3.26).** This means the neural network outputs approximate a posteriori probabilities based on the modified distribution when E_a' is minimized. This shows the mean square error approximation is computed as if the input vector X were drawn from $f^*(X)$ rather than $f(X)$. This result shows that we can approximate a posteriori probabilities of rare events more accurately by using stratifying coefficients. This conclusion will be observed to coincide with experimental results discussed in Section **3.7.**

3.5 :Bootstrap Stratification

We propose an alternative approach to sample stratification: we make smaller sized sets of examples from the entire data. Every set includes all the examples of bootstrapped examples of the smaller class and subsampled examples of the larger class. With these sets, we train a set of neural networks. Since there are an equal number of occurrences of every event, the neural networks spend as much time in learning about the rare events as about the common events.

This approach comes out of three basic facts: First, bootstrap methods are extremely valuable in situations where data sizes are too small to invoke good results. Second, a classifier trained with subsampled data does not degrade much compared with the one trained with complete data. Third, aggregating can improve performance of a classifier.

3.5.1 Bootstrap procedures

In general, the bootstrap is a technique for resampling the given data in order to induce information about the sampling distribution of a classifier [24], [25]. This generates multiple copies of a classifier. Aggregation averages over the copies when predicting a numerical outcome and does a multiple vote when predicting a class. The multiple copies are made by building bootstrap replicates of the learning set and using these as new learning sets. The method can be quite effective, especially, for an "unstable" learning algorithm for which a small change in the data effects a large change in the computed hypothesis.

Consider a given set of N examples, each belonging to one of M classes, and a classifier for a training set of examples. Bootstrap procedures construct multiple classifiers from the examples. The classifier trained on trial k will be denoted as C_k while Cs is the consensus classifier. C_k (x) and $C_S(x)$ are the classes decided by C_k and C_S .

For each trial k = 1,2... K, the training set of size N is sampled from the original examples. This training set is the same size as the original data, but sorne examples may not appear in it while others appear more than once. A classifier C_k is generated from the sample, and the final classifier C_S formed by aggregating the K classifiers from these trials. To classify an instance x, a vote for class *c* is recorded by every classifier for which $C_t(x) = c$, and $C_S(x)$ is then the class with the most votes.

3.5.2 Bootstrapping of rare events

The bootstrap technique is suitable for training DNA sequences which are rare events and have a limited number of examples. Bootstrap procedures can be a way to overcome the difficulties as follows. Suppose, we have DNA sequences S which may not be enough to train neural networks. Hence, we take bootstrapped samples $\{S_B\}$ from {S), and form $\{C(x,S_B)\}$. Finally, let the $\{C(x,S_B)\}$ vote to form $C_B(x)$. The $\{S_B\}$ form replicate data sets, each consisting of N cases, drawn at random, but with replacement, from S. Each example may appear repeated times or not at all in any particular S_B. The $\{S_B\}$ are replicate data set drawn from the bootstrap distribution approximating the distribution from S. By doing this, we get multiple versions of learning sets and gain increased classification accuracy.

3.5.3 Subsampling of common events

We divide common event data into some subsamples to make them balanced with rare event data. That means sampling without replacement from data to get subsamples. We don't need to use the bootstrap because we have enough examples. Subsampling can be thought to be even more intuitive than the bootstrap, because the: subsamples are actually samples from the true distribution, whereas the bootstrap resamples are samples frorn an estimated distribution.

3.5.4 Aggregating of multiple neural networks

The earliest attempt at combining multiple networks can be credited to Nilsson [26] who proposed "committee" machines based on a collection of single layer networks as an attempt to design multilayer neural networks that could classify complicated data. Hansen and Salamon [27] discussed the application of an ensemble of multilayer neural networks. The parallel self-organizing consensual neural network was proposed by Valafar and Ersoy [28] and the parallel consensual neural network by Wenediksson [29]. Opitz and Shavlik [30] presented a technique that searched for a correct and diverse population of neural networks to be used in ensemble by genetic algorithms. Tumer and Ghosh [31] provided an analytical framework to quantify the improvements in classification results due to combining. Freund and Schapire [32] proposed a boosting method which produced a set of classifiers by adjusting the weights of training instances and combind them by voting.

3.5.5 The Bootstrap aggregating rare event neural network

The bootstrap aggregating rare event neural networks (BARENN) is developed for the purpose of increasing classification accuracy, avoiding local minima, reducing learning times, obtaining a high degree of robustness and fault tolerance and achieving truly parallel architecture. The BARENN consists of a set of unit neural networks (UNN's). Each unit is a particular neural network, and can be trained by a learning algorithm such as backpropagation or delta rule. The training procedure for the BARENN is as follows:

1. Divide common event data into n data sets.

2. Bootstrap rare event data into n data sets.
3. Train n NNs independently.
4. Combine the outputs of the individual neural networks by consensus.

The system block diagram for BARENN is shown in Fig. 3.1.

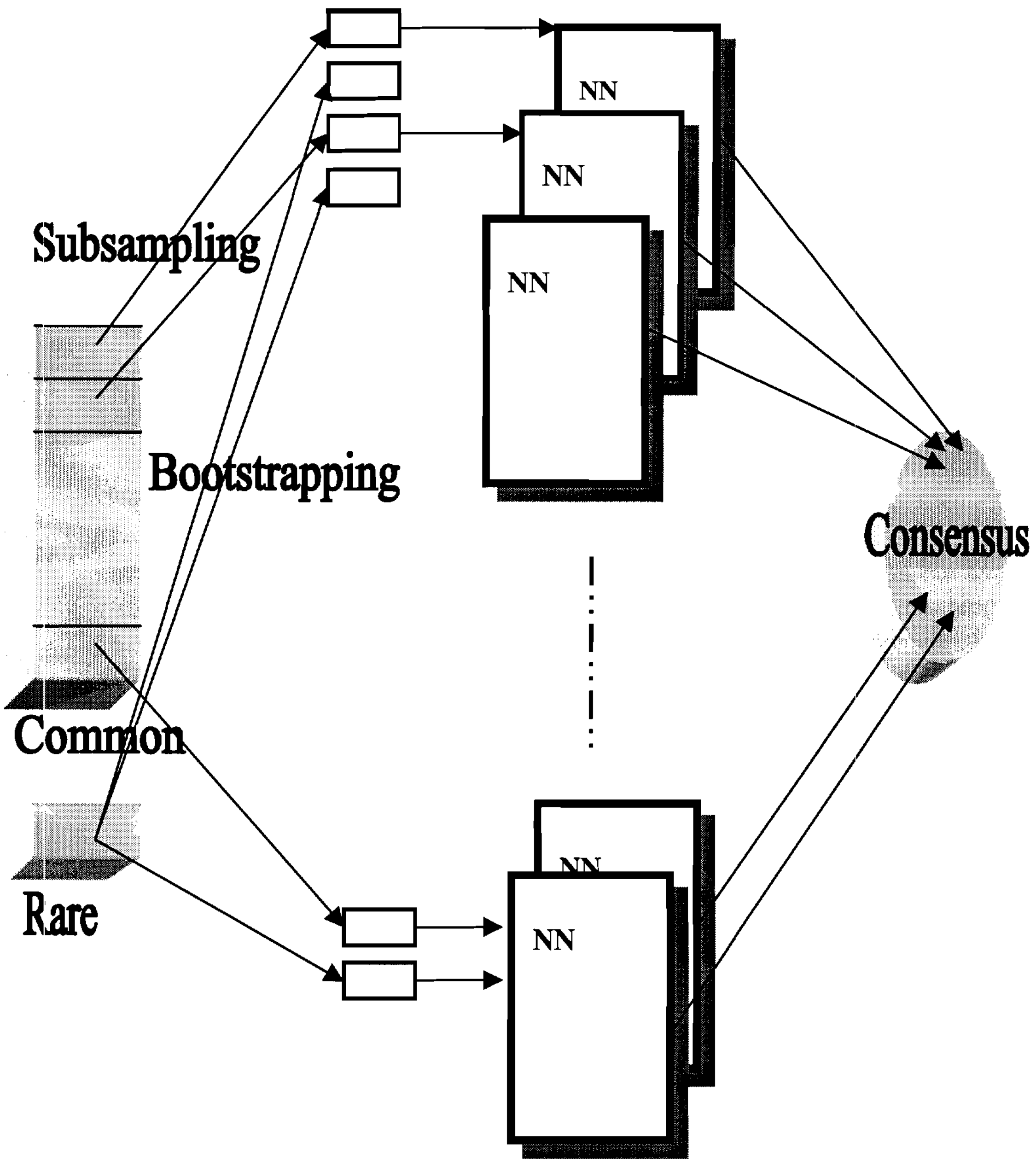


Fig. 3.1 Bootstrap stratification scheme.

3.6 Data Set Used in Experiments

3.6.1 Genomic sequence data

The first data set used in the experiments is genomic sequence data. A sets of Alu sequences (17,073 entries) was obtained from Jurka's Repbase Repository [33]. Table 3.1 shows some of those entries, and Table 3.2 gives an example. Table 3.3 provides a list of Alu sub-classes and their frequency of occurrence in the Repbase.

The other set of data (unique sequences) correspond to annotated human coding sequences in UniGene at NCBI. All annotated human unique sequences were extracted (83,141 entries); They represent protein coding regions. From this set, we discarded the entries that did not contain the string "complete cds", where cds represents the coding region. We obtained 8236 entries. Table 3.4 shows some of the entries, and an example of the sequence data is shown in Table 3.5.

Table 3.1 Locations of extracted repetitive sequences.

HSAL00997.REPBASE;1	HSAL07247.REPBASE;1	HSAL13582.REPBASE;1
HSAL00998.REPBASE;1	HSAL07248.REPBASE;1	HSAL13583.REPBASE;1
HSAL00999.REPBASE;1	HSAL07249.REPBASE;1	HSAL13584.REPBASE;1
HSAL01000.REPBASE;1	HSAL07250.REPBASE;1	HSAL13585.REPBASE;1
HSAL01001.REPBASE;1	HSAL07251.REPBASE;1	HSAL13586.REPBASE;1
HSAL01002.REPBASE;1	HSAL07252.REPBASE;1	HSAL13587.REPBASE;1
HSAL01003.REPBASE;1	HSAL07253.REPBASE;1	HSAL13588.REPBASE;1
HSAL01004.REPBASE;1	HSAL07254.REPBASE;1	HSAL13589.REPBASE;1
HSAL01005.REPBASE;1	HSAL07255.REPBASE;1	HSAL13590.REPBASE
HSAL01007.REPBASE;1	HSAL07256.REPBASE;1	HSAL13591.REPBASE;1
HSAL01008.REPBASE;1	HSAL07257.REPBASE;1	HSAL13592.REPBASE;1
HSAL01009.REPBASE;1	HSAL07258.REPBASE;1	HSAL13593.REPBASE;1
HSAL01010.REPBASE;1	HSAL07259.REPBASE;1	HSAL13594.REPBASE;1
HSAL01011.REPBASE;1	HSAL07260.REPBASE;1	HSAL13595.REPBASE;1
HSAL01012.REPBASE;1	HSAL07261.REPBASE;1	HSAL13596.REPBASE;1
HSAL01013.REPBASE;1	HSAL07262.REPBASE;1	HSAL13597.REPBASE;1
HSAL01014.REPBASE;1	HSAL07263.REPBASE;1	HSAL13598.REPBASE;1
HSAL01015.REPBASE;1	HSAL07264.REPBASE;1	HSAL13599.REPBASE;1
HSAL01016.REPBASE;1	HSAL07265.REPBASE;1	HSAL13600.REPBASE;1
HSAL01017.REPBASE;1	HSAL07266.REPBASE;1	HSAL13601.REPBASE;1
HSAL01018.REPBASE;1	HSAL07267.REPBASE;1	HSAL13602.REPBASE;1
HSAL01019.REPBASE;1	HSAL07268.REPBASE;1	HSAL13603.REPBASE;1
HSAL01020.REPBASE;1	HSAL07269.REPBASE;1	HSAL13604.REPBASE;1
HSAL01021.REPBASE;1	HSAL07270.REPBASE;1	HSAL13605.REPBASE;1
HSAL01022.REPBASE;1	HSAL07271.REPBASE;1	HSAL13606.REPBASE;1
HSAL01023.REPBASE;1	HSAL07272.REPBASE;1	HSAL13607.REPBASE;1
HSAL01024.REPBASE;1	HSAL07273.REPBASE;1	HSAL13608.REPBASE;1
HSAL01025.REPBASE;1	HSAL07274.REPBASE;1	HSAL13609.REPBASE;1
HSAL01026.REPBASE;1	HSAL07275.REPBASE;1	HSAL13610.REPBASE;1
HSAL01027.REPBASE;1	HSAL07276.REPBASE;1	HSAL13611.REPBASE;1
HSAL01028.REPBASE;1	HSAL07277.REPBASE;1	HSAL13612.REPBASE;1
HSAL01029.REPBASE;1	HSAL07278.REPBASE;1	HSAL13613.REPBASE;1
HSAL01030.REPBASE;1	HSAL07279.REPBASE;1	HSAL13614.REPBASE;1

Table 3.2 Contents of a repetitive sequence file.

```
ID  HSAL00581 repbase; DNA; PRI; 338 BP.
CC  HSAL000581 DNA
XX
AC  X67491;
XX
DE  Alu repetitive element (Alu-Sp)
XX
KW  GLUDP5 gene; glutamate dehydrogenase.
XX
OS  Homo sapiens (human)
CC  human
OC  Eukaryota; Animalia; Metazoa; Chordata; Vertebrata; Mammalia;
OC  Theria; Eutheria; Primates; Haplorhini; Catarrhini; Hominidae.
XX
RN  [1]
RP  1-338
RC  [1]  (bases 1 to 2679)
RA  Moschonas N.K.;
RT  "Direct Submission";
RL  Submitted (22-JUL-1992) N.K. Moschonas, Inst. of Molecular Biol.$\&$
RL  Biotechnology, Forth Dept. of Biology, Univ. of Crete, P.O. Box
XX
RN  [2]
RP  1-338
RC  [2]  (bases 1 to 2679)
RA  Tzimagiorgis G., Leversha M.A., Chroniary K., Goulielmos G.,
RA  Sargent C.A., Ferguson-Smith M., Moschonas N.K.;
RT  "Structure and expression analysis of a member of the human
RT  glutamate dehydrogenase (GLUD) gene family mapped to chromosome";
RL  Hum. Genet. 91,433-438
XX
DR  GENBANK; X67491; 85.0.
CC  Positions   2135   2472  Accession No X67491      GenBank (rel. 85.0)
XX
FH  Key           Location/Qualifiers
FH
FT  repeat-region   1..338
FT                /rpt_family="Alu-Sp"
XX
SQ  Sequence 338 BP; 113 A; 78 C; 88 G; 59 T; 0 other;

Hsal00581  Length: 338  September 11, 1997 12:40  Type: N  Check: 1800 ..


  I  GGCCAGGCAC GGTAGCTCAT GCCTACAATC CTAGTGCTTT GGGAGGCCAA

 51  GGCGGGTGGA TCACCTCAGG TAGGGAGTTT GAGACCAGCC TGACCAACAT

101  GGTGAAACCC CGTCTCTAGT AAAAATACAA AAAATTAGCC GGGCGTGGTG

151  GTGCATGCCT GTAATCCCAG CAACTTGGGA GGCTGAGGCA GGAGAATCAC
```

Table 3.3 Types and Distributions of Repetitive Alu Sequences.

Type	Frequency	Type	Frequency
Sx	962	Sc	871
Jb	1474	Ya8	31
Spqxzg	2210	Sz	924
Scpqxzg	48	FLA	192
Sxzg	1324	Sg	285
J	2345	Sq	451
X	907	Ya5	46
Ya	900	Szg	108
Jo	2186	FLAX	47
S	1175	Sb0	24
Sp	550	Spq	13
Cls	939	Total	18012

Table 3.4 Locations of extracted unique sequences.

Hs#S222012	Hs#S305515	Hs#S430646	Hs#S553445	Hs#S696622
Hs#S222230	Hs#S305517	Hs#S430922	Hs#S553460	Hs#S697401
Hs#S222333	Hs#S305519	Hs#S431069	Hs#S553465	Hs#S698622
Hs#S222452	Hs#S305520	Hs#S431125	Hs#S553469	Hs#S699401
Hs#S222538	Hs#S305521	Hs#S431235	Hs#S553474	Hs#S700480
Hs#S222677	Hs#S305524	Hs#S431250	Hs#S553475	Hs#S702622
Hs#S223036	Hs#S305525	Hs#S431669	Hs#S553483	Hs#S703776
Hs#S223101	Hs#S305527	Hs#S431709	Hs#S553486	Hs#S704622
Hs#S223216	Hs#S305528	Hs#S432299	Hs#S553489	Hs#S705436
Hs#S223364	Hs#S305529	Hs#S432345	Hs#S553491	Hs#S705459
Hs#S223414	Hs#S305533	Hs#S432389	Hs#S553492	Hs#S705462
Hs#S223604	Hs#S305537	Hs#S432593	Hs#S553495	Hs#S705463
Hs#S223823	Hs#S305540	Hs#S432644	Hs#S553508	Hs#S705467
Hs#S223936	Hs#S305542	Hs#S432862	Hs#S553514	Hs#S705468
Hs#S224051	Hs#S305543	Hs#S432921	Hs#S553516	Hs#S705469
Hs#S224260	Hs#S305544	Hs#S433162	Hs#S553518	Hs#S705470
Hs#S224334	Hs#S305545	Hs#S433310	Hs#S553519	Hs#S705471
Hs#S224420	Hs#S305546	Hs#S433658	Hs#S553520	Hs#S705473
Hs#S224570	Hs#S305552	Hs#S433795	Hs#S553522	Hs#S705474
Hs#S225120	Hs#S305554	Hs#S433954	Hs#S553524	Hs#S705475
Hs#S226034	Hs#S305555	Hs#S434093	Hs#S553527	Hs#S705476
Hs#S226052	Hs#S305610	Hs#S434611	Hs#S553536	Hs#S705477
Hs#S226054	Hs#S305651	Hs#S434763	Hs#S553550	Hs#S705478
Hs#S226058	Hs#S305787	Hs#S434793	Hs#S553556	Hs#S705479
Hs#S226059	Hs#S305999	Hs#S434950	Hs#S553565	Hs#S705480
Hs#S226061	Hs#S306044	Hs#S435293	Hs#S553572	Hs#S705482
Hs#S226062	Hs#S306169	Hs#S435311	Hs#S553583	Hs#S705486
Hs#S226063	Hs#S306396	Hs#S435420	Hs#S553586	Hs#S705487
Hs#S226064	Hs#S306510	Hs#S435893	Hs#S553587	Hs#S705488
Hs#S226066	Hs#S306647	Hs#S436055	Hs#S553590	Hs#S705489
Hs#S226067	Hs#S307427	Hs#S436136	Hs#S553594	Hs#S705490
Hs#S226070	Hs#S307595	Hs#S436350	Hs#S553600	Hs#S705493

Table 3.5 Contents of a unique sequence file.

```
>gnl|UG|Hs$\#$S582962 Calcium/calmodulin-dependent protein kinase
IV    /gene=CAMK4    /cyto=5q21-q23    /cds=(77,1498)    /gb=D30742
/gi=487908 /ug=Hs.351 /1en=1740

GCGGCGGCTGGCGGCCGGCTTCTCGCTCGGGCAGCGGCGGCGGCGGCGGCGGCTTCC
GGAGTCCCGCTGCGAAGATGCTCAAAGTCACGGTGCCCTCCTGCTCCGCCTCGTCCTGCT
CTTCGGTCACCGCCAGTGCGGCCCCGGGGACCGCGAGCCTCGTCCCGGATTACTGGATCG
ACGGCTCCAACAGGGATGCGCTGAGCGATTTCTTCGAGGTGGAGTCGGAGCTGGGACGGG
GTGCTACATCCATTGTGTACAGATGCAAACAGAAGGGGACCCAGAAGCCTTATGCTCTCA
AAGTGTTAAAGAAAACAGTGGACAAAAAAATCGTAAGAACTGAGATAGGAGTTCTTCTTC
GCCTCTCACATCCAAACATTATAAAACTTAAAGAGATATTTGAAACCCCTACAGAAATCA
GTCTGGTCCTAGAACTCGTCACAGGAGGAGAACTGTTTGATAGGATTGTGGAAAAGGGAT
ATTACAGTGAGCGAGATGCTGCAGATGCCGTTAAACAAATCCTGGAGGCAGTTGCTTATC
TACATGAAAATGGGATTGTCCATCGTGATCTCAAACCAGAGAATCTTCTTTATGCAACTC
CAGCCCCAGATGCACCACTCAAAATCGCTGATTTTGGACTCTCTAAAATTGTGGAACATC
AAGTGCTCATGAAGACAGTATGTGGAACCCCAGGGTACTGCGCACCTGAAATTC'FTAGAG
GTTGTGCCTATGGACCTGAGGTGGACATGTGGTCTGTAGGAATAATCACCTACA'FCTTAC
TTTGTGGATTTGAACCATTCTATGATGAAAGAGGCGATCAGTTCATGTTCAGGAGAATTC
T G A A T T G T G A A T A T T A C T T T A T C T C C C C C T G G T G G G A T G A G G
ACTTGGTCAGAAAATTAATTGTTTTGGATCCAAAGAAACGGCTGACTACATTTCIWGCTC
TCCAGCATCCGTGGGTCACAGGTAAAGCAGCCAATTTTGTACACATGGATACCGCTCAAA
AGAAGCTCCAAGAATTCAATGCCCGGCGTAAGCTTAAGGCAGCGGTGAAGGCTGCFGGTGG
CCTCTTCCCGCCTGGGAAGTGCCAGCAGCAGCCATGGCAGCATCCAGGAGAGCCACAAGG
CTAGCCGAGACCCTTCTCCAATCCAAGATGGCAACGAGGACATGAAAGCTATTCCAGAAG
GAGAGAAAATTCAAGGCGATGGGGCCCAAGCCGCAGTTAAGGGGGCACAGGCTGAGCTGA
TGAAGGTGCAAGCCTTAGAGAAAGTTAAAGGTTAAAGGTGCAGATATAAAGTTAAAGTGCTGAAGAGGCCCCCA
AAATGGTGCCCAAGGCAGTGGAGGATGGGATMGGTGGCTGACCTGGAACTAGAGGAGG
GCCTAGCAGAGGAGAAGCTGAAGACTGTGGAGGAGGCAGCAGCTCCCAGAGAAGGGCAAG
GAAGCTCTGCTGTGGGTTTTGAAGTTCCACAGCAAGATGTGATCCTGCCAGAGTACTAAA
CAGCTTCCTTCAGATCTGGAAGCCAAACACCGGCATTTTATGTACTTTGTCCTTCAGCAA
GAAAGGTGTGGAAGCATGATATGTACTATAGTGATTCTGTTTTTGAGGTGCAAAAAACAT
ACATATATACCAGTTGGTAATTCTAACTTCAATGCATGTGACTGCTTTATGAAAATAATA
```

3.6.2 Normally distributed data

The second data sets are two-dimensional, Gaussian-distributed patterns labeled 1 and 2. We can express the conditional probability density functions for the two classes as follows:

$$f(x \mid C_i) = \frac{1}{2\pi\sigma_i^2} \exp\left(-\frac{1}{2\sigma_i^2} \| x - \mu_i \|^2 \right) \quad \text{for } i = 1,2$$

where μ_1 = mean vector = $[0\ 0]^T$

σ_1^2 = variance = [1 1;1 9]

μ_2 = mean vector = $[3\ 3]^T$

σ_2^2 = variance = [1 1;1 9]

Figures 3.2 and 3.3 show joint scatter plots of classes C1 and C2 for training and for testing, respectively.

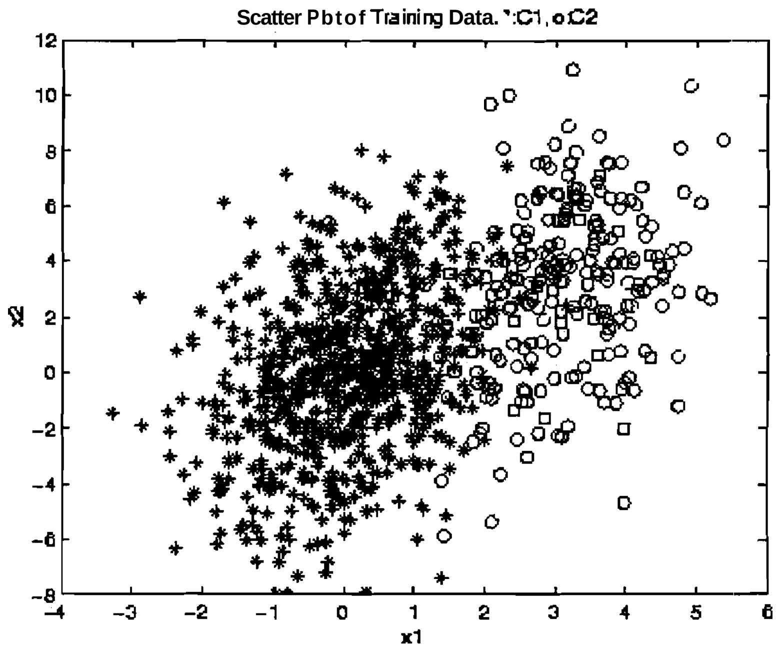


Figure 3.2. Normally distributed synthetic data 1 for training.

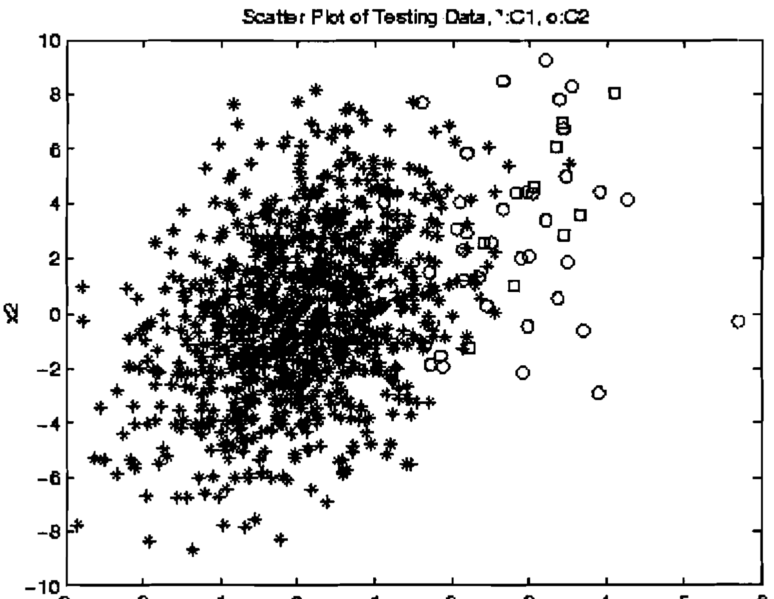


Figure 3.3. Normally distributed synthetic data 1 for training.

3.6.3 Four-class synthetic data

Figure 3.4 and 3.5 show joint scatter plots for four-class synthetic Gaussian-distributed data generated in the following way:

$$f(x \mid C_i) = \frac{1}{2\pi\sigma_i^2}\exp\left(-\frac{1}{2\sigma_i^2}\| x - \mu_i \|^2\right) \quad \text{for } i = 1, 2, 3, 4$$

where $\sigma_i^2 =$ variance$=$ $[2\ 0; 0\ 2]$ for $i = 1, 2, 3, 4$

$\mu_1 =$ mean vector $= [0\ \ 0]^T$

$\mu_2 =$ mean vector $= [0\ \ 5]^T$

$\mu_3 =$ mean vector $= [5\ \ 5]^T$

$\mu_4 =$ mean vector $= [5\ \ 0]^T$

We use this data to show the robustness of the stratifying coefficient scheme for multi-class data.

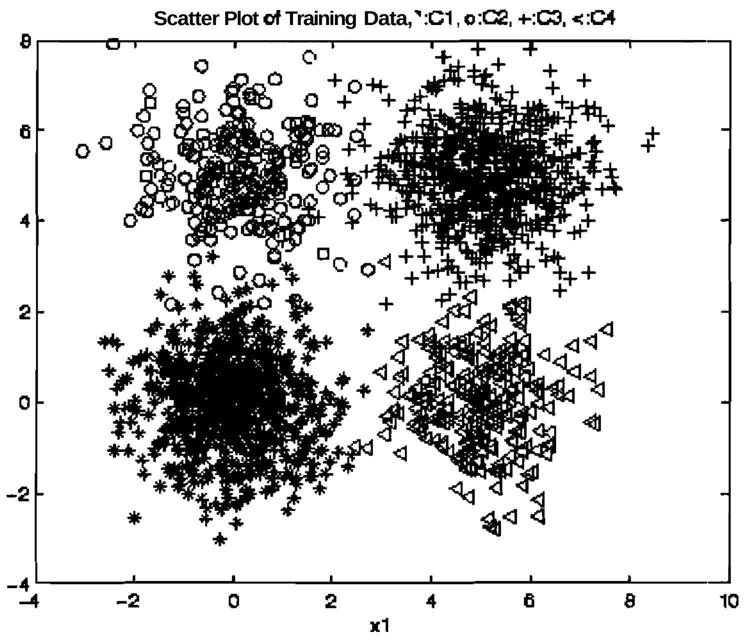


Figure **3.4.** Four-class synthetic data for training.

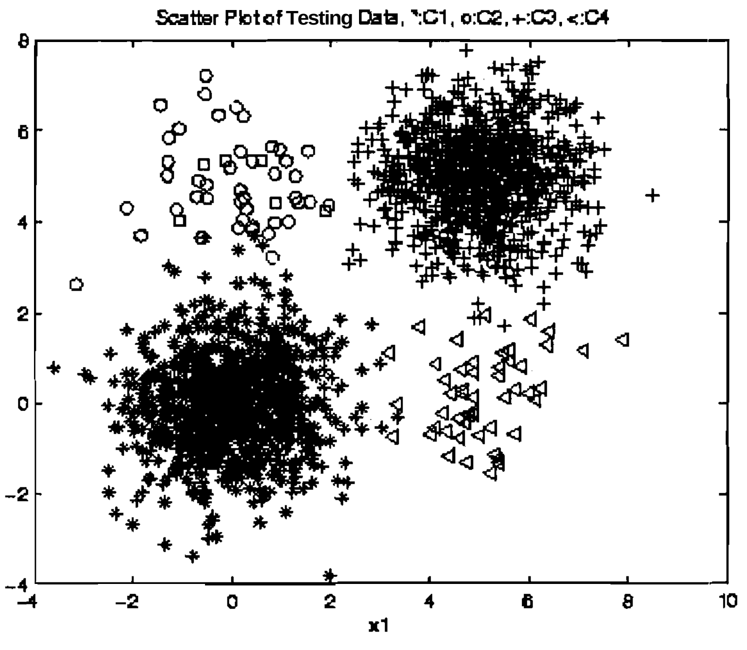


Figure 3.5. Four-class synthetic data for testing.

3.6.4 Remotely sensed data

The third data set is a multispectral earth observation remotely sensed data covering a mountainous area in Colorado [29]. It has 10 ground cover classes which are listed in Table 2. Each channel comprises an image of 135 row and 131 columns, all of which are coregistered. Ground reference data were compiled for the area by comparing a cartographic map to a color composite of the Landsat data and also to a line printer output of each Landsat channel. 2019 ground reference points (11.4% of the area) were selected. Ground reference consisted of two or more homogeneous fields in the imagery for each class. Among them, we chose class 1 and class 10 for our two-class problem because class 10 is a rare event class as compared with class 1 with a ratio of 6 tol.

Table 3.6 Ten dominant classes of the multispectral image data of colorado area.

Class	Field	# of reference points
1	Water	603
2	Colorado blue spruce	112
3	Montane/Subalpine meadow	97
4	Aspen	140
5	Ponderosa pine	244
6	Ponderosa pine/Douglas fir	314
7	Engelmann spruce	294
8	Douglas fir/White fir	76
9	Douglas fir/Ponderosa pine/Aspen	50
10	Douglas fir/White fir/Aspen	99

3.7 Experimental Results

Experimental results provide a test of the theoretical schemes discussed in the previous sections. The classification performances of the new schemes were compared with, a backpropagation neural network (BP), and a BP network incorporating LRWF. The results are provided in Tables 3.8-22. In these tables, detection probability is defined as the percentage of rare events that are correctly classified as rare events. Similarly, false alarm rate is defined as the percentage of the total number of events falsely declared as rare events. The overall testing classification accuracy by 10-fold cross validation and split-sample validation are also given.

To decide our neural network architecture, several model were investigated in terms of number of hidden nodes in hidden layer. The presented experimental results were obtained using the networks with 256 nodes in the input layer, 32 nodes in the hidden layer, and 2 nodes in the output layer. This configuration was chosen after performing t-test when using genomic data on various hidden node configurations. With error performance results for every 10-fold cross-validation, we obtained t-value for each hidden node configuration, and the results are shown in Table 3.7 [34]. The t-values were compared to the critical value of t. The critical vaule of t with significance level of 95 % is 1.96 and the t-values for our experiments were greater than 1.96 and this means that all these networks have sufficiently different performances. We chose the neural network with 32 hidden nodes because it gave the highest cross-validation testing accuracy.

As an overfitting avoidance methodology, we used 10% of training set as a "validation set" to determine when to stop training. We used three different kinds of data to draw general-purpose conclusions.

3.7.1 Experiments with genomic sequence data

The first set of experiments had 2000 examples. We chose the ratio of common event to rare events (to be referred to as class ratio) as 3:1, 9:1, and 19:1. For example, in the 3:1 ratio, the data contained 500 Alu sequences and 1500 sequences from UniGene (non-Alu).

Tables 3.8-11 show the experimental results by cross validation. Tables 3.12-15 show the experimental results by split-sample validation. Table 3.8 shows the results of a conventional neural network. It is observed that the network does not perform well as the class ratio increases. In Table 3.9, similar results were obtained with a 2-stage BP network with LRWF. In these two cases, the neural networks performed poorly. Table 3.10 shows that the first proposed scheme for rare event detection in unbalanced data performs well. We note the BP with LRWF and the BP with stratifying coefficient (SC) have similar weighting schemes, but the weights and consequently the results are different. Table 3.11 shows the results of the BP with bootstrap stratification. Very similar results were obtained by split-sample validation as shown in Table, 3.12-15. Here, we can see the result of simple BP is slightly worse than those of the proposed ones for we use 1000 training data differently from the previous results.

Table 3.7 T-test for performance comparison of various number of hidden nodes (NN1 with no. of hidden nodes:32, NN2 with no. of hidden nodes:64, NN3 with no. of hidden nodes:128)

Comparison	NN1 : NN2	NN2 : NN3	NN3 : NN1
T-Value	-9.1205	-5.6136	5.8273

Table 3.8 Performance of a 2-stage backpropagation neural network (sequence data).

$P[H_0] : P[H_1]$	No. of Testing Data	Testing Accuracy	Detection Probability	Fallse Alarm Rate
3:1	2000	0.8515	0.4280	0.0108
9:1	2000	0.8950	0	0
19:1	2000	0.9535	0.0538	0

Table 3.9 Performance of a 2-stage backpropagation neural network with LRWF (sequence data).

$P[H_0] : P[H_1]$	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
3:1	2000	0.9205	0.7567	0.0283
9:1	2000	0.9036	0.0846	0.0006
19:1	2000	0.9535	0.0535	0

Table 3.10 Performance of a 2-stage backpropagation neural network with stratifying coefficent (sequence data).

$P[H_0] : P[H_1]$	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
3:1	2000	0.9450	0.8723	0.0302
9:1	2000	0.8860	0.8925	0.1125
19:1	2000	0.9865	0.8477	0.0047

Table 3.11 Performance of a bootstrap stratification neural network (sequence data).

$P[H_0] : P[H_1]$	No. of Testing Data	Testing Accuracy	Detection Probability	Fallse Alarm Rate
3:1	2000	0.9055	0.8335	0.0697
9:1	2000	0.9325	0.8671	0.0590
19:1	2000	0.9340	0.8126	0.0581

Table 3.12 Performance of a 2-stage backpropagation neural network (sequence data, $P[H_1]$: 0.25 , $P^*[H_1]$: 0.05).

Sample	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
Sample1	1000	0.9220	0.7800	0.0705
Sample2	1000	0.9210	0.7600	0.0705
Sample3	1000	0.9290	0.6000	0.6537
Sample4	1000	0.9450	0.6200	0.0379
Sample5	1000	0.9370	0.5600	0.0432
Average	1000	0.9308	0.6640	0.0552

Table 3.13 Performance of a 2-stage backpropagation neural network with LRWF (sequence data, $P[H_1]$: 0.25 , $P^*[H_1]$: 0.05).

Sample	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
Sample1	1000	0.9510	0.0200	0
Sample2	1000	0.9500	0	0
Sample3	1000	0.9720	0.5600	0.0063
Sample4	1000	0.9740	0.5600	0.0042
Sample5	1000	0.9500	0.0042	0
Average	1000	0.8131	0.2280	0.0021

Table 3.14 Performance of a 2-stage backpropagation neural network with stratifying coefficent (sequence data, $P[H_1]$: 0.25 , $P^*[H_1]$: 0.05).

Sample	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
Sample1	1000	0.8920	0.8000	0.1032
Sample2	1000	0.9260	0.7600	0.0653
Sample3	1000	0.9270	0.6400	0.0579
Sample4	1000	0.9190	0.6600	0.0674
Sample5	1000	0.9130	0.5400	0.0674
Average	1000	0.9154	0.6800	0.0722

Table 3.15 Performance of a bootstrap stratification neural network (sequence data, $P[H_1]$: 0.25 , $P^*[H_1]$: 0.05).

Sample	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
Sample1	1000	0.9030	0.8000	0.0916
Sample2	1000	0.9020	0.7400	0.0895
Sample3	1000	0.9310	0.6400	0.0537
Sample4	1000	0.9290	0.6400	0.0558
Sample5	1000	0.9070	0.5200	0.0726
Average	1000	0.9144	0.6680	0.0726

3.7.2 Experiments with normally distributed data

The results with Gaussian distributed synthetic data with a class ratio of 9:1 are shown in Table 3.16. It was tested with cross validation. It is observed that detection probability increases with the proposed methods, although there were slight increases in false alarm rate.

Synthetic data can be Laplace or Cauchy distributed, but Bayesian classifier is optimal classifier when data is normally distributed. That's why Bayesiian classifier with normally distributed data is designed to show our scheme's excellence over Bayesian classifier with rare event data. Table 3.17 shows the failure of a simple Bayesian classifier [35] for detection of rare events although it works well for ordinary Gaussian-distributed data. In Table 3.1, we can see the improved results in a Bayesian classifier by the consideration of rare events in terms of Bayesian classifier. This was the Bayes test for minimum cost. By using conditional cost when we design the Bayesian classifier, we can make up for rare event cases but it's not good compared with our proposed schemes.

Table 3.16 Performances of 2-stage backpropagation neural networks (synthetic data).

Neural Networks	No. of Testing Data	Testing Accuracy	Detection Probability	False **Alarm** Rate
BP	1000	0.9520	0.8878	0.0266
LRWF	1000	0.9490	0.8753	0.0267
SC	1000	0.9420	0.9459	0.0584
BS	1000	0.9502	0.9502	0.0475

Table 3.17 Performance of a Bayes classifier for minimum error

(synthetic data, # of train data: 1000, # of test data: 1000).

$P[H_1]$	$P^*[H_1]$	Testing Accuracy	P_D	FAR
0.25	0.05	0.732	0.58	0.0053

Table 3.18 Performance of a Bayes classifier for minimum cost

(synthetic data, # of train data: 1000, # of test data: 1000).

$P[H_1]$	$P^*[H_1[$	Testing Accuracy	P_D	FAR
0.25	0.05	0.72	0.80	0.0189

3.7.3 Experiments with four-class synthetic data

Tables 3.19 to 21 demonstrate the robustness of BP with SC with four-class data. Class 2 and class 4 are rare event classes, and we used stratifying coefficients for class 2 and class 4 data to compensate for the paucity. Simple BP and BP with LRWF fail more frequently than BP with SC. These results indicate that we can preferably utilize this scheme for the detection of rare events for multi-class data.

Table 3.19 Performance of a 2-stage backpropagation neural network

(four-class synthetic data, # of train data: 1000, # of test data::1000).

$P[H_1]$	$P^*[H_1]$	Testing Accuracy	P_D	FAR
0.25	0.05	97.10	0.98	0.0036
0.25	0.05	48.1	0.36	0.0047
0.25	0.05	95.0	0	0
0.25	0.05	95.0	0	0
0.25	0.05	68.85	0	0.0176
0.25	0.05	96.95	0	0
0.25	0.05	49.55	0	0
0.25	0.05	95.0	0	0
0.25	0.05	95.0	0	0
0.25	0.05	48.15	0	0
	Average	78.97	0.1340	0.0026

Table3.20 Performance of a 2-stage backpropagation neural network with LRWF

(four-class synthetic data, # of train data: 1000, # of test data: 1000).

$P[H_1]$	$P^*[H_1]$	Testing Accuracy	P_D	FAR
0.25	0.05	97.15	0.96	0.0021
0.25	0.05	97.2	0.96	0.0014
0.25	0.05	95.0	0	0
0.25	0.05	95.0	0	0
0.25	0.05	97.15	0.94	0.0007
0.25	0.05	95.0	0	0
0.25	0.05	95.0	0	0
0.25	0.05	97.2	0.98	0.0013
0.25	0.05	95.0	0	0
0.25	0.05	47.55	0.02	0.1003
	Average	91.125	0.386	0.0106

Table 3.21 Performance of a 2-stage backpropagation neural network with SC(four-class synthetic data, # of train data: 1000, # of test data: 1000).

$P[H_l]$	$P^*[H_l]$	Testing Accuracy	P_D	FAR
0.25	0.05	51.5	0.98	0.1779
0.25	0.05	96.9	0	0
0.25	0.05	51.55	0.98	0.2590
0.25	0.05	98.3	0.98	0.0071
0.25	0.05	49.75	0	0
0.25	0.05	98.3	0.98	0.0081
0.25	0.05	98.65	1	0.0049
0.25	0.05	51.9	1	0.0703
0.25	0.05	98.75	1	0.0038
0.25	0.05	98.65	1	0.0039
	Average	79.425	0.79	0.0535

3.7.4 Experiments with remotely sensed data

The results with remotely sensed data was shown in Table 3.22. It was tested with cross validation and class ratio of 6:1. It is observed that detection probability increases with the proposed methods. Large improvement is observed with the proposed schemes in terms of detection probability and false alarm rate for remotely sensed data. We note that gaussian data used is two-dimensional, whereas genomic data and remotely sensed data are 256-dimensional and 7-dimensional, respectively. These results may imply that proposed schemes achieve larger improvement in detection probability and false alarm rate as the dimensionality of data increases.

Table 3.22 Performances of 2-stage backpropagation neural networks (remotely sensed data).

Neural Networks	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
BP	702	0.8643	0.1458	0.0016
LRWF	702	0.8729	0.2000	0
SC	702	0.7886	0.6896	0.2035
BS	702	0.9729	0.8759	0.0066

3.7.5 Experiments with a bootstrap stratification neural network II

Another type of a bootstrap stratification neural network was tried. There are two diflerences from the algorithm discussed in Section **3.5.** The first difference is bootstrapping of common event. The second is boostrapping of rare event more than the original size of the rare event to make the size of both events equal. The algorithm is as follows:

1. Bootstrap common event data into n data sets.
2. Bootstrap rare event data into n data sets, then each data set has size larger than original data size. The data sets of common event and rare event have the same size.
3. Train n NNs independently.
4. Combine the outputs of the individual neural networks by consensus.

The: experimental results of this algorithm are shown in Tables **3.23** through **3.25.** These results are not promising since rare event detection becomes totally dominant.

Table 3.23 Performance of a bootstrap stratification neural network II (genomic sequence data).

$P[H_0] : P[H_1]$	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
3:1	2000	0.2520	1	1
9:1	2000	0.0850	1	1
19:1	2000	0.0350	1	1

Table 3.24 Performance of a bootstrap stratification neural network II (synthetic data).

$P[H_0] : P[H_1]$	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
9:1	1000	0.5740	0.9960	0.05872

Table 3.25 Performance of a bootstrap stratification neural network II (remote sensing data).

$P[H_0] : P[H_1]$	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
6:1	702	0.1529	1	1

3.8 Conclusions

In this chapter, we presented neural network methods for rare event detection. We developed two sample stratification techniques for rare event detection. In the first scheme, we introduced the stratifying coefficients which modify the probability distribution of the underlying random process in order to make rare events appear to occur more frequently. This method uses stratifying coefficients multiplying the weighted sum of the derivatives during the backward pass of training in the modified backpropagation algorithm. In the second scheme, we introduced a bootstrap technique which is especially valuable when data sizes are too small to invoke good results. These two schemes make rare events have a better chance of being represented in the sample for training, and improve the detection probability and false alarm rate of' rare events. The results indicate that the proposed schemes have the potential of significantly improving the classification performance of neural networks to recognize rare events.

4. RULE EXTRACTION BY NEURAL NETWORKS

4.1 Introduction

Neural networks were originally not efficient for rule extraction. Indeed, it has been a common criticism of black box approach of neural networks that they are nice to come up with an answer, without the underlying rules. The result of training a neural network is internal weights distributed throughout the network. These weights provide no direct insight why the solution is valid. The weights are not readily understandable although sophisticated techniques for probing into a neural network help provide some explanation. Neural network developers are well aware of this problem. and state-of-the-art neural network research during the past few years addressed this issue. For neural networks to gain an even wider degree of user acceptance and to enhance their overall utility as learning and generalization tools, it is highly desirable that an explanation capability becomes an integral part of the functionality of a trained NN. Since rule extraction from neural networks comes at a cost in terms of resources and additional efforts, it is imperative to delineate the reason why rule extraction is an important extension of conventional NN techniques. The merits in including rule extraction techniques as an adjunct to conventional neural network learning include the following [36]:

1. Provide a user explanation capability.
2. Extend NN systems to 'safety-critical' problem domains.
3. Give software verification and debugging of NN components in software systems.
4. Improve the generalization of NN solutions.
5. Data exploration and the induction of scientific theories.
6. Knowledge acquisition for symbolic AI systems.

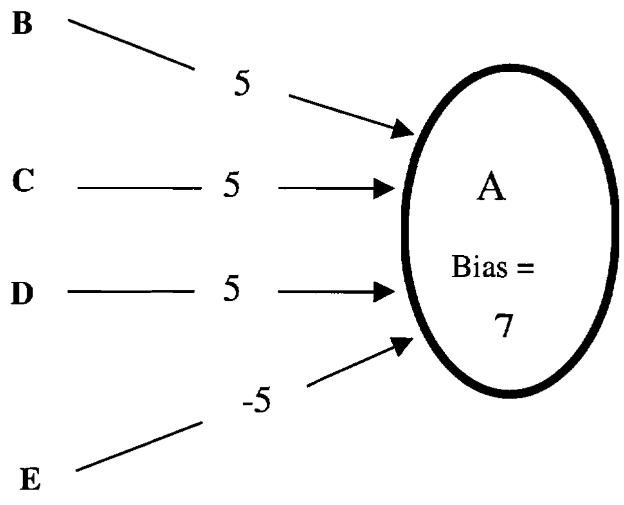
4.2 Existing Schemes [36][38]

This section describes some of the existing approaches with emphasis on extracting rules from feedforward NN architectures. A review of different rule extraction approaches is provided by a technical report written by Andrews et al [36].

4.2.1 Decompositional approaches

This type of approach focuses on searching for and extracting conventional Boolean rules at the level of the individual (hidden and output) units within the trained NN. The basic strategy is to search initially for sets of weights containing a single link/connection of sufficient (positive) value to guarantee that the bias on the unit being analyzed is exceeded irrespective of the values on the other links/connections. If a link is found which satisfies the criterion, it is written as a rule. The search then proceeds to subsets of two elements and the rules extracted at the individual unit level are then aggregated to form the composite rule base for the NN as a whole. Finally, the algorithm removes subsumed rules. For example, given that the link weights and the bias are shown as in Fig. 4.1(a), the algorithm would initially find five rules. After eliminating one subsumed rule (if B,C,D, and not E, then A), the algorithm returns the four rules listed in Fig. 4.1(b) (assuming that all units have activations near zero or one).

The earliest implementation of this style algorithm was the KT algorithm developed by Fu [37]. A more recent example of this line of approach is the subset algorithm developed by Towell and Shavlik [38]. Fu reported initial success in applying the KT algorithm to the problem domain of wind shear detection by infrared sensors. Similarly, Towell and Shavlik [38] showed that their subset implementation is capable of



(a)

if B, C, and not (E) then **A.**

if B, D, and not (E) then **A.**

if C, D, and not (E) then **A.**

if B,C,D then **A.**

(b)

Fig. 4.1 Rule extraction by weight search.

Table 4.1 Subset algorithm.

```
For each hidden and output unit:
  Extract up to $S_P$ subsets of the positively-weighted incoming links for which
  the summed weight is greater than the bias on the unit;

    For each element p of the $S_P$ subsets:
        Search for a set $S_N$ of a set of negative-attributes so that
        the summed weight of p plus the summed weights of N-n
        exceed the threshold on the unit;
        where N is the set of all negative-attributes and n is an element of $S_N$

        With each element n of the $S_N$ set, form a rule:
        'if p and NOT n, then the concept'
```

delivering a set of rules which are, at least, potentially tractable and smaller than many handcrafted expert systems. However, some problems with both the KT and subset algorithms are as follows: I) the solution time for finding all possible subsets is a function of the size of the power set of the links to each unit, 2) the algorithm extracts a large set of rules, 3) some generated rules may be repetitive, 4) the extracted rules tend to hide significant structure in the trained networks, 5) the values of hidden unit activation function is continuous although we threshold the value into binary values.

Some methods was introduced to make up for these weaknesses. The option used by Fu for restricting the size of the solution search space is to place a ceiling on the number of antecedents per extracted rule. Unfortunately, this potentially has adverse implications for rule quality since some rules may be omitted. Setiono [39] showed the improved results by discretizing the hidden unit activation values. Notwithstanding their limitations, the rules extracted from both algorithms are simple to understand. It also offers the capability to provide transparency of the trained NN solution at the level of individual hidden and output units. Generally, the rules extracted by both KT and Subset algorithms are tractable especially in small application domains.

An important development is another rule extraction algorithm called M-of-N by Towell and Shavlik for the purpose of making up for the deficiencies of the subset algorithm. The name of the algorithm reflects the rule format the algorithm uses to represent the extracted rules:

If "at least" M of the following N premises are true
then the concept designated by the unit is true.

The rationale behind the M-of-N is to find a group of links that form an equivalence class in that all class members have the same effect (i.e. have similar weight values) and can be used interchangeably with one another. M-of-N extracts rules from the KBANN trained networks through six main procedures [38]. Rules extracted are reportedly superior than rules extracted by other symbolic approaches such as C4.5 [40].

4.2.2 Pedagogical approaches

Another class of rule extraction approach extracts rules from a neural network only by examining their input-output mapping behavior. An example of such a rule exti-action approach is the algorithm developed by Saito and Nakano to extract medical diagnostic rules from a trained network [41]. BRANNIE [42], Rule-extraction-as-learning [43], and DEDEC [44] are other examples of extracting rules by investigating the input-out mapping of trained networks. For example, the BRANNIE system of Sestito and Dillon is designed to extract rules from neural networks trained using standard backpropagation. It has been classified as pedagogical since the basic motivation is to use a measure of the closeness between the networks' inputs and outputs; as the basis for generating the rule set. A major innovation in the BRAINNE technique is the capability to deal with continuous data as input without first having to employ a discretization phase.

4.2.3 Other approaches

Research in the area of fuzzy logic neural networks (FLNNs) or neurofuzzy systems is concerned with combining neural networks and fuzzy logic. Some FLNN systems include a fuzzy rule extraction module for refining fuzzy set membership functions and explaining the trained neural network [45].

Recurrent networks have shown great success in representing finite state languages and deterministic finite state automata (DFA). The DFA rules extraction algorithm improves network generalization performance based on the stability of internal DFA representation [46].

4.3 Proposed Scheme

4.3.1 Parallel self-organizing hierarchical neural networks [47]

The Parallel self-organizing hierarchical neural networks (PSHNN) [47], [48] architecture is proposed for the purposes of increasing classification accuracy, reducing leaning rates, and achieving true parallelism.

The PSHNN involves a number of neural networks, similar to a multilayer network. Each neural network is a particular neural network to be referred to as a stage neural network (SNN). Unlike a multilayer network, each SNN is essentially independent of the other SNN's in the sense that each SNN does not receive its input directly from the previous SNN. At the output of each SNN there is an error detection scheme. If an input vector is rejected, it goes through a nonlinear transformation before being input to the next SNN. Only those input vectors which are rejected by the present stage are fed into the next stage after the nonlinear transformations. The error detection scheme is based on the fact that sufficiently complex neural networks trained by delta rule under square error loss estimate Bayesian posterior probabilities. By interpreting outputs of neural networks as probabilities, one can set error detection bounds. The error detection scheme utilizes these bounds and is described as follows. Error bounds are the largest posterior probabilities observed for error causing vectors. Hence, error bounds provide information about how far the tail extends into the region where main body of other class' distribution lies. No-error bounds are the smallest posterior probabilities for the vectors causing no errors. Similarly, no-error bounds are points that get closest to the Bayesian classifier threshold.

The following is the algorithm for estimating the error bounds. There are three vectors involved: the input vector X, the output vector after the activation function Y with elements y_i, and the final output vector Z with elements z_i.

Error Bounds

```
Assume:    number of data vectors = l
           length of vector = n
           $y^i_j$ = j th component of the ith vector $Y^I$

Initialize the error bounds as
       $y^0_j(upper) = 0.5$
       $y^0_,(lower) = 0.5$  where  $j = 1, 2, \ldots, n$

Initialize: i= I.

I. Check whether the ith data vector is an error-potential vector. If so
       a) If $y^i_j \geq 0.5$, then $y^i_j(upper) = MAX\ [\, y^{(i-1)}_j(upper), y^i_j\,]$
       b) If $y^i_j < 0.5$, then $y^i_j(lower) = MZN\ [\, y^{(i-1)}_j(lower), y^i_j\,]$
2. If i=l, the final error bounds are
       $r_j(upper) = y^l_j(upper)$
       $r_j(lower) = y)(lower)$
  else i = i + I and go to step I.
  end
```

The following is the procedure for estimating the no-error bounds:

No-Error Bounds

```
Initialize the error bounds as
       $y^0_j(upper) = 0$
       $y^0_,(lower) = I$  where  $j = I, 2, \ldots, n$

Initialize: i= I
```

```
1. Check whether the I th data vector is an error-potential vector.
   If so, then $i = i + 1$ and go to step 1
   else go to step 2.

2. Update the no-error bounds $y^i_j$ for $j = 1,2, \ldots, n$ as follows:

      a) If $y^i_j \geq 0.5$, then $y^i_j(upper) = MIN [\, y^{(i-1)}_j(upper), y^i_j \,]$
      b) If $y^i_j < 0.5$, then $y^i_j(lower) = MAX [\, y^{(i-1)}_j(lower), y^i_j \,]$

3. If $i=l$, the final error bounds are
      $s_j(upper) = y^l_j(upper)$
      $r_j(lower) = y^l_j(lower)$
   else $i = i + 1$ and go to step 1.
   end
```

A procedure which gives best results experimentally is to utilize both the error and the no-error bounds. For this purpose, three intervals $I_1(j)$, $I_2(j)$, $I_E(j)$, $j=1,2,\ldots,n$ are defined as

$$I_1(j) = [\, r_j(lower), r_j(upper) \,] \tag{4.6}$$

$$I_2(j) = [\, s_j(lower), s_j(upper) \,] \tag{4.7}$$

$$I_E(j) = I_1(j) \cap I_2(j) \tag{4.8}$$

Then an input vector is classified as an error-potential vector if any y_j belongs to $I_E(j)$.

The motivation for the proposed architecture evolves from the consideration that most errors occur due to input signals which are linearly nonseparable or which are close to boundaries between classes. At the output of each neural network, such signals are detected by a scheme and rejected. Then rejected signals are passed through a nonlinear transformation so that they are converted into other vectors which are more easily classified by the succeeding neural networks

Learning with the PSHNN is similar to learning with a multilayer network, except that the error detection is carried out at the output of each SNN, and the procedure is stopped without further propagation into the succeeding SNN's if no errors are detected. Classification with the PSHNN can be done in parallel with the SNN's simultaneously rather than each SNN waiting for data from the previous SNN, as seen in Fig. 4.2.

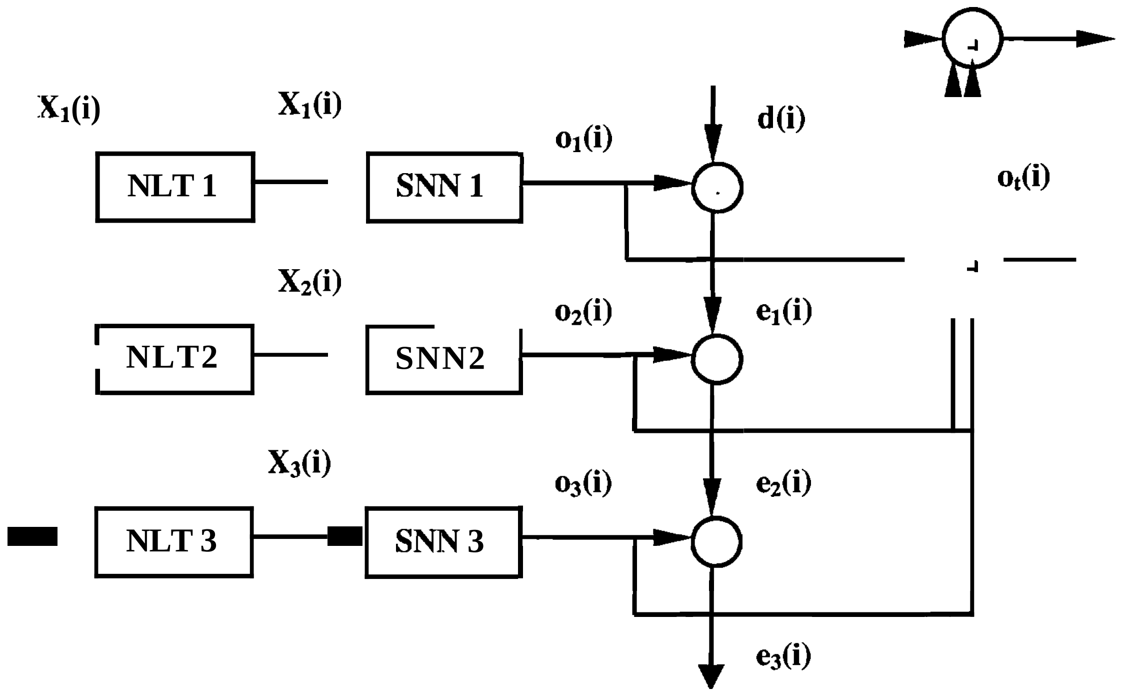


Fig. 4.2 Block diagram of a three-stage PSHNN.

4.3.2 A method for rule extraction with PSHNN

It is pointed out from early research papers [37], [61] that the search space for potential rules is very large, and it is necessary to limit the set of combinations of weights to a reasonable size. This means only a subset of possible rules are used, the trained system covers the decision space partially, and this results in classification as undecided for some input vectors. It may be convenient to cover the cases that cannot be covered by extracted rules by using default rules. If a default rule is used, its output. can be chosen to maximize the correct classification rate. Such default rules make the set of extracted rules complete but they do not provide any interpretation of why a decision is made.

In this section, we discuss a rule extraction method using PSHNN to get around this problem. As mentioned before, PSHNN postpone decision making on the examples in the boundary of each class and perform another test. We can apply PSHNN concept in rule extraction. That is, the input vectors which can not be covered by extracted rules are fed into another neural network, and rules are extracted from the network. It will be referred to as rule extracting PSHNN. In this method, the rejection schemes are constructed differently from the conventional PSHNN. The flow diagram for the rejection scheme for rule extracting system is shown in Fig. 4.4. As stage neural network SNN_i of the PSHNN is constructed, the rules from the stage are also generated. The resulting rule-based system from the stage will be referred to as SRB_i . The rejection scheme is based on the error-bounds, discussed earlier in Section 4.3.1, as well as decided/undecided outputs from SRB_i.

SRB_i can be based on any of the algorithms discussed in Section 4.2. The algorithm we used was based on subset algorithm from [38], [61]. The idea underlying this algorithm is that it first sorts both positive and negative incoming links for each output in descending order into two different sets based on their values. Starting from the highest positive weight, it searches for individual incoming links that can cause an output to be active regardless of other input links to this node.

Theoretically we have to consider all combinations of weights but it is NP-hard to consider all the combinations. We use some heuristics to reduce searclh space. First, we limit the number of antecedents as in the algorithm by Fu [38]. Secondly, we take into account the effect of negative weights by considering the sum of significant negative weights rather than individual negative weight with these heuristics, we develop the basic rule extraction algorithm for each stage of REPSHNN.

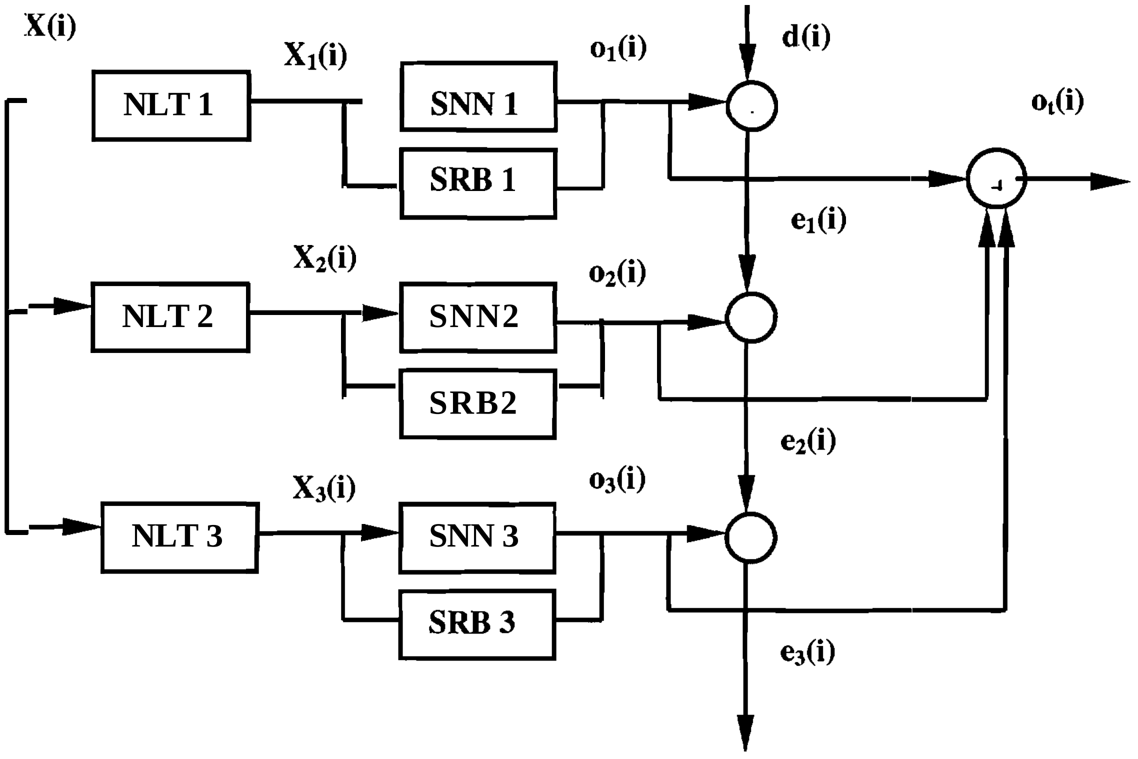


Fig. 4.3 Block diagram of a three-stage rule extracting PSHNN.

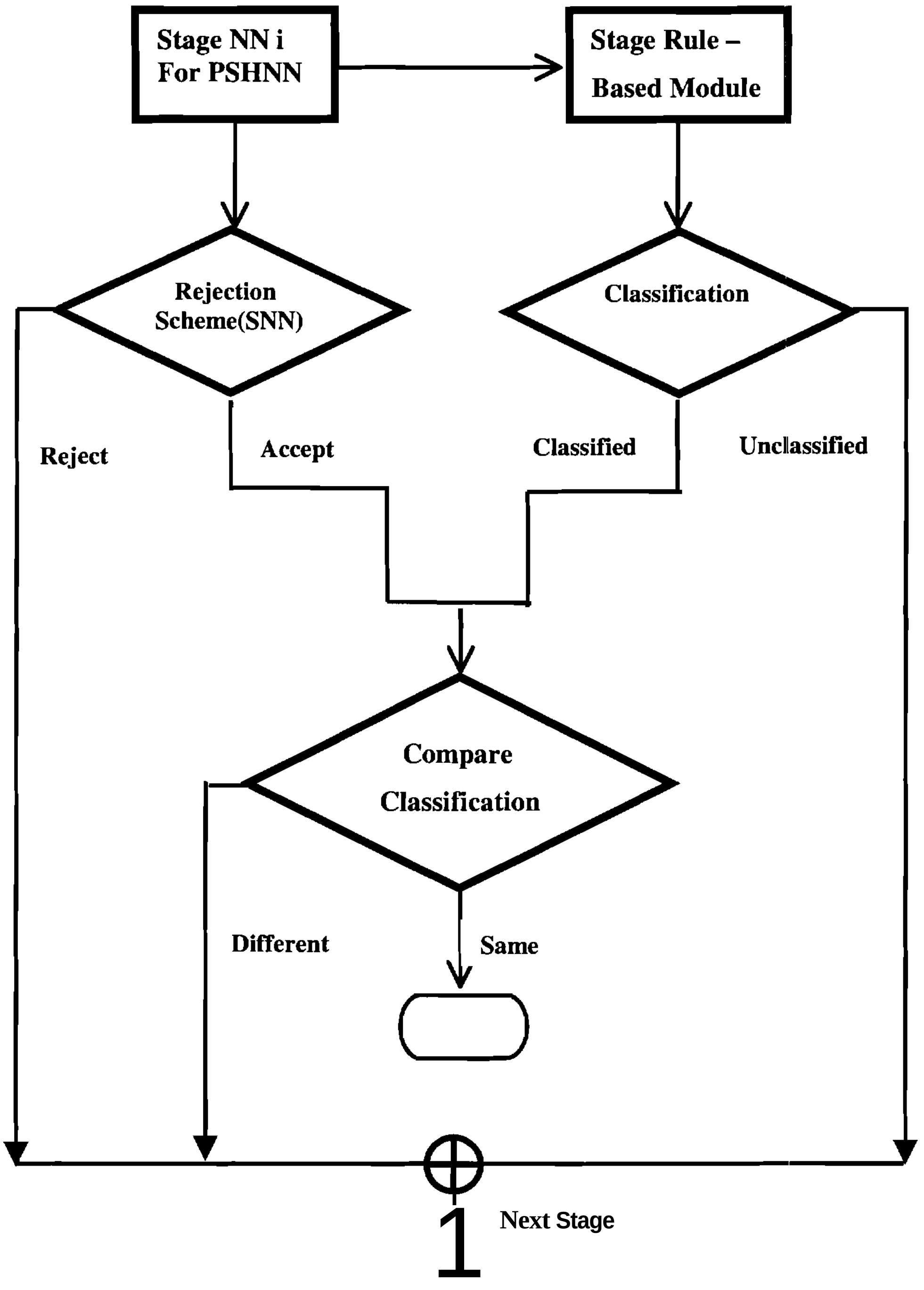


Fig. 4.4 Flow diagram for rejection schemes in REPSHNN system.

4.3.3 Rule extraction procedure

The rule extracting PSHNN consists of a number of stages. Each stage is a pair of a neural network module and a rule based subsystem. At the output of each stage, there is an error detection scheme which allows acceptance or rejection of input vectors. If an input vector is rejected, it is fed into the next stage. The flow diagram for rejection schemes is shown in Fig. 4.4. The procedure is as follows:

Training

Initialization: n=0

Phase 1. Increase n by 1. Train SNN_n (the n-th stage neural network) by a simple learning algorithm such as delta rule and construct SRB, (the n-th stage rule based module) by the rule extraction algorithm below.

Phase 2. Construct rejection boundaries.

Phase 3. Select the input data which are detected to give output errors.

Phase 4. Provide the rejected inputs to the next stage.

Phase 5. Go to phase 1 unless a sufficient number of stages are generated.

Rule Extraction

Phase 1. Prune the neural network module. Then we have the neural network module with significant weights.

Phase 2. Divide weights into positive and negative weights.

Phase 3. Search for big positive weights enough to cause the output to be active regardless of other weights.
If there is such a weight, it produces a rule:
" If input i is active, then output j is active."
If a weight is found strong enough to activate an output j, then this weight is marked and cannot be used in any further combinations when checking the same output j.
It continues checking subsequent weights in the positive set until it cannot find that activates the current output j by itself.

Phase 4. If more exact rules are needed, then the algorithm investigates combinations of positive weights. So it generates a rule:
" If input i and input j are active, then output k is active."
If more detailed rules are required, then the algorithm starts looking for combinations of two unmarked links starting from the first element of the positive set. This process continues until the algorithm reaches its terminating criteria.

Phase 5. A rewriting procedure is needed after all the rules are extracted if more than two layers are used. Through the rewriting procedure, the results from each layers (hidden layers and output layer) are combined.

Evaluation

The following describes the complete evaluation procedure of the proposed system.

Phase 1. Input the testing vector to SNN_n.

Phase 2. Check whether the testing vector is rejected by reject schemes.

Phase 3. If rejected, go to Phase 1 and test again with next SNN_{n+1}.

If not, decide the class.

4.4 Extracted Rules

Tables 4.2-4.7 show the rules we obtained by the methods we explained in Section 4.3.2.1. Table 4.2 contains the rules from the first stage of the REPSHNN with the Alu and non-Alu sequence data. Table 4.3 is from the second stage and Table 4.4 is frorn the third stage of the proposed system. Tables. 4.5-4.7 reveals the results with leader and non-leader sequence data by REPSHNN.

Table 4.2 Extracted rules from first stage of REPSHNN

(Alu and non-Alu sequences, class 1: non-Alu, class 2: Alu).

```
A40 = C   A46 = A      ->  class 1
A40 = C   A4 = C       ->  class 1
A40 = C   A5 = A       ->  class 1
A40 = C   A39 = C      ->  class 1
A40 = C   A12 = C      ->  class 1
A40 = C   A62 = N      ->  class 1
A40 = C   A1 = A       ->  class 1
A40 = C   A8 = C       ->  class 1
A40 = C   A43 = N      ->  class 1
A40 = C   A29 = G      ->  class 1
A40 = C   A38 = C      ->  class 1
A40 = C   A61 = C      ->  class 1
A40 = C   A10 = C      ->  class 1
A40 = C   A50 = A      ->  class 1
A46 = A   A4 = C       ->  class 1
A46 = A   A5 = A       ->  class 1
A46 = A   A39 = C      ->  class 1
A46 = A   A12 = C      ->  class 1
A46 = A   A62 = N      ->  class 1
A46 = A   A1 = A       ->  class 1
A46 = A   A8 = C       ->  class 1
A46 = A   A43 = N      ->  class 1
A46 = A   A29 = G      ->  class 1
A46 = A   A38 = C      ->  class 1
A46 = A   A61 = C      ->  class 1
A46 = A   A10 = C      ->  class 1
A46 = A   A50 = A      ->  class 1
A4 = C   A5 = A        ->  class 1
A4 = C   A39 = C       ->  class 1
A4 = C   A12 = C       ->  class 1
A4 = C   A62 = N       ->  class 1
A4 = C   A1 = A        ->  class 1
A4 = C   A8 = C        ->  class 1
A4 = C   A43 = N       ->  class 1
A4 = C   A29 = G       ->  class 1
A4 = C   A38 = C       ->  class 1
A4 = C   A61 = C       ->  class 1
A4 = C   A10 = C       ->  class 1
A4 = C   A50 = A       ->  class 1
A5 = A   A39 = C       ->  class 1
A5 = A   A12 = C       ->  class 1
A5 = A   A62 = N       ->  class 1
A5 = A   A1 = A        ->  class 1
A5 = A   A8 = C        ->  class 1
A5 = A   A43 = N       ->  class 1
A5 = A   A29 = G       ->  class 1
A5 = A   A38 = C       ->  class 1
A5 = A   A61 = C       ->  class 1
```

```
A4 = G    A38 = G        ->  class 2
A0 = A    A2 = C    A4 = A        ->  class 2
A0 = A    A2 = C    A37 = G       ->  class 2
A0 = A    A2 = C    A40 = G       ->  class 2
A0 = A    A2 = C    A44 = G       ->  class 2
A0 = A    A2 = C    A49 = A       ->  class 2
A0 = A    A2 = C    A52 = A       ->  class 2
A0 = A    A2 = C    A56 = A       ->  class 2
A0 = A    A2 = C    A57 = A       ->  class 2
A0 = A    A2 = C    A59 = A       ->  class 2
A0 = A    A2 = C    A61 = N       ->  class 2
A0 = A    A2 = C    A62 = A       ->  class 2
A0 = A    A4 = A    A37 = G       ->  class 2
A0 = A    A4 = A    A40 = G       ->  class 2
A0 = A    A4 = A    A44 = G       ->  class 2
A0 = A    A4 = A    A49 = A       ->  class 2
A0 = A    A4 = A    A52 = A       ->  class 2
A0 = A    A4 = A    A56 = A       ->  class 2
A0 = A    A4 = A    A57 = A       ->  class 2
A0 = A    A4 = A    A59 = A       ->  class 2
A0 = A    A4 = A    A61 = N       ->  class 2
A0 = A    A4 = A    A62 = A       ->  class 2
A0 = A    A37 = G    A40 = G       ->  class 2
A0 = A    A37 = G    A44 = G       ->  class 2
A0 = A    A37 = G    A49 = A       ->  class 2
A0 = A    A37 = G    A52 = A       ->  class 2
A0 = A    A37 = G    A56 = A       ->  class 2
A0 = A    A37 = G    A57 = A       ->  class 2
A0 = A    A37 = G    A59 = A       ->  class 2
A0 = A    A37 = G    A61 = N       ->  class 2
A0 = A    A37 = G    A62 = A       ->  class 2
A0 = A    A40 = G    A44 = G       ->  class 2
A0 = A    A40 = G    A49 = A       ->  class 2
A0 = A    A40 = G    A52 = A       ->  class 2
A0 = A    A40 = G    A56 = A       ->  class 2
A0 = A    A40 = G    A57 = A       ->  class 2
A0 = A    A40 = G    A59 = A       ->  class 2
A0 = A    A40 = G    A61 = N       ->  class 2
A0 = A    A40 = G    A62 = A       ->  class 2
A0 = A    A44 = G    A49 = A       ->  class 2
A0 = A    A44 = G    A52 = A       ->  class 2
A0 = A    A44 = G    A56 = A       ->  class 2
A0 = A    A44 = G    A57 = A       ->  class 2
A0 = A    A44 = G    A59 = A       ->  class 2
A0 = A    A44 = G    A61 = N       ->  class 2
A0 = A    A44 = G    A62 = A       ->  class 2
A0 = A    A49 = A    A52 = A       ->  class 2
A0 = A    A49 = A    A56 = A       ->  class 2
A0 = A    A49 = A    A57 = A       ->  class 2
A0 = A    A49 = A    A59 = A       ->  class 2
A0 = A    A49 = A    A61 = N       ->  class 2
A0 = A    A49 = A    A62 = A       ->  class 2
A37 = G    A40 = G    A44 = G       ->  class 2
```

Table 4.3 Extracted rules from second stage of REPSHNN
(Alu and non-Alu sequences, class 1: non-Alu, class 2: Alu).

A30 = N	A12 = G	->	class 1
A30 = N	A24 = C	->	class 1
A30 = N	A10 = N	->	class 1
A30 = N	A4 = C	->	class 1
A30 = N	A17 = A	->	class 1
A30 = N	A49 = N	->	class 1
A30 = N	A31 = N	->	class 1
A30 = N	A23 = C	->	class 1
A30 = N	A7 = A	->	class 1
A30 = N	A13 = A	->	class 1
A30 = N	A46 = G	->	class 1
A30 = N	A62 = C	->	class 1
A30 = N	A36 = A	->	class 1
A30 = N	A41 = C	->	class 1
A12 = G	A24 = C	->	class 1
A12 = G	A10 = N	->	class 1
A12 = G	A4 = C	->	class 1
A12 = G	A17 = A	->	class 1
A12 = G	A49 = N	->	class 1
A12 = G	A31 = N	->	class 1
A12 = G	A23 = C	->	class 1
A12 = G	A7 = A	->	class 1
A12 = G	A13 = A	->	class 1
A12 = G	A46 = G	->	class 1
A12 = G	A62 = C	->	class 1
A12 = G	A36 = A	->	class 1
A12 = G	A41 = C	->	class 1
A24 = C	A10 = N	->	class 1
A24 = C	A4 = C	->	class 1
A24 = C	A17 = A	->	class 1
A24 = C	A49 = N	->	class 1
A24 = C	A31 = N	->	class 1
A24 = C	A23 = C	->	class 1
A24 = C	A7 = A	->	class 1
A24 = C	A13 = A	->	class 1
A24 = C	A46 = G	->	class 1
A24 = C	A62 = C	->	class 1
A24 = C	A36 = A	->	class 1
A24 = C	A41 = C	->	class 1
A10 = N	A4 = C	->	class 1
A10 = N	A17 = A	->	class 1
A10 = N	A49 = N	->	class 1
A10 = N	A31 = N	->	class 1
A10 = N	A23 = C	->	class 1
A10 = N	A7 = A	->	class 1
A10 = N	A13 = A	->	class 1
A10 = N	A46 = G	->	class 1
A10 = N	A62 = C	->	class 1
A10 = N	A36 = A	->	class 1

```
A10 = N   A41 = C        -> class 1
A4 = C    A17 = A        -> class 1
A4 = C    A49 = N        -> class 1
A4 = C    A31 = N        -> class 1
A4 = C    A23 = C        -> class 1
A4 = C    A7 = A         -> class 1
A4 = C    A13 = A        -> class 1
A4 = C    A46 = G        -> class 1
A4 = C    A62 = C        -> class 1
A4 = C    A36 = A        -> class 1
A4 = C    A41 = C        -> class 1
A17 = A   A49 = N        -> class 1
A17 = A   A31 = N        -> class 1
A17 = A   A23 = C        -> class 1
A17 = A   A7 = A         -> class 1
A17 = A   A13 = A        -> class 1
A17 = A   A46 = G        -> class 1
A17 = A   A62 = C        -> class 1
A17 = A   A36 = A        -> class 1
A17 = A   A41 = C        -> class 1
A49 = N   A31 = N        -> class 1
A49 = N   A23 = C        -> class 1
A49 = N   A7 = A         -> class 1
A49 = N   A13 = A        -> class 1
A49 = N   A46 = G        -> class 1
A49 = N   A62 = C        -> class 1
A49 = N   A36 = A        -> class 1
A49 = N   A41 = C        -> class 1
A31 = N   A23 = C        -> class 1

A46 = C   A5 = G         -> class 2
A46 = C   A42 = G        -> class 2
A46 = C   A43 = A        -> class 2
A46 = C   A16 = N        -> class 2
A46 = C   A29 = A        -> class 2
A46 = C   A41 = G        -> class 2
A46 = C   A26 = N        -> class 2
A46 = C   A30 = C        -> class 2
A11 = A   A17 = C   A19 = G      -> class 2
A11 = A   A17 = C   A32 = A      -> class 2
A11 = A   A17 = C   A60 = A      -> class 2
A11 = A   A19 = G   A32 = A      -> class 2
A11 = A   A19 = G   A58 = G      -> class 2
A11 = A   A19 = G   A60 = A      -> class 2
A17 = C   A19 = G   A32 = A      -> class 2
A17 = C   A19 = G   A58 = G      -> class 2
A17 = C   A19 = G   A60 = A      -> class 2
A17 = C   A32 = A   A58 = G      -> class 2
A17 = C   A32 = A   A60 = A      -> class 2
A17 = C   A58 = G   A60 = A      -> class 2
A19 = G   A32 = A   A58 = G      -> class 2
A19 = G   A32 = A   A60 = A      -> class 2
A19 = G   A58 = G   A60 = A      -> class 2
```

Table 4.4 Extracted rules from third stage of REPSHNN

(Alu and non-Alu sequences, class 1: non-Alu, class 2: Alu).

```
A0  = G    A19 = A                -> class 1
A0  = G    A25 = G                -> class 1
A0  = G    A34 = G                -> class 1
A0  = G    A18 = G                -> class 1
A0  = G    A4  = G                -> class 1
A0  = G    A6  = A                -> class 1
A0  = G    A56 = A                -> class 1
A0  = G    A13 = N                -> class 1
A0  = G    A2  = A                -> class 1
A0  = G    A36 = N                -> class 1
A0  = G    A55 = C                -> class 1
A0  = G    A33 = A                -> class 1
A0  = G    A63 = C                -> class 1
A0  = G    A55 = A                -> class 1
A19 = A    A25 = G                -> class 1
A19 = A    A34 = G                -> class 1
A19 = A    A18 = G                -> class 1
A19 = A    A4  = G                -> class 1
A19 = A    A6  = A                -> class 1
A19 = A    A56 = A                -> class 1
A19 = A    A13 = N                -> class 1
A25 = G    A34 = G                -> class 1
A25 = G    A18 = G                -> class 1
A25 = G    A4  = G                -> class 1
A25 = G    A6  = A                -> class 1
A34 = G    A18 = G                -> class 1
A34 = G    A4  = G                -> class 1
A34 = G    A6  = A                -> class 1
A18 = G    A4  = G                -> class 1

A22 = N    A37 = G                -> class 2
A22 = N    A11 = G                -> class 2
A0  = A    A1  = G    A2  = A     -> class 2
A0  = A    A1  = G    A10 = N     -> class 2
A0  = A    A1  = G    A21 = A     -> class 2
A0  = A    A1  = G    A26 = N     -> class 2
A0  = A    A1  = G    A27 = N     -> class 2
A0  = A    A1  = G    A30 = C     -> class 2
A0  = A    A1  = G    A33 = C     -> class 2
A0  = A    A1  = G    A50 = G     -> class 2
A0  = A    A1  = G    A51 = C     -> class 2
A0  = A    A1  = G    A56 = G     -> class 2
A0  = A    A2  = A    A10 = N     -> class 2
A0  = A    A2  = A    A21 = A     -> class 2
A0  = A    A2  = A    A26 = N     -> class 2
A0  = A    A2  = A    A27 = N     -> class 2
A0  = A    A2  = A    A30 = C     -> class 2
A0  = A    A2  = A    A33 = C     -> class 2
A0  = A    A2  = A    A50 = G     -> class 2
```

```
A0 = A    A2 = A    A51 = C        -> class 2
A0 = A    A2 = A    A56 = G        -> class 2
A0 = A    A10 = N    A21 = A        -> class 2
A0 = A    A10 = N    A26 = N        -> class 2
A0 = A    A10 = N    A27 = N        -> class 2
A0 = A    A10 = N    A30 = C        -> class 2
A0 = A    A10 = N    A33 = C        -> class 2
A0 = A    A10 = N    A50 = G        -> class 2
A0 = A    A10 = N    A51 = C        -> class 2
A0 = A    A10 = N    A56 = G        -> class 2
A0 = A    A21 = A    A26 = N        -> class 2
A0 = A    A21 = A    A27 = N        -> class 2
A0 = A    A21 = A    A30 = C        -> class 2
A0 = A    A21 = A    A33 = C        -> class 2
A0 = A    A21 = A    A50 = G        -> class 2
A0 = A    A21 = A    A51 = C        -> class 2
A0 = A    A21 = A    A56 = G        -> class 2
A0 = A    A26 = N    A27 = N        -> class 2
A0 = A    A26 = N    A30 = C        -> class 2
A0 = A    A26 = N    A33 = C        -> class 2
A0 = A    A26 = N    A50 = G        -> class 2
A0 = A    A26 = N    A51 = C        -> class 2
A0 = A    A26 = N    A56 = G        -> class 2
A0 = A    A27 = N    A30 = C        -> class 2
A0 = A    A27 = N    A33 = C        -> class 2
A0 = A    A27 = N    A50 = G        -> class 2
A0 = A    A27 = N    A51 = C        -> class 2
A0 = A    A27 = N    A56 = G        -> class 2
A0 = A    A30 = C    A33 = C        -> class 2
A0 = A    A30 = C    A50 = G        -> class 2
A0 = A    A30 = C    A51 = C        -> class 2
A0 = A    A30 = C    A56 = G        -> class 2
A0 = A    A33 = C    A50 = G        -> class 2
A0 = A    A33 = C    A51 = C        -> class 2
A0 = A    A33 = C    A56 = G        -> class 2
A0 = A    A50 = G    A51 = C        -> class 2
A0 = A    A50 = G    A56 = G        -> class 2
A0 = A    A51 = C    A56 = G        -> class 2
A1 = G    A2 = A    A10 = N        -> class 2
A1 = G    A2 = A    A21 = A        -> class 2
A1 = G    A2 = A    A26 = N        -> class 2
A1 = G    A2 = A    A27 = N        -> class 2
A1 = G    A2 = A    A30 = C        -> class 2
A1 = G    A2 = A    A33 = C        -> class 2
A1 = G    A2 = A    A50 = G        -> class 2
A1 = G    A2 = A    A51 = C        -> class 2
A1 = G    A2 = A    A56 = G        -> class 2
A1 = G    A10 = N    A21 = A        -> class 2
A1 = G    A10 = N    A26 = N        -> class 2
A1 = G    A10 = N    A27 = N        -> class 2
A1 = G    A10 = N    A30 = C        -> class 2
A1 = G    A10 = N    A33 = C        -> class 2
A1 = G    A10 = N    A50 = G        -> class 2
A1 = G    A10 = N    A51 = C        -> class 2
```

Table 4.5 Extracted rules from first stage of REPSHNN

(leader and non-leader sequences, class 1: non-leader, class 2: leader).

```
A0 = G  ->  class 1
A0 = G    A30 = N  ->  class 1
A0 = G    A42 = C  ->  class 1
A0 = G    A41 = C  ->  class 1
A0 = G    A4 = C  ->  class 1
A0 = G    A13 = A  ->  class 1
A0 = G    A49 = C  ->  class 1
A0 = G    A23 = C  ->  class 1
A0 = G    A60 = G  ->  class1
A0 = G    A51 = C  ->  class 1
A0 = G    A27 = G  ->  class 1
A0 = G    A36 = N  ->  class 1
A0 = G    A8 = C  ->  class 1
A0 = G    A39 = N  ->  class 1
A0 = G    A26 = N  ->  class 1
A30 = N    A42 = C  ->  class 1
A30 = N    A41 = C  ->  class 1
A30 = N    A4 = C  ->  class 1
A30 = N    A13 = A  ->  class 1
A30 = N    A49 = C  ->  class 1
A30 = N    A23 = C  ->  class 1
A30 = N    A60 = G  ->  class 1
A30 = N    A51 = C  ->  class 1
A30 = N    A27 = G  ->  class 1
A30 = N    A36 = N  ->  class 1
A30 = N    A8 = C  ->  class 1
A30 = N    A39 = N  ->  class 1
A30 = N    A26 = N  ->  class 1
A42 = C    A41 = C  ->  class1
A42 = C    A4 = C  ->  class 1
A42 = C    A13 = A  ->  class 1
A42 = C    A49 = C  ->  class 1
A42 = C    A23 = C  ->  class 1
A42 = C    A60 = G  ->  class 1
A42 = C    A51 = C  ->  class 1
A42 = C    A27 = G  ->  class 1
A42 = C    A36 = N  ->  class 1
A42 = C    A8 = C  ->  class 1
A42 = C    A39 = N  ->  class 1
A42 = C    A26 = N  ->  class1
A41 = C    A4 = C  ->  class 1
A41 = C    A13 = A  ->  class 1
A41 = C    A49 = C  ->  class 1
A41 = C    A23 = C  ->  class 1
A41 = C    A60 = G  ->  class 1
A41 = C    A51 = C  ->  class 1
A41 = C    A27 = G  ->  class 1
A41 = C    A36 = N  ->  class 1
```

```
A0 = A  ->  class 2
A0 = A    A13 = N  ->  class 2
A0 = A    A56 = A  ->  class 2
A0 = A    A25 = N  ->  class 2
A0 = A    A5 = G  ->  class 2
A0 = A    A26 = G  ->  class 2
A0 = A    A2 = C  ->  class 2
A0 = A    A27 = N  ->  class 2
A0 = A    A59 = N  ->  class 2
A0 = A    A53 = G  ->  class 2
A0 = A    A43 = A  ->  class 2
A0 = A    A9 = C  ->  class 2
A0 = A    A4 = A  ->  class 2
A0 = A    A13 = G  ->  class 2
A0 = A    A24 = A  ->  class 2
A13 = N    A56 = A  ->  class 2
A13 = N    A25 = N  ->  class 2
A13 = N    A5 = G  ->  class 2
A13 = N    A26 = G  ->  class 2
A13 = N    A2 = C  ->  class 2
A13 = N    A27 = N  ->  class 2
A13 = N    A59 = N  ->  class 2
A13 = N    A53 = G  ->  class 2
A13 = N    A43 = A  ->  class 2
A13 = N    A9 = C  ->  class 2
A13 = N    A4 = A  ->  class 2
A13 = N    A13 = G  ->  class 2
A13 = N    A24 = A  ->  class2
A56 = A    A25 = N  ->  class 2
A56 = A    A5 = G  ->  class 2
A56 = A    A26 = G  ->  class 2
A56 = A    A2 = C  ->  class 2
A56 = A    A27 = N  ->  class 2
A56 = A    A59 = N  ->  class2
A56 = A    A53 = G  ->  class 2
A56 = A    A43 = A  ->  class 2
A56 = A    A9 = C  ->  class 2
A56 = A    A4 = A  ->  class 2
A56 = A    A13 = G  ->  class 2
A56 = A    A24 = A  ->  class 2
A25 = N    A5 = G  ->  class 2
A25 = N    A26 = G  ->  class 2
A25 = N    A2 = C  ->  class 2
A25 = N    A27 = N  ->  class 2
A25 = N    A59 = N  ->  class 2
A25 = N    A53 = G  ->  class 2
A25 = N    A43 = A  ->  class 2
A25 = N    A9 = C  ->  class 2
A25 = N    A4 = A  ->  class 2
A25 = N    A13 = G  ->  class 2
A5 = G    A26 = G  ->  class 2
A5 = G    A2 = C  ->  class 2
A5 = G    A27 = N  ->  class 2
```

Table 4.6 Extracted rules from second stage of REPSHNN

(leader and non-leader sequences, class 1: non-leader, class 2: leader)

```
A2 = A  ->  class 1
A28 = G    A53 = C  ->  class 1
A28 = G    A31 = G  ->  class 1
A28 = G    A25 = A  ->  class 1
A28 = G    A25 = G  ->  class 1
A28 = G    A2 = G  ->  class 1
A28 = G    A47 = G  ->  class 1
A28 = G    A32 = G  ->  class 1
A28 = G    A26 = G  ->  class 1
A28 = G    A57 = C  ->  class 1
A28 = G    A49 = N  ->  class 1
A28 = G    A2 = A  ->  class 1
A28 = G    A50 = A  ->  class 1
A28 = G    A15 = A  ->  class 1
A28 = G    A38 = C  ->  class 1
A53 = C    A31 = G  ->  class 1
A53 = C    A25 = A  ->  class 1
A53 = C    A25 = G  ->  class 1
A53 = C    A2 = G  ->  class 1
A53 = C    A47 = G  ->  class 1
A53 = C    A32 = G  ->  class 1
A53 = C    A26 = G  ->  class 1
A53 = C    A57 = C  ->  class 1
A53 = C    A49 = N  ->  class 1
A53 = C    A2 = A  ->  class 1
A53 = C    A50 = A  ->  class 1
A53 = C    A15 = A  ->  class 1
A53 = C    A38 = C  ->  class 1
A31 = G    A25 = A  ->  class 1
A31 = G    A25 = G  ->  class 1
A31 = G    A2 = G  ->  class 1
A31 = G    A47 = G  ->  class 1
A31 = G    A32 = G  ->  class 1
A31 = G    A26 = G  ->  class 1
A31 = G    A57 = C  ->  class 1
A31 = G    A49 = N  ->  class 1
A31 = G    A2 = A  ->  class 1
A31 = G    A50 = A  ->  class 1
A31 = G    A15 = A  ->  class 1
A31 = G    A38 = C  ->  class 1
A25 = A    A2 = G  ->  class 1
A25 = A    A47 = G  ->  class 1
A25 = A    A32 = G  ->  class 1
A25 = A    A26 = G  ->  class 1
A25 = A    A57 = C  ->  class 1
A25 = A    A49 = N  ->  class 1
```

```
A27 = A  -> class 2
A27 = A    A25 = N  ->  class 2
A27 = A    A38 = A  ->  class 2
A27 = A    A45 = G  ->  class 2
A27 = A    A15 = G  ->  class 2
A27 = A    A48 = A  ->  class 2
A27 = A    A43 = G  ->  class 2
A27 = A    A31 = G  ->  class 2
A27 = A    A52 = A  ->  class 2
A27 = A    A21 = G  ->  class 2
A27 = A    A17 = A  ->  class 2
A27 = A    A55 = C  ->  class 2
A27 = A    A23 = A  ->  class 2
A27 = A    A56 = N  ->  class 2
A27 = A    A17 = C  ->  class 2
A25 = N    A38 = A  ->  class 2
A25 = N    A45 = G  ->  class 2
A25 = N    A15 = G  ->  class 2
A25 = N    A48 = A  ->  class 2
A25 = N    A43 = G  ->  class 2
A25 = N    A31 = G  ->  class 2
A25 = N    A52 = A  ->  class 2
A25 = N    A21 = G  ->  class 2
A25 = N    A17 = A  ->  class 2
A25 = N    A55 = C  ->  class 2
A25 = N    A23 = A  ->  class 2
A25 = N    A56 = N  ->  class 2
A25 = N    A17 = C  ->  class 2
A38 = A    A45 = G  ->  class 2
A38 = A    A15 = G  ->  class 2
A38 = A    A48 = A  ->  class 2
A38 = A    A43 = G  ->  class 2
A38 = A    A31 = G  ->  class 2
A38 = A    A52 = A  ->  class 2
A38 = A    A21 = G  ->  class 2
A38 = A    A17 = A  ->  class 2
A38 = A    A55 = C  ->  class 2
A38 = A    A23 = A  ->  class 2
A38 = A    A56 = N  ->  class 2
A38 = A    A17 = C  ->  class 2
A45 = G    A15 = G  ->  class 2
A45 = G    A48 = A  ->  class 2
A45 = G    A43 = G  ->  class 2
A45 = G    A31 = G  ->  class 2
A45 = G    A52 = A  ->  class 2
A45 = G    A21 = G  ->  class 2
A45 = G    A17 = A  ->  class 2
A45 = G    A55 = C  ->  class 2
A45 = G    A23 = A  ->  class 2
A45 = G    A56 = N  ->  class 2
A45 = G    A17 = C  ->  class 2
A15 = G    A48 = A  ->  class 2
A15 = G    A43 = G  ->  class 2
```

Table 4.7 Extracted rules from third stage of REPSHNN

(leader and non-leader sequences, class 1: non-leader, class 2: leader).

```
A2 = N  ->  class 1
A47 = A    A48 = C  ->  class 1
A47 = A    A7 = C  ->  class 1
A47 = A    A16 = N  ->  class 1
A47 = A    A10 = C  ->  class 1
A47 = A    A51 = A  ->  class 1
A47 = A    A56 = C  ->  class 1
A47 = A    A31 = A  ->  class 1
A47 = A    A22 = A  ->  class1
A47 = A    A10 = A  ->  class 1
A47 = A    A17 = N  ->  class 1
A47 = A    A12 = A  ->  class 1
A47 = A    A2 = N  ->  class 1
A47 = A    A52 = C  ->  class 1
A47 = A    A3 = A  ->  class 1
A48 = C    A7 = C  ->  class 1
A48 = C    A16 = N  ->  class 1
A48 = C    A10 = C  ->  class 1
A48 = C    A51 = A  ->  class 1
A48 = C    A56 = C  ->  class 1
A48 = C    A31 = A  ->  class 1
A48 = C    A22 = A  ->  class 1
A48 = C    A10 = A  ->  class 1
A48 = C    A17 = N  ->  class 1
A48 = C    A12 = A  ->  class 1
A48 = C    A2 = N  ->  class 1
A48 = C    A52 = C  ->  class 1
A48 = C    A3 = A  ->  class 1
A7 = C    A16 = N  ->  class 1
A7 = C    A10 = C  ->  class 1
A7 = C    A51 = A  ->  class 1
A7 = C    A56 = C  ->  class 1
A7 = C    A31 = A  ->  class 1
A7 = C    A22 = A  ->  class 1
A7 = C    A10 = A  ->  class 1
A7 = C    A17 = N  ->  class 1
A7 = C    A12 = A  ->  class 1
A7 = C    A2 = N  ->  class 1
A7 = C    A52 = C  ->  class 1
A7 = C    A3 = A  ->  class 1
A16 = N    A10 = C  ->  class 1
A16 = N    A51 = A  ->  class 1
A16 = N    A56 = C  ->  class 1
A16 = N    A31 = A  ->  class 1
A16 = N    A22 = A  ->  class 1
A16 = N    A10 = A  ->  class 1
A16 = N    A17 = N  ->  class 1
A16 = N    A12 = A  ->  class 1
```

```
A4 = G    A16 = C  ->  class 2
A4 = G    A29 = A  ->  class 2
A4 = G    All = N  ->  class 2
A4 = G    A28 = G  ->  class 2
A4 = G    A46 = A  ->  class 2
A4 = G    A59 = C  ->  class 2
A4 = G    A12 = N  ->  class 2
A4 = G    A63 = A  ->  class 2
A4 = G    A20 = G  ->  class 2
A4 = G    A18 = N  ->  class 2
A4 = G    A21 = C  ->  class 2
A4 = G    A2 = N  ->  class 2
A4 = G    A6 = N  ->  class 2
A4 = G    A15 = A  ->  class 2
A16 = C    A29 = A  ->  class 2
A16 = C    All = N  ->  class 2
A16 = C    A28 = G  ->  class 2
A16 = C    A46 = A  ->  class 2
A16 = C    A59 = C  ->  class 2
A16 = C    A12 = N  ->  class 2
A16 = C    A63 = A  ->  class 2
A16 = C    A20 = G  ->  class 2
A16 = C    A18 = N  ->  class 2
A16 = C    A21 = C  ->  class 2
A16 = C    A2 = N  ->  class 2
A16 = C    A6 = N  ->  class 2
A16 = C    A15 = A  ->  class 2
A29 = A    All = N  ->  class 2
A29 = A    A28 = G  ->  class 2
A29 = A    A46 = A  ->  class 2
A29 = A    A59 = C  ->  class 2
A29 = A    A12 = N  ->  class 2
A29 = A    A63 = A  ->  class 2
A29 = A    A20 = G  ->  class 2
A29 = A    A18 = N  ->  class 2
A29 = A    A21 = C  ->  class 2
A29 = A    A2 = N  ->  class 2
A29 = A    A6 = N  ->  class 2
A29 = A    A15 = A  ->  class 2
All = N    A28 = G  ->  class 2
All = N    A46 = A  ->  class 2
All = N    A59 = C  ->  class 2
All = N    A12 = N  ->  class 2
All = N    A63 = A  ->  class 2
All = N    A20 = G  ->  class 2
All = N    A18 = N  ->  class 2
All = N    A21 = C  ->  class 2
All = N    A2 = N  ->  class 2
All = N    A6 = N  ->  class 2
All = N    A15 = A  ->  class 2
A28 = G    A46 = A  ->  class 2
A28 = G    A59 = C  ->  class 2
A28 = G    A12 = N  ->  class 2
```

4.5 Conclusions

In this chapter, a new rule extraction technique in connection with parallel self-organizing hierarchical neural networks was introduced. Neural networks were originally not efficient for rule extraction and it has been a common criticism of black box approach of neural networks that they are nice to come up with an answer without underlying rules. Sorne researchers gave solutions to the problem, but they had a serious drawback. For the cases of input data which cannot be covered by extracted rules, they just give default rules which have no interpretation about the data. We proposed a new scheme which does not use the default rules. In the proposed scheme, more rules are generated from other neural networks with the input data which cannot be covered by previously extracted rules. This means that we can obtain more information about the data than using the simple default rules which cannot provide any clues about the data.

5. RULE EXTRACTION BY HYBRID NEURAL NETWORK-DECISION TREE SYSTEM

5.1 Introduction

It is desirable to combine consensus theoretic approaches with neural networks and decision trees since consensus theory has the goal of combining several opinions, and a collection of different neural networks and decision trees should be more accurate than a single neural network and/or decision tree, at least in the mean square sense. Moreover, feedforward neural networks minimizing mean–square error at the output have been shown to approximate posterior probabilities when one output neuron is assigned to each class [58], and decision trees have been shown to be easier to extract rules from. Using these properties, it becomes possible to implement consensus theory in neural networks and decision trees.

5.2 Preliminaries

5.2.1 Consensus theory [29]

Consensus theory [51], [52], [53], [54], [55], [56] is a well-established research field involving procedures with the goal of combining single probability distributions to summarize estimates from multiple experts with the assumption that the experts make decisions based on Bayesian decision theory. Consensus theory is closely related to the method of stacked generalization [57] where outputs of experts are combined in a weighted sum with weights which are based on the individual performances of the experts. In most consensus theoretic methods, each data source is at first considered separately. For a given source, an appropriate training procedure can be used to model the data by a number of source-specific densities that will characterize the source [29]. The data types are assumed to be very general. The source-specific classes or clusters are therefore referred to as data classes, since they are defined from relationships in a particular data space. In general, there may not be a simple one-to-one relation between the user-desired information classes are not necessarily a property. of the data. In consensus theory, the information from the data sources is aggregated by a global membership function, and the data are classified according to the usual maximum selection rule into the information classes. The combination formula obtained is a consensus rule.

Consensus theory can be justified by the fact that a group decision is better in tenns of mean square error than a decision from a single expert. To show this, let us define an indicator function as

$$1_{\omega_j} = \begin{cases} 1 & \text{if } \omega_j \text{ occurs} \\ 0 & \text{if } w, \text{ does not occur,} \end{cases} \tag{5.2}$$

where ω_j is an information class. Now it is needed to find an estimate, r, of the "best" probability that minimizes the mean square error

$$\varepsilon^2_{\omega_j}(r) = \sum_{Z} (r - I_{\omega_j})^2 p(Z) \tag{5.3}$$

where Z = [$z_{i,}$ ... , z_n] is a compound vector consisting of observations; from all the data sources, n is the number of data sources, z_i ($i = 1, \ldots, n$) is an observation from a single data source (z_i can be a vector if the corresponding data source makes a multidimensional observation), and $p(Z)$ is the prior probability of Z. Differentiating $\varepsilon^2_{\omega_j}$(r) with respect to r and setting the result equal to zero gives

$$\sum_{Z} 2(r - I_{\omega_j}) p(Z) = 0 \qquad (5.4)$$

The solution to the above equation is r = p(w, | Z) which implies that the group probability p(ω_j | Z) is optimal for classification in the mean square sense.

Several consensus rules have been proposed. Probably the most commonly used consensus rule is the linear opinion pool which has the following (group probability) fonm for the information class ω_j if n is data sources are used:

$$C_j(\omega_j \mid Z) = \sum_{i=1}^{n} \lambda_i p(\omega_j \mid z_i) \qquad (5.5)$$

$p(\omega_j \mid z_i)$ is a source-specific posteriori probability and λ_i's (i = *1, ... , n*) are source-specific weights which control the relative influence of the data sources. $C_j(\omega_j \mid Z)$ is a cornbined posteriori probability for class j. The weights are associated with the sources in the global membership function to express quantitatively the goodness of each source [53].

The linear opinion pool has a number of appealing properties. For example, it is sirnple, yields a probability distribution, and the weight λ_i reflects in some way the relative expertise of the ***i-th*** expert. Also, if the data sources have absolutely continuous probability distributions, the linear opinion pool gives an absolutely continuous distribution. In using the linear opinion pool, it is assumed that all of the experts observe the input vector Z. Therefore, Eq. (5.5) is simply a weighted average of the probability disitributions from all the experts and the result $C_j(\omega_j \mid Z)$ is a combined probability disitribution. The linear opinion pool, though simple, has several disadvantages [54]. First, it shows dictatorship when Bayes' theorem is applied, i.e., only one data source will dominate in making a decision. Second, it is also not externally Elayesian, i.e., the dicision maker will not be Baysian. The reason for this lack of external Baysianity is that the linear opinion pool is not derived from the joint probabilities using Bayes' rule.

Another consensus rule, the logarithmic opinion pool, has been proposed to overcome some of the problems with the linear opinion pool. The logarithmic opinion pool can be described by

$$C_j^* \ (\omega_j \mid Z) = \prod_{i=1}^{n} p(\omega_j \mid z_i)^{\lambda_i} \tag{5.6}$$

where λ_i , ... , λ_i are weights which should reflect the goodness of the data sources. Often it is assumed that $\sum_{i=l}^{n} \lambda_i = 1$.

In [55], the logarithmic opinion pool is given a natural-conjugate interpretation and it is shown that the logarithmic opinion pool differs from the linear opinion pool in that it is unimodal, less dispersed, and externally Baysian.

The logarithmic opinion pool treats the data sources independently. Zeros in the logarithmic opinion pool are vetoes; i.e., if any expert assigns $p(\omega_j \mid z_i) = 0$, then C_j^* $(\omega_j \mid \mathrm{Z})$ = 0. This dramatic behavior is a drawback if the density functions are not carefully estimated. It is also computationally more complicated than the linear opinion pool .

5.2.2 Geometric view of decision trees [40]

Early research in induction, such as the one described in the classic reference Learning Machine [26], was based on a geometric model of the learning task in which objects (cases) are described by vectors of real numbers. If there are N attributes, such a vector corresponds to a point in an N-dimensional Euclidean space. From this perspective, a classifier corresponds to a division of the description space into regions, each labeled with a class. An unseen case is classified by determining the region into which the corresponding point falls and assigning it to the class associated with that region. The induction task is then one of finding an appropriate partition of the description space or, equivalently, of describing the surfaces that bound each region.

This notion can easily be generalized to discrete attributes as well-such an attribute corresponds to an axis with only a fixed number of possible values. Ignoring the messy complication of unknown attribute values, it is clear that each case can still be represented as a point. The resulting description space is not generally Euclidean, however, because distances and distance relations can be altered by reordering the discrete values on such an axis or by spacing them differently.

Like any classifier, a decision tree specifies how a description space is to be carved up into regions associated with the classes. This point-and-region perspective holds many insights about induction tasks and the behavior of a learning algorithm. The geometric interpretation of classification tasks is used here to highlight some weaknesses inherent in divide-and-conquer methods and discuss ways of detecting when they affect particular tasks.

Each of the divisions that results from the sort of the test we have encountered so far corresponds to a special kind of surface in the description space, namely a hyperplane that is orthogonal to the axis of the tested attribute and parallel to other axes. This observation is crucial because the regions produced by a decision tree that uses such tests are not arbitrary – they are all hyperrectangles.

5.2.3 Drawbacks of decision trees [40]

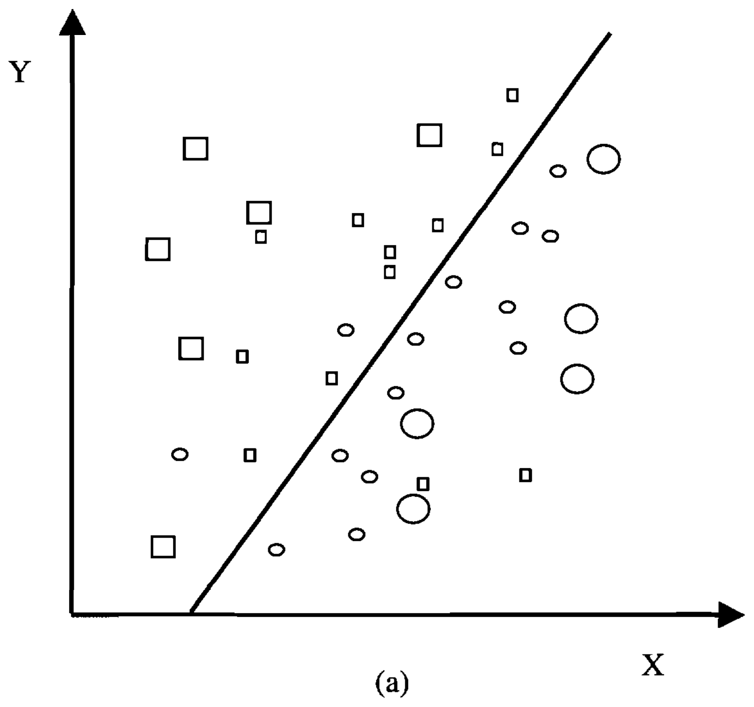
When the task at hand is such that class regions are not hyperrectangles, the best that a decision tree can do is to approximate the regions by hyperrectangles. This is illustrated by the artificial task of Fig. 5.1 in which two classes are described by two continuous attributes, X and Y. The intended division of the description space by an oblique line is shown in (a), while (b) displays the approximation to this division that is founded by decision trees. It is known that as the number of training cases is increased, the approximation of the oblique division by a collection of rectangles becomes better and better, but at the expense of a substantial increase in the number of regions.

The unlimited growth of the decision tree as more training cases are used, coupled with a more or less constant error rate on the same training cases, serves as a reminder of the condition that the surfaces bounding regions are not naturally orthogonal hyperplanes. The remedy is to look for some arithmetic combination of the attributes to be included as a new attribute. Some decision tree systems provide assistance in this direction by allowing tests that contain linear combinations of attributes. Instead of comparing on selected attribute against a threshold, these systems consider tests such as

$$\sum \omega_i \bullet A_i \geq Z \qquad (5.1)$$

And find weights $\{\omega_i\}$ to maximize the worth of a split under the prevailing criterion. The CART system allows this as an option [49]. In a similar way, linear machine decision trees [50] uses tests of this kind, but find the weights by hill-climbing. Moreover, the search for weights in the latter system is biased towards combinations of a few attributes, rather than all of them, with the idea of producing more comprehensible tests. These approaches can be useful, and would circumvent the problem of non-orthogonal hyperplanes.

However, broadening the range of possible tests in this way covers only one form of arithmetic combination of the attributes and does not address, for example, products of the attributes. To incorporate general arithmetic combinations would require the kind of extensive search, but even the limited generalization to linear combinations can slow down the process of building trees by an order of magnitude.



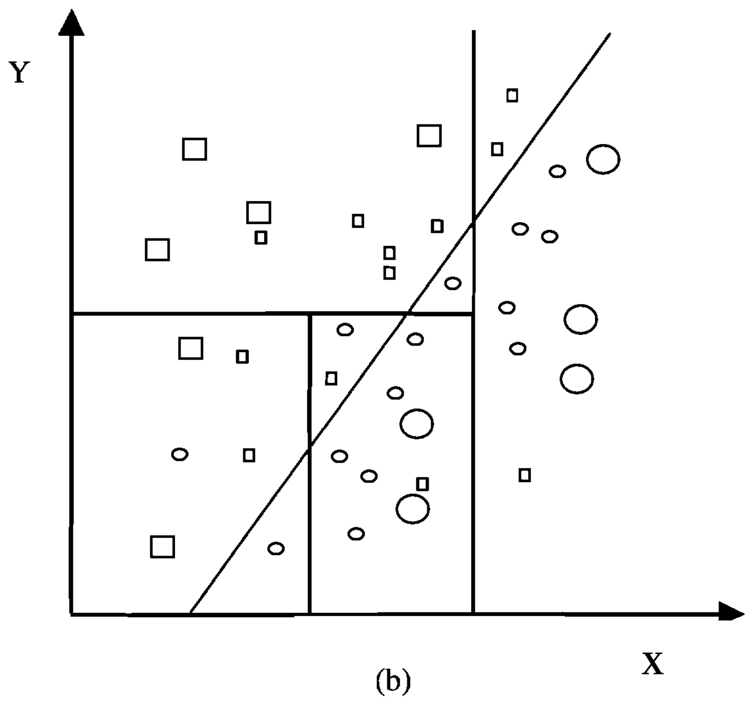


Fig. 5.1 Real and approximate divisions for an artificial work.

5.3 Hybrid Neural Network-Decision Tree System

By constructing neural networks outside of decision trees in parallel and combining the outputs of the decision trees and neural networks, synergy effect can be made. Better performance can be obtained not only in term of classification accuracy but also in term of rule extraction since decision trees are known to produce interpretable rules more easily.

5.3.1 Procedure

The hybrid neural network-decision tree system consists of a number of stages. Each stage is a pair of a neural network and a decision tree. These will be denoted as SNN, and SDT_n for the n-th stage, respectively. At the output of each stage, there is an error detection scheme which allows acceptance or rejection of input vectors. If an input vector is rejected, it is fed into the next stage. The block diagram for a 3-stage system is shown in Fig. 4.6. The flow diagram for the reject schemes is shown in Fig. 4.7. The detailed algorithm is as follows.

Training

Initialization: $n = 0$

Phase 1. Increase n by 1. Train SDT_n by a decision tree algorithm and SNN_n by a simple learning algorithm such as delta rule learning.

Phase 2. Compare the outputs of both systems. If they diagree, the input is rejected.

Phase **3.** Select the input data which are detected to give output errors; according to the PSHNN stage rejection scheme.

Phase 4. Provide the rejected inputs to the next stage.

Phase 5. Test for sufficient accuracy. If not, go to phase 1.

Testing

Tht: following describes the complete evaluation procedure of the proposed system.

Phase 1. Input the testing vector to SNN_n and SDT_n.

Phase 2. Check whether the testing vector is rejected by reject schemes.

Phase **3.** If rejected, go to Phase 1 and test again with next SNN_{n+1} and SDT_{n+1}.
If not, decide the class.

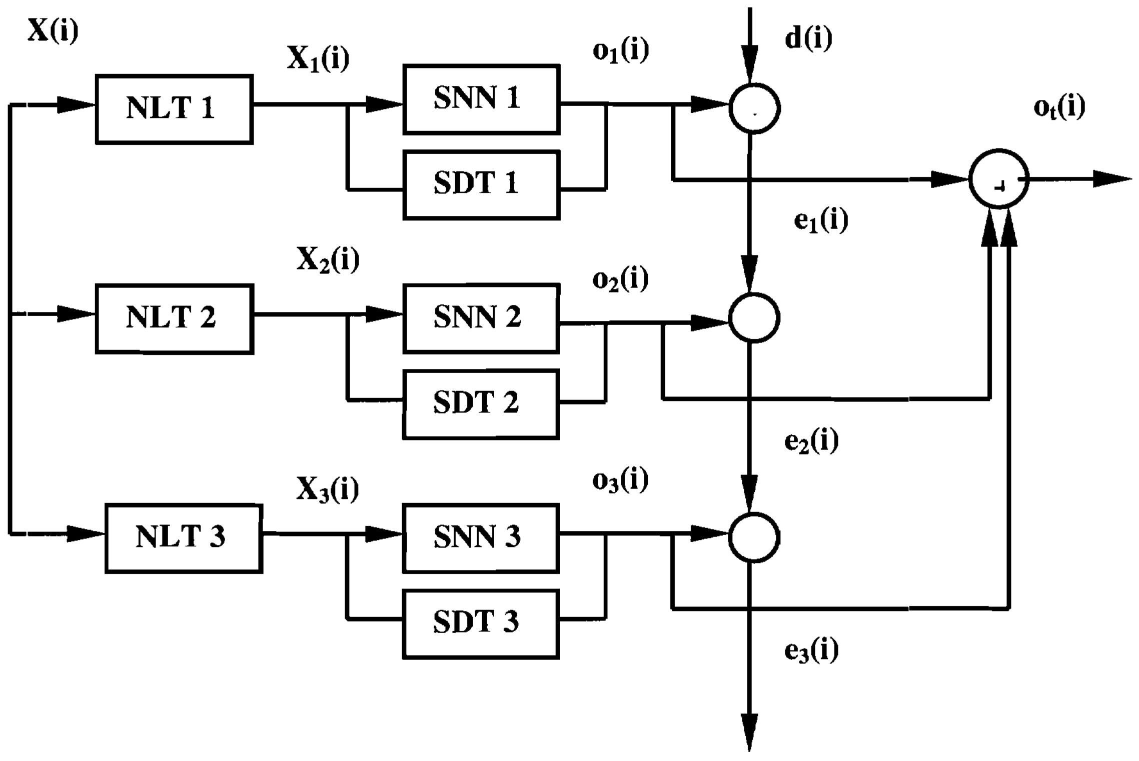


Fig. 5.2 Block diagram of a three-stage NN-DT system.

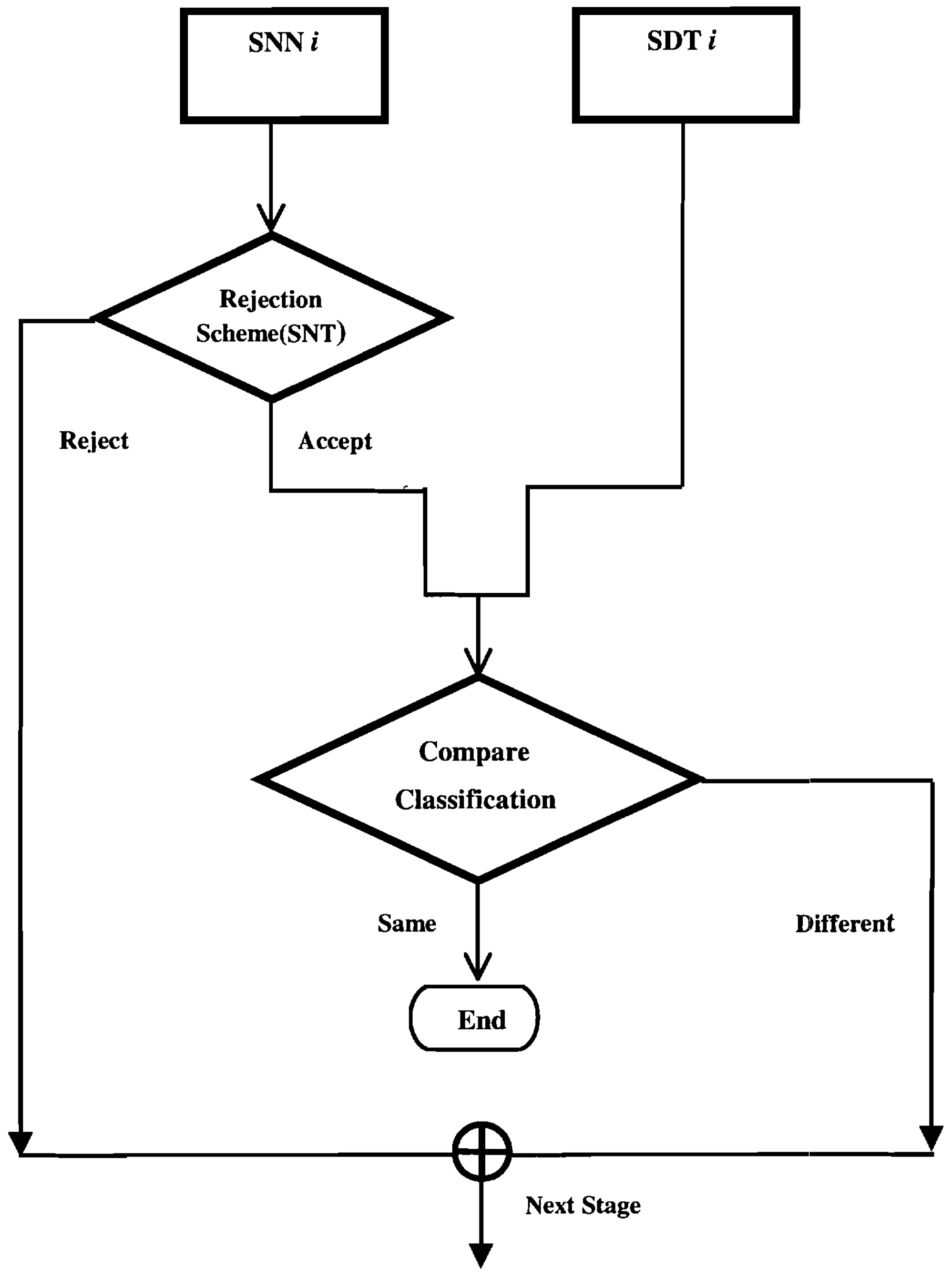


Fig. 5.3 Flow diagram for rejection schemes in NN-DT system.

5.3.2 Extracted Rules

Tables 5.1-5.12show the rules we extracted from the proposecl NN-DT system. We tested our system with various data sets and the results reveal that our proposed system is robust overally in every case.

Tables 5.1-5.3 contain the rules from the Alu and non-Alu sequnce data.
Table 5.1 shows that most of rules are extracted from the first stage of NN-DT system and Table 5.2 is for the rules from the second stage and Table 5.3 is fronn the third stage.

Tables 5.4-5.6 show the rules underlying data set for leader and non-leader sequences in the same fashion as we see in the first genomic data set.

Tables 5.7-5.9 include the rules from the synthetic Gaussian distributed data. We can see that there are smaller number of rules compared with the results of the previous data sets. As we can see, this data set is more simple and easier to classify than the previous data sets and the data input dimension is just 2 although they have continuous values. That's why our proposed system produced smaller number of rules.

Tables 5.10-5.12 represent the obtained rules from remote sensing data. There are more premises in the rules than the ones from the synthetic data because input dimension is 7 in this case.

Table 5.1 Extracted rules from first stage of NN-DT system
(Alu and non-Alu sequences, class 1: non-Alu, class 2: Alu).

```
    A3 = G      A58 = C     -> class 1
    A14 = T     A43 = C     -> class 1
    A1 = G      A3 = C      A13 = C     -> class 1
    A14 = A     A54 = C     -> class 1
    A28 = G     A31 = G     -> class 1
    A13 = G     A34 = G     A43 = G     -> class 1
    A3 = T      A54 = T     -> class 1
    A3 = A      A57 = C     -> class 1
    A3 = G      A34 = C     -> class 1
    A10 = C     A43 = C     -> class 1
    A14 = C     A43 = C     -> class 1
    A1 = G      A13 = G     A34 = T     -> class 1
    A20 = C     A59 = C     -> class 1
    A3 = C      A20 = T     A59 = C     -> class 1
    A3 = G      A58 = G     -> class 1
    A14 = T     A59 = T     -> class 1
    A41 = T     A59 = G     -> class 1
    A3 = T      A32 = T     -> class 1
    A3 = C      A14 = A     A43 = C     -> class 1
    A1 = G      A34 = T     A43 = G     -> class 1
    A3 = A      A57 = A     A58 = G     -> class 1
    A1 = G      A3 = G      A58 = T     -> class 1
    A43 = T     A63 = T     -> class 1
    A3 = A      A31 = G     -> class 1
    A2 = T      A43 = G     -> class 1
    A1 = G      A13 = G     A34 = C     A43 = G     -> class
1
    A3 = T      A54 = C     -> class 1
    A3 = A      A31 = A     -> class 1
    A1 = G      A3 = C      A43 = T     -> class 1
    A1 = G      A3 = C      A13 = A     A34 = G     -> class
1
    A14 = C     A43 = A     A59 = T     -> class 1
    A1 = T      A43 = T     -> class 1
    A3 = T      A43 = A     A54 = A     -> class 1
    A1 = G      A3 = C      A13 = T     A41 = G     -> class
2
    A3 = C      A43 = A     A59 = A     -> class 2
    A3 = G      A4 = C      A5 = T      A58 = A     -> class
2
    A3 = C      A13 = G     A34 = A     A43 = G     A49 = G
    -> class 2
    A1 = C      A15 = A     A43 = T     -> class 2
    A14 = N     -> class 2
    A43 = N     -> class 2
    A1 = A      -> class 2
```

Table 5.2 Extracted rules from second stage of NN-DT system

(Alu and non-Alu sequences, class 1: non-Alu, class 2: Allu).

```
A21 = T     ->  class 1
A1 = G      A52 = G     ->  class 1
A54 = C     ->  class 1
A5 = A      A44 = A     A52 = A     A54 = A     ->  class 2
A1 = C      A22 = A     A54 = A     ->  class 2
A11 = A     A13 = G     A43 = A     A54 = G     ->  class 2
A1 = A      A13 = G     A43 = C     A54 = G     ->  class 2
A29 = T     A52 = T     A54 = A     ->  class 2
A52 = C     A53 = T     A54 = A     ->  class 2
A38 = N     ->  class 2
```

Table 5.3 Extracted rules from third stage of NN-DT system

(Alu and non-Alu sequences, class 1: non-Alu, class 2: Alu).

```
A54 = C     ->  class 1
A57 = C     ->  class 1
A1 = G      ->  class 1
A57 = G     ->  class 1
A1 = T      ->  class 1
A1 = A      A41 = G     A57 = A     ->  class 2
A1 = A      A30 = G     A57 = T     ->  class 2
A1 = C      A22 = A     A54 = A     ->  class 2
A27 = T     A45 = A     A50 = G     ->  class 2
```

Table 5.4Extracted rules from first stage of NN-DT system

(leader and non-leader sequences, class 1: non-leader, class 2: leader).

A20 = C	A40 = G	A48 = C	-> class 2	
A6 = T	A40 = C	A49 = T	-> class 2	
A30 = G	A40 = C	A49 = C	A50 = G	-> class 2
A40 = T	A54 = G	A60 = C	-> class 2	
A9 = T	A40 = A	A52 = C	-> class 2	
A40 = G	A48 = C	A60 = G	-> class 2	
A32 = N	A40 = C	-> class 2		
A40 = A	A59 = G	-> class 2		
A40 = A	A59 = C	A60 = T	-> class 2	
A11 = G	A22 = C	-> class 2		
A60 = C	A64 = G	-> class 2		
A40 = C	A49 = G	-> class 2		
A6 = C	A40 = A	A59 = T	-> class 1	
A40 = G	A48 = A	A49 = C	-> class 1	
A10 = A	A40 = C	A44 = G	A49 = A	-> class 1
A4 = A	A40 = C	A49 = A	-> class 1	
A15 = C	A30 = C	A40 = C	A49 = C	-> class 1
A8 = C	A28 = T	A40 = G	A48 = G	-> class 1
A30 = T	A39 = A	A40 = C	A49 = C	-> class 1
A40 = A	A52 = T	A59 = A	-> class 1	
A32 = T	A40 = C	A54 = G	-> class 1	
A17 = T	A59 = C	A60 = T	-> class 1	
A6 = G	A12 = A	A40 = C	-> class 1	
A40 = A	A59 = C	A60 = A	-> class 1	
A52 = A	A59 = A	-> class 1		
A20 = T	A25 = C	A40 = G	-> class 1	
A40 = T	-> class 1			
A9 = G	A28 = A	A40 = G	-> class 1	
A40 = G	A48 = T	-> class 1		

Table 5.5 Extracted rules from second stage of NN-DT system

(leader and non-leader sequences, class 1: non-leader, class 2: leader).

```
A28 = C     A35 = C     A54 = C     ->  class 2
A28 = G     A36 = C     A43 = T     ->  class 2
A28 = C     A35 = G     A36 = C     ->  class 2
A63 = N     ->  class 2
A28 = C     A35 = A     ->  class 2
A11 = T     A28 = A     ->  class 1
A28 = T     A63 = T     ->  class 1
A47 = A     ->  class 1
A11 = A     ->  class 1
```

Table 5.6 Extracted rules from third stage of NN-DT system

(leader and non-leader sequences, class 1: non-leader, class 2: leader).

```
A4 = G      A30 = T     A35 = A     ->  class 2
A8 = C      A31 = A     A35 = G     ->  class 2
A4 = C      A35 = A     A48 = A     ->  class 2
A6 = A      A19 = A     A35 = T     A39 = A     ->  class 2
A11 = T     A35 = C     A63 = G     ->  class 2
A4 = T      A30 = T     A35 = A     ->  class 2
A21 = G     A30 = G     A35 = A     ->  class 2
A31 = T     A35 = G     A64 = C     ->  class 2
A31 = C     A35 = G     ->  class 1
A64 = A     ->  class 1
A8 = G      A31 = A     A35 = G     ->  class 1
A35 = T     ->  class 1
A1 = G      ->  class 1
A2 = A      ->  class 1
A35 = C     ->  class 1
A30 = A     A35 = A     ->  class 1
```

Table 5.7 Extracted rules from first stage of NN-DT system (synthetic data, class 1: *, class 2: o).

```
        X2 <= -3.0521      ->  class 1
        X1 <= 1.789        ->  class 1
        X1 > 1.6914        X2 > -3.0521      ->  class 2
        X1 > 1.3107        X2 > 3.5335       ->  class 2
        X1 > 0.7092        X1 <= 0.7114      ->  class 2
        X1 > 0.4919        X1 <= 0.5429      X2 <= -0.8676     ->
class 2
```

Table 5.8 Extracted rules from second stage of NN-DT system (synthetic data, class 1: *, class 2: o).

```
        X1 <= 1.6914       X2 <= -1.5333     ->  class 1
        X2 <= -3.0521      ->  class 1
        X1 <= 0.7092       ->  class 1
        X1 > 0.7114        X1 <= 1.3107      ->  class 1
        X1 > 0.7092        X1 <= 0.7114      ->  class 2
        X1 > 1.3107        X1 <= 1.3454      X2 > -1.5333      ->
class 2
        X1 > 1.6289        ->  class 2
        X1 > 1.3107        X2 > 3.5335       ->  class 2
        X1 > 0.4919        X1 <= 0.5429      X2 <= -0.8676     ->
class 2
```

Table 5.9 Extracted rules from third stage of NN-DT system (synthetic data, class 1: *, class 2: o).

```
        X1 <= 1.6914       X2 <= -1.5333     ->  class 1
        X2 <= -3.0521      ->  class 1
        X1 <= 0.7092       ->  class 1
        X1 > 0.7114        X1 <= 1.3107      ->  class 1
        X1 > 0.7092        X1 <= 0.7114      ->  class 2
        X1 > 1.3107        X1 <= 1.3454      X2 > -1.5333      ->
class 2
        X1 > 1.6289        ->  class 2
        X1 > 0.4919        X1 <= 0.5429      X2 <= -0.8676     ->
class 2
        X1 > 1.3107        X2 > 3.5335       ->  class 2
```

Table 5.10 Extracted rules from first stage of NN-DT system

(remotely sensed data, class 1: fine, class 2: fir).

```
      Aspect <= 26       ->  class 2
      Channel4 <= 15     Slope <= 19       Aspect > 33       ->
class 2
      Channell <= 25     Elevation <= 166        Slope <= 19
      Aspect > 33        ->  class 2
      Channel2 <= 13     ->  class 2
      Channel4 > 14      Aspect > 26       Aspect <= 33      ->
class 1
      Channel2 > 13      Slope > 19        Slope <= 27
      Aspect > 26 ->   class 1
      Channell > 25      Channel2 > 17     Channel4 <= 18
      Aspect > 33        ->  class 1
      Channel2 <= 16     Channel3 > 30     Aspect > 26       ->
class 1
```

Table 5.11 Extracted rules from second stage of NN-DT system

(remotely sensed data, class 1: pine, class 2: fir).

```
      Channel3 > 25      Aspect > 26       Aspect <= 33      ->
class 1
      Channel4 > 18      Aspect > 26       ->  class 1
      Aspect <= 26       ->  class 2
      Channel3 <= 25     ->  class 2
      Channel4 <= 18     Slope <= 18       Aspect > 33       ->
class 2
```

Table 5.12Extracted rules from third stage of NN-DT system

(remotely sensed data, class 1: pine, class 2: fir).

```
      Channel3 > 25      Slope > 18        Aspect > 26       ->
class 1
      Slope <= 6         Aspect <= 46      ->  class 1
      Aspect <= 26       ->  class 2
      Channel3 <= 25     ->  class 2
      Channel3 <= 29     Slope <= 18       ->  class 2
```

5.4 Experimental Results

Some experimental results are presented in order to compare the proposed scheme and other systems. Two sets of genomic sequence data, a synthetic Gaussian-distributed data, and a remote sensing data are used in the experiments.

5.4.1 Experiments with genornic sequence data 1 (Alu and non-Alu sequences)

In this experiment, we used the same data set as in the experiment described in Section 3.6. See Section 3.6 for the description of the data

The experimental results with the genomic sequence data 1 are summarized in Table 5.13 and 5.14. We observed that two proposed scheme REPENN and NNDT have better classification performance than the backpropagation network, decision trees, and the PSHNN with the delta rule.

Table 5.13 Decision results from NN-DT during testing

(Alu and non-Alu sequences, class 1: non-Alu, class 2: Alu).

```
STAGE 1 :   A3 = C      A14 = A     A43 = C     ->  class 1
STAGE 1 :   A41 = T     A59 = G     ->  class 1
STAGE 1 :   A20 = C     A59 = C     ->  class 1
STAGE 1 :   A1 = G      A3 = C      A13 = C     ->  class 1
!STAGE 1 :  A3 = G      A58 = C     ->  class 1
!STAGE 1 :  A3 = G      A58 = C     ->  class 1
!STAGE 1 :  A1 = G      A13 = G     A34 = C     A43 = G     ->
class 1
STAGE 1 :   A1 = T      A43 = T     ->  class 1
STAGE 1 :   A28 = G     A31 = G     ->  class 1
STAGE 1 :   A28 = G     A31 = G     ->  class 1
STAGE 1 :   A1 = G      A3 = C      A13 = C     ->  class 1
STAGE 1 :   A14 = A     A54 = C     ->  class 1
STAGE 1 :   A43 = T     A63 = T     ->  class 1
STAGE 1 :   A10 = C     A43 = C     ->  class 1
STAGE 1 :   A3 = A      A57 = A     A58 = G     ->  class 1
!STAGE 1 :  A14 = T     A43 = C     ->  class 1
!STAGE 1 :  A3 = G      A34 = C     ->  class 1
!STAGE 1 :  A28 = G     A31 = G     ->  class 1
STAGE 1 :   A3 = A      A57 = C     ->  class 1
STAGE 1 :   A1 = G      A3 = C      A43 = T     ->  class 1
STAGE 1 :   A1 = G      A3 = G      A58 = T     ->  class 1
STAGE 1 :   A3 = A      A57 = A     A58 = G     ->  class 1
STAGE 1 :   A14 = T     A43 = C     ->  class 1
STAGE 1 :   A3 = C      A43 = A     A59 = A     ->  class 2
:STAGE 1 :  A1 = A      ->  class 2
:STAGE 1 :  A1 = A      ->  class 2
STAGE 1 :   A1 = A      ->  class 2
STAGE 1 :   A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
;STAGE 1 :  A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
STAGE 1 :   A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
STAGE 1 :   A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
STAGE 3 :   A1 = C      A22 = A     A54 = A     ->  class 2
STAGE 1 :   A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
STAGE 1 :   A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
STAGE 1 :   A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
STAGE 1 :   A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
STAGE 1 :   A1 = G      A3 = C      A13 = T     A41 = G     ->
class 2
```

Table 5.14 Performances of classifier systems (sequence data 1).

Neural Networks	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
NN	1000	0.9220	0.7800	0.0705
DT	1000	0.8520	0.6800	0.0139
PSHNN	1000	0.9410	0.7600	0.0495
REPSHNN	1000	0.9231	0.7200	0.0442
NNDT	1000	0.9640	0.7200	0.0221

5.4.2 Experiments with genomic sequence data 2 (leader and non-leader sequences)

The data set used in this experiment is another genomic sequence data different from the one in Sec. 5.4.1. One class from leader region of human DNA sequences were taken from UniGene files in NCBI Repository. The other class represents non-leader sequence and obtained from coding region of protein coding regions and some repetitive DNA regions from NCBI Repository.

The experimental results with the genomic sequence data 2 are summarized in Table 5.15 and 5.16. We observed that NN-DT has better classification performance than the backpropagation networks, decision trees, PSHNN with the delta rule. REPENN has shown better results over backpropagation networks and decision trees but we could not say it showed better performance than the PSHNN.

Table 5.15 Decision results from NN-DT during testing;

(leader and non-leader sequences, class 1: non-leader, class 2: Leader).

```
STAGE 1 :   A52 = A    A59 = A    -> class 1
STAGE 1 :   A6 = C     A40 = A    A59 = T    -> class 1
STAGE 1 :   A40 = G    A48 = T    -> class 1
STAGE 1 :   A40 = G    A48 = A    A49 = C    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 1 :   A52 = A    A59 = A    -> class 1
STAGE 1 :   A40 = G    A48 = T    -> class 1
STAGE 1 :   A6 = C     A40 = A    A59 = T    -> class 1
STAGE 1 :   A15 = C    A30 = C    A40 = C    A49 = C    ->
class 1
STAGE 1 :   A52 = A    A59 = A    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 2 :   A11 = A    -> class 1
STAGE 1 :   A52 = A    A59 = A    -> class 1
STAGE 1 :   A20 = T    A25 = C    A40 = G    -> class 1
STAGE 1 :   A40 = G    A48 = T    -> class 1
STAGE 1 :   A40 = A    A52 = T    A59 = A    -> class 1
STAGE 1 :   A6 = C     A40 = A    A59 = T    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 1 :   A9 = G     A28 = A    A40 = G    -> class 1
STAGE 1 :   A17 = T    A59 = C    A60 = T    -> class 1
STAGE 1 :   A6 = C     A40 = A    A59 = T    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 1 :   A15 = C    A30 = C    A40 = C    A49 = C    ->
class 1
STAGE 1 :   A52 = A    A59 = A    -> class 1
STAGE 1 :   A40 = T    -> class 1
STAGE 1 :   A40 = G    A48 = A    A49 = C    -> class 1
STAGE 1 :   A10 = A    A40 = C    A44 = G    A49 = A    ->
class 1
STAGE 1 :   A40 = A    A59 = C    A60 = T    -> class 2
STAGE 1 :   A40 = G    A48 = C    A60 = G    -> class 2
STAGE 1 :   A9 = T     A40 = A    A52 = C    -> class 2
STAGE 1 :   A11 = G    A22 = C    -> class 2
STAGE 1 :   A40 = A    A59 = C    A60 = T    -> class 2
STAGE 1 :   A20 = C    A40 = G    A48 = C    -> class 2
STAGE 1 :   A6 = T     A40 = C    A49 = T    -> class 2
STAGE 1 :   A40 = A    A59 = C    A60 = T    -> class 2
STAGE 1 :   A40 = T    A54 = G    A60 = C    -> class 2
STAGE 1 :   A40 = T    A54 = G    A60 = C    -> class 2
STAGE 1 :   A11 = G    A22 = C    -> class 2
STAGE 1 :   A11 = G    A22 = C    -> class 2
STAGE 1 :   A6 = T     A40 = C    A49 = T    -> class 2
STAGE 1 :   A40 = G    A48 = C    A60 = G    -> class 2
```

Table 5.16 Performances of classifier systems (sequence data 2).

Neural Networks	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
NN	1000	0.5510	0.5320	0.4300
DT	1000	0.5190	0.4980	0.4600
PSHNN	1000	0.5500	0.3380	0.2380
REPSHNN	1000	0.5240	0.3600	0.3120
NNDT	1000	0.5440	0.3460	0.2440

5.4.3 Experiments with synthetic Gaussian distributed data

In this experiment, we used the same data set as in the experirnent described in Section **3.6.** See Section **3.6** for the description of the data set

The experimental results with Gaussian distributed data are summarized in Table 5.1'7 and 5.18. We observed that the NNDT has better classification performance than the backpropagation network, decision trees, and the PSHNN with the delta rule.

Table 5.17 Decision results from NN-DT during testing (synthetic data, class 1: *, class 2: o).

```
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X2 <= -3.0521      ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X2 <= -3.0521      ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X2 <= -3.0521      ->  class 1
STAGE 1 :   X2 <= -3.0521      ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 <= 1.789        ->  class 1
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3,0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
STAGE 1 :   X1 > 1.6914        X2 > -3.0521      ->  class 2
```

Table 5.18 Performances of classifier systems (synthetic data).

Neural Networks	No. of Testing Data	Testing Accuracy	Detection Probability	False **Alarm** Rate
NN	1000	0.9660	0.9400	0.0326
DT	1000	0.9300	0.8200	0.0640
PSHNN	1000	0.9680	0.9400	0.0305
NNDT	1000	0.9680	0.9200	0.0253

5.4.4 Experiments with remotely sensed data

In this section we describe the experimental result with a multispectral earth observation remotely sensed data covering a mountainous area in Colorado.

In this experiment, we used the same data set as in the experirnent described in Section 3.6. See Section 3.6 for the description of the data set. 2 ground cover classes were chosen to represent 2 particular classes among 10 ground cover classes and both classes are difficult to classify.

The experimental results with the remote sensing data are summarized in Table 5.19 and 5.20. We observed that the NNDT has better classification performance than the backpropagation network, decision trees, and the PSHNN with the delta rule.

Table 5.19 Decision results from NN-DT during testing

(remotely sensed data, class 1: pine, class 2: fir).

```
STAGE 1 :   Channel1 > 25     Channel2 > 17       Channel4 <= 18
      Aspect > 33       ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
STAGE 1 :   Channel2 > 13     Slope > 19          Slope <= 27
      Aspect > 26 ->  class 1
BTAGE 1 :   Channel4 <= 15    Slope <= 19         Aspect > 33
      ->  class 2
STAGE 1 :   Channel4 <= 15    Slope <= 19         Aspect > 33
      ->  class 2
STAGE 1 :   Channel4 <= 15    Slope <= 19         Aspect > 33
      ->  class 2
STAGE 1 :   Channel1 <= 25    Elevation <= 166    Slope <= 19
      Aspect > 33       ->  class 2
STAGE 1 :   Channel4 <= 15    Slope <= 19         Aspect > 33
      ->  class 2
STAGE 1 :   Aspect <= 26      ->  class 2
STAGE 1 :   Channel1 <= 25    Elevation <= 166    Slope <= 19
      Aspect > 33       ->  class 2
STAGE 1 :   Channel1 <= 25    Elevation <= 166    Slope <=: 19
      Aspect > 33       ->  class 2
STAGE 1 :   Channel4 <= 15    Slope<= 19          Aspect > 33
      ->  class 2
STAGE 1 :   Channel1 <= 25    Elevation <= 166    Slope <= 19
      Aspect > 33       ->  class 2
STAGE 1 :   Channel4 <= 15    Slope <= 19         Aspect > 33
      ->  class 2
STAGE 2 :   Aspect <= 26      ->  class 2
STAGE 1 :   Channel2 <= 13    ->  class 2
STAGE 1 :   Channel2 <= 13    ->  class 2
```

Table 5.20 Performances of classifier systems (remotely sensed data).

Neural Networks	No. of Testing Data	Testing Accuracy	Detection Probability	False Alarm Rate
NN	1000	0.5749	1.0000	1.0000
DT	1000	0.5140	0.6060	0.6120
PSHNN	1000	0.5075	0.8670	0.9928
NNDT	1000	0.5440	0.5957	0.2302

5.4.5 Experiments with general sequences

In this section we describe the experimental results with two long general sequences to prove the usability of our schemes. Two long sequences, U21730 and L34.157 were obtained for testing from GenBank. With the trained NN-DT system, we detected the leader regions. For both cases, our system found out the leader regions almost exactly.

Table 5.21 and 5.22 show the sequence files we used and Fig. 5.4 shows the testing results for sequence U21730. Fig. 5.5 is the smoothed version of Fig. 5.4. Fig. 5.6 represents zoomed-in version of non-leader region and Fig. 5.7 is for leader region. We can see the probability of leader region is almost 1 in the region of leader class in Fig. 5.7.

Fig. 5.8 through 5.11, also prove the detection of leader regions with the NN-DT system in the long sequence L34157.

Table 5.21 Contents of a test sequence file (U21730).

```
>gi|1143805|gb|U21730.1|HSU21730 Human 5'-nucleotidase (CD73) gene,
partial cds
AGCTTGAACTTGGTGCAATAAACAATCAAAGGCCATTTTAGATGTATATGAAAACAAATGGCATGGAAAT
GATATTGAAGCAGMTCTGGTAGCATCCCCAGAACAGGAATTTCAGMCTATTGCAGTGGCTTAGGTT
TAAAAGACAAGMCAACMTGATCAGTTCCCCAGTGATCAGTGATAGGTCATGTTGATACCGTGTACC
CACAGATACCAGGCAATGAGMGGCACATAATCTCTGTAGCATTCTTCCTMCATCCACAGCTTCAGT
CCAATCATGAGAAAACATCAGACMCCCMCTGAGGGATAGTATACAGMCCAGTACTGTTCWG
TGTCAAGGTTTTTGAAGACAACTGACTGAGMCTGTCACAGATTAGAGGAGGCTAAGAACACAGGW
CTAACTGCAATATGGTATCCTGAATTGGGTCCTGGAATAGAAAACGGACATTAGTGGAACAGTTGGTGAA
ATTAAAATAAAATCAGTAACACAGTTAATAGTTTCGTATCACTGTTAAGTTTTTAGTTTCATTGGTTATA
TAAGCTGTTAACATAAGGCAATATGGGTGATGGGTATGTGGGAACTCCCTTTATTACCTTTGCAACTTTT
CTGTAAGTCTAAAATCATTCTTTTTTTAAGTTAAAACAGCWCTACCAGTTCTTTTACCTGCTTCTAG
CAGGTGCTCMGAGGGACAGAGCWCCTTGCMGCATGACATTAGAGGGTGTCCCAGAAGCAAGGG
TCTTCTGTAGCCCCAGGCACTGCTGCTCCCMCTGCATGGCCTTTCTTGGCMTGATTTACAATCTCC
GAACCTCAGGTCAATGTCTTAAAAACGGGCACAGAATGTCTTTTCAACAAATGGTACTAAGAAAACTGGA
TATCCACATGAAAAAGAGTMTGTGGACTCATACCACATATGWTTAACTCMATGGATTMGGT
CTAAGAGTTTMTTATAAAACTCTTAGGAGWCATAGAGGWTCTTCCTAACATTGTATTTGGCT
ATGACTTCACTGCGACACCGAAAGCACACGCGACAAAATAAAAATAAAATAAAATAAAATAAGACCAGAA
AAGCAAAGAAGGGCAGCATCCCTTCCCGTCCTGATAGCTGCCGGATTTTACCCCAATCGTGTTTCTGCTT
C C A G G A G C T G T C C G C G C G T C G C C T C T G T T T T A T C A C T A A G
T T C G A A T G A T T T C T T A G T T T C T G A G A T T T T T T T G T C T A A G
CAGGAAGCGTCACATCTCAGTGAGCTTCGGCTCTCACAGCCCGMCTGCAAGGAGCTCCTCTGCCCGGC
CTCTCTTTACTCCTCCTCTCTGCCCCTCAGCTCGCTCATCTTTCTTCCCGCCCCCTCTCTTTTCCTTCTT
TGGTTCTTTGAAGTGATGAGCTAGCGCAACCACMCCATACATTCCTTTTGTAGAAAAACCCGTGCCTC
GFATGAGGCGAGACTCAGAGAGGACCCAGGCGCGGGGCGGACCCCTCCAATTCCTTCCTCGCGCCCCCGA
APGAGCGGCGCACCAGCAGCCGAACTGCCGGCGCCCAGGCTCCCTGGTCCGGCCGGGATGCGGCCGGTAC
CCGCTCCCCGCCGGGAACAACCTCTCCACTCTTCCTGCAGGGAGCTGGTGCCAGCCGACAGCCGCGCCAG
GCCCGCTCCGGGTACCAGGGTCGGATCGGGTGACGTCGCGAACTTGCGCCTGGCCGCCAAGCCGGCCTCC
ACGCTGAAGAAGGACCCGCCCCGGCCTTGACCCGGGCCCCGCCCCTCCAGCCGGGGCACCGAGCCCCGGC
CCTAGCTGCTCGCCCCTACTCGCCGGCACTCGCCCGGCTCGCCCGCTTTCGCACCCAGTTCACGCGCCAC
AGCTATGTGTCCCCGAGCCGCGCGGGCGCCCGCGACGCTACTCCTCGCCCTGGGCGCGGTGCTGTGGCCT
GCGGCTGGCGCCTGGGAGCTTACGATTTTGCACACCAACGACGTGCACAGCCGGCTGGAGCAGACCAGCG
AGGACTCCAGCAAGTGCGTCAACGCCAGCCGCTGCATGGGTGGCGTGGCTCGGCTCTTCACCAAGGTTCA
GCAGATCCGCCGCGCCGAACCCAACGTGCTGCTGCTGGACGCCGGCGACCAGTACCAGGGCACTATCTGG
TT'CACCGTGTACAAGGGCGCCGAGGTGGCGCACTTCATGAACGCCCTGCGCTACGATGCCATGGTAAGAC
CGAGCCGCG
```

Table 5.22 Contents of a test sequence (L34157).

>gi|1129043|gb|L34157.1|HUMSOD2TS Human manganese superoxide dismutase (SOD2) gene exons 1-2, 5' end
GGCACAAAAGAAAAGAAACGGAGCCTGTTCACTGGGTGTGGTAGACAAGGTAAACTTTTCTTTACCTCCC
ATATCCCACAACCTTGGATGTGCTCACAGTCATGGTAGTGTTTTGTMTGATGTAGCTGATGACAGGTGT
GATGTTGGAGATTCTTCTACCTGACTGCTGCTATCAGTCCTACCAGCCCCCAACGTTTGGTGCTTGTTCT
AAAGGGCATGTCCTAGGAGTCGCTTTAAAGAGCTCAGCCTCCCAATAGGAATATTTTATTATCACTAGAT
C A A G T C T T T C C A T T A C A A T G A C T G A T C T A G T C T C T G A A T C T G C C C T T T T
CTTCCAGTTCTCATAGCTAGTGCCCTTAAAAGTGACCTGCAGTACCTCCTGCTGAGACGAATGTACCAGC
TTCCTAACTAGCCTGCACTCCCTTCATCCCCCCAAGTCAGTGCCAGACCACCTTGCCTGWCCACTT
TCAGTGTGTCTCACCTCAGCAGAAATGTTTCTCAGCTTCCAATTAACAATCACATCAAACCCCTGCTCTT
GTCTGCGTTTTAAGGGTATCTATAGGCCGGGCGCCGTGGCTCCTACCTGTAATCCAGCACTTTGGAAGGC
CGAGGCGGGCAGATCACTTGAGGTCAGGCGTTCGAGACCATCCTGTGACCAACATAGTGAAACCCCGTCT
C T A C C A A A A A T A C P T A G T G G G G C G T G G A G G T G C A C G C T G T A A T T C C A G C T A C T C G G
GAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCAGAGGTTCCAGTGAGCCGACATGCGCACACAGT
ACTCCAGCCTGAGGCACAGAGCGAGGCTGTGTCTCMTAAATAAATAATAAATT —————— TAAGAG
TATCTATAACCTGGTCCCAGCCTGAATTTCCTTTTTCACCCCCAACACGTAGCCCTAGTTACATTCTTCT
GACGTCTGTAAACAAGCCCAGCCCTTCCTGTTGTGAAGCCAAGTTCAGGTGGTTCCTCTTCGCCTGACTG
TTTTCCCATTCCACTTACCGGAAGCCTAGTCATCTTCGGAGGGCTGTACGGGGGTTGCAAGAAGCAACGG
AAACGGTTCAGCACCTGCTACCTTCCATCATATTCTTTTCAATAAAGGGGCAACTCCCGCCAATGGCAGT
GTAGATTTCCTAACCTCTACACATGGAAGATTCACACCATTCAGGATTGTTGTTTAACTGTTGAGAGAGC
ACTTGATACTTAACAGCTTACTAGGCTACAAGACAGCGCAGNAAAGAATCCTCTGTTGTCCTTTTATGTT
ATCCTGAACAGTTGGTTCACAGAGTTACTGTAAACACACAAAACATGACTGCCAGGGCTTAGTAGTGAGG
AAGGTGGGAACTAGTCCTGACTCAGTTAACTGTGCCCAGGAGAAGCTGCTTAACCTCAAAGGATTTCACT
ATTACTAGAATCAATAATACCAACCCTAGGGGT ———— TAAAGATAAATGTGTGCAAATCCTGCCTGCAG
TCTCGGGCACGTCGTGGGTGTCCAAGAACTGTTCTTAGGCAGCCGGTGGGGACAAAGTCTGTGTGCCTCC
TCTCCTGGAATAGGTCCCAAGGTCGGCTTACTTGCAAAGCAAGGGTACGGCGCAAGAGTACTGAATACGG
GTTGGAAGGGCGCTGGCTCTACCCTCAGCTCATAGGCCGGCTGGGCGGCGCTGACCAGCAGCTAGGCCCC
GT'CTTCCCTAGGAACGGCCACGGGGGCCCTGGGAGGGTATGAATGTCTTTTTGCAGTGAGGCCTCTGGAC
CCCGCGGCCCCCCGGCAGCGCAACCAAAACTCAGGGGCAGCGCCGCAGCCGCCTAGTGCAGCCAGATCCC
CGCCGGCACCCTCAGGGGCGGACCGGAGGCAGGGCTTCGGGCCGTACCAACTCCACGGGGGCAGGGGCCG
CCTCCCTTCGGCCGCGCGCCACTCAAGTACGGCAGACAGGCAGCGAGGTTGCCGAGGCCGAGGCTAGCCT
GC'AGCCTCCTTTCTCCCGTGCCCATTGGGCGCGGGTGTACGGCAAGCGCGGGCGGGCGGGACAGGCACGC
ACIGGCACCCCCGGGGTTGGGCGCGGCGGGCGGCGGGGCGGGGCTCGCGGGGGGAGGGGGCGGGGCGGCGG
TGCCCTTGCGGCGCAGCTGGGGTCGCGGCCCTGCTCCCGGCGCTTTCTTAAGGCCCGCGGGCGGCGCAGG
AGCGGCACTCGTGGCTGTGGTGGCTTCGGCAGCGGCTTCAGCAGATCGGCGGCATCAGCGGTAGCACCAG
CPCTAGCAGGAGCGGCACTCGTGGCTGTGGTGGCTTCGGCAGCGGCTTCAGCAGATCGGCGGCATCAGCG
GTAGCACCAGCACTAGCAGCATGTTGAGCCGGGCAGTGTGCGGGTGAGAAGGAAGGGGACCCGGTCACGC
CCCAAGGGGCAAGGGGCTCGCGGCGGCAGGGCCTCCGCGGCAATGGCGACAGTGGCCGCACCGGGCCTGG
CGGGACCGGGGCACCTGCAGGCGGTTCTCCCGGGCAGCTGCCCGNCGGCGGCGGCCTGGAGCGGGGATCC
GCAGGGACGGGGACCGCGGGGACTCGGGGGACGCCGACGCCGCGCTTCCTCGGCAGCCCAGCCGTGCGTA
GACGGTCCCGCGGTACGCTGACTGACCGGGCTGTGCTTTCTCGCTTTCAGCACCAGCAGGCAGCTGGCTC
CGGCTTTGGGGTATCTGGGCTCCAGGCAGAAGCACAGCCTCCCCGACCTGCCCTACGACTACGGCGCCCT
GGAACCTCACATCAACGCGCAGATCATGCAGCTGCACCACAGCAAGCACCACGCGGCCTACGTGAACAAC
CTGAACGTCACCGAGGAGAAGTACCAGGAGGCGTTGGCCAAGGGTAGGTCCAGGCTGAGCGGCGGGAGGC
AGTCCCCGGCAGAGGCGACCCCAGGGAGCCAGGCCCCATACGGACGGGCCTCTCCGT'GGAGGAGAACTCG
CTTCGTATTTGTACCGGTTCCGAGTTTTCCATGGGCACGATAGTCTCTCTTTTAAACACATGGTCTACCT
CATTGTGAAGGAGTGCCTCGATGGGTTTGAACACACTTCTGTCATCTCAGGGAACTTGGGGTCCTGCGAA
GGAGCTTGCCTTACTGTTGTGAGCCACATTCCGTTACACATATTGCCAGCACTGGTGAATTGTAGGGCCT
GAAAAGAAAGCTCTACTGTGTCACTCGTTTTTTTTGCAAAATTGAAATTGTTCTTGTTGTATAATGTGCT
TGGGAATGTTGG

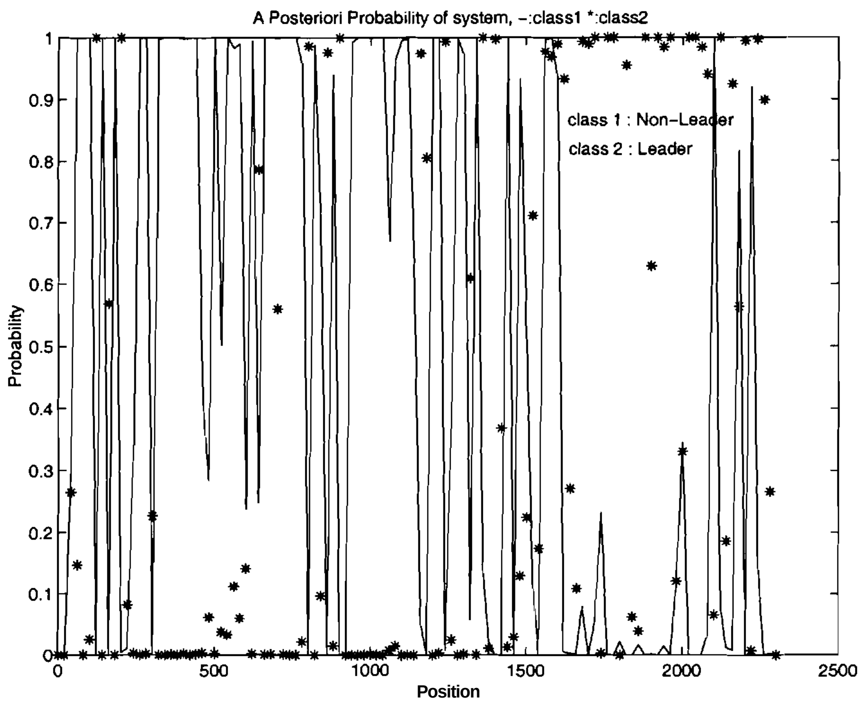


Fig. 5.4 Detection of leader region in U21730.

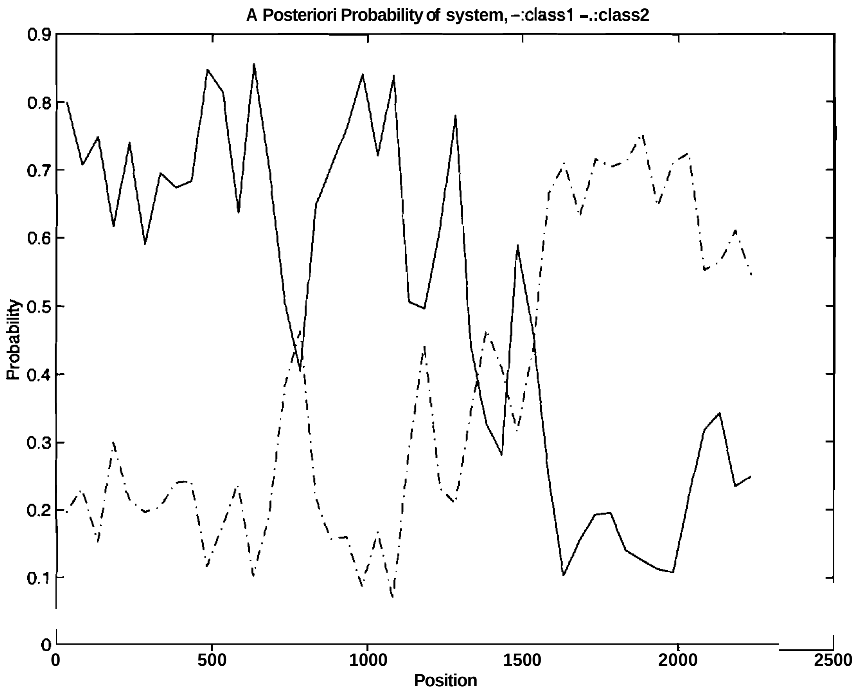


Fig. 5.5 Smoothed result of detection in U21730

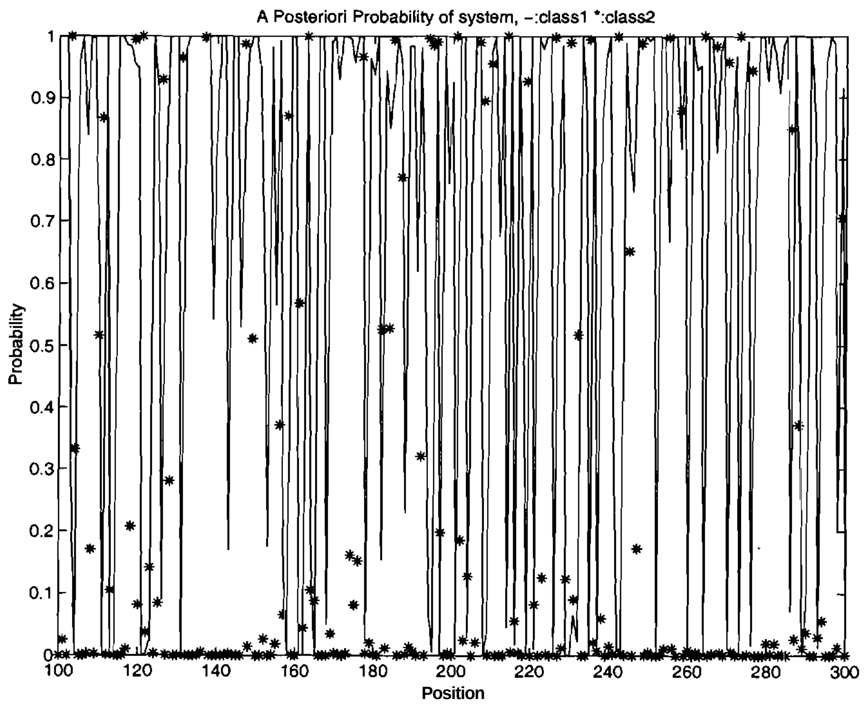


Fig. 5.6 Detailed result for non-leader region inU21730.

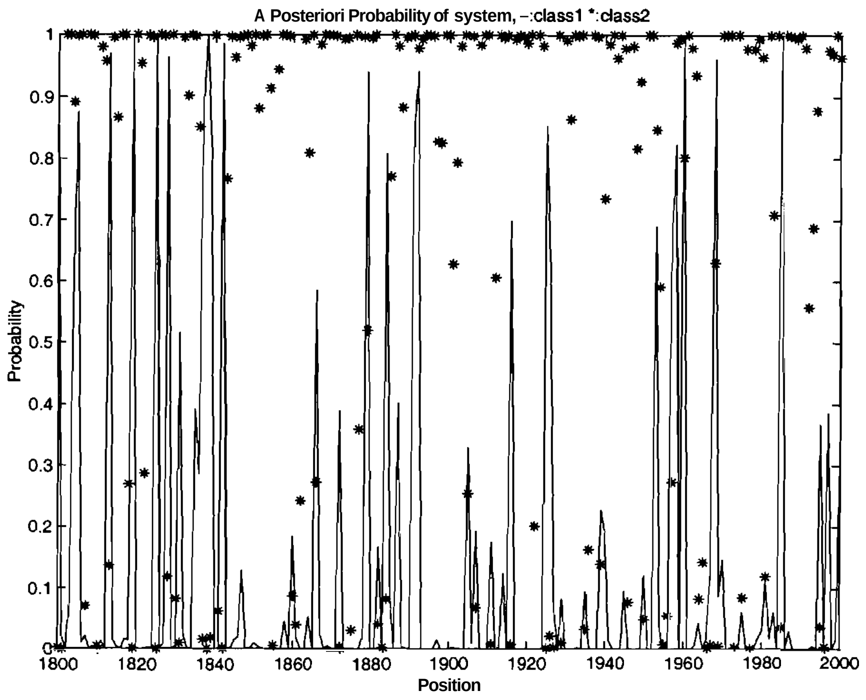


Fig. 5.7 Detailed result for leader region in U21730.

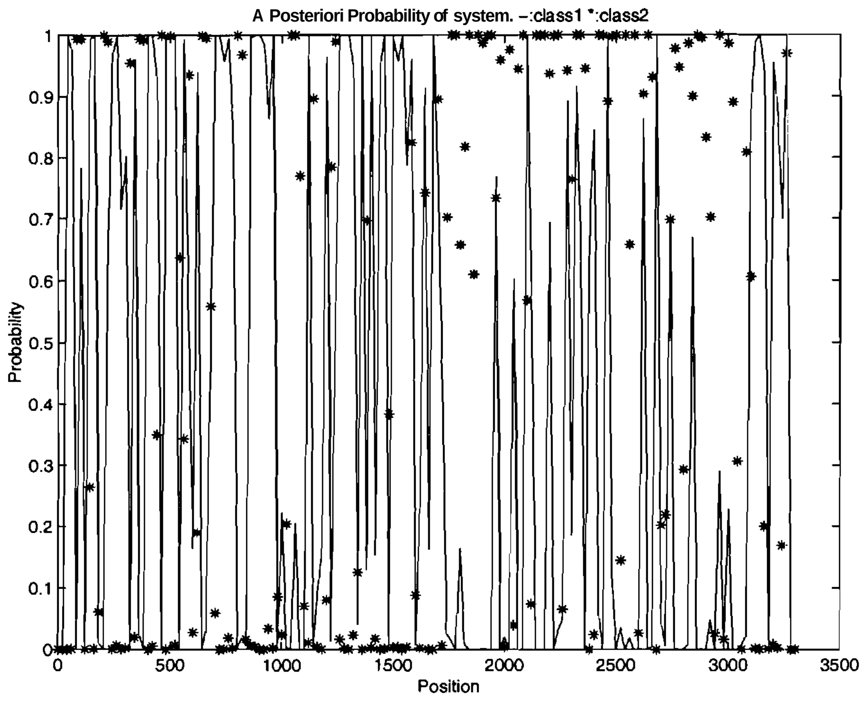


Fig. 5.8 Detection of leader region in L34157.

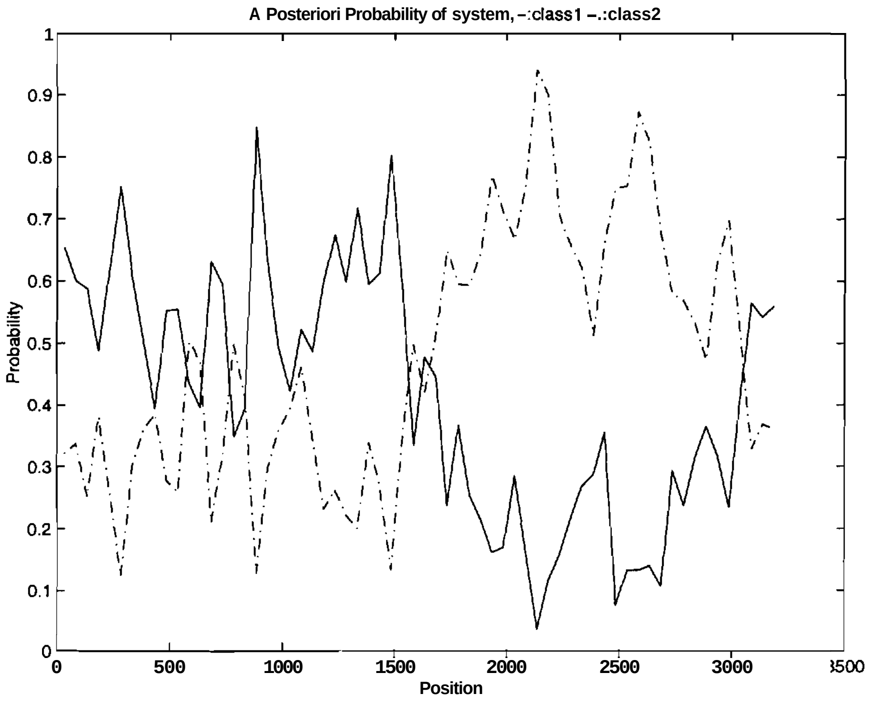


Fig. 5.9 Smoothed result of detection in L34157

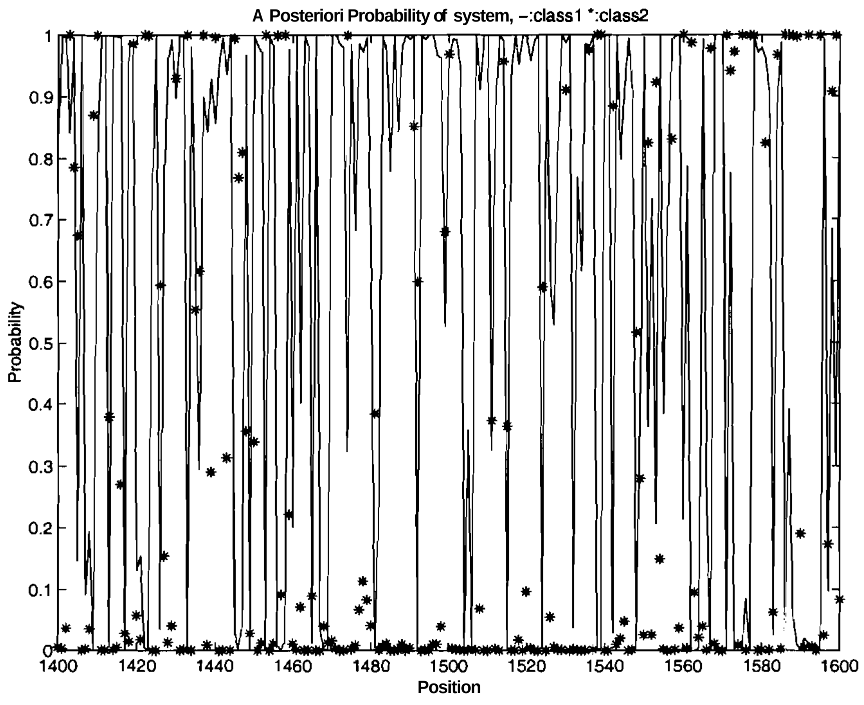


Fig. 5.10 Detailed result for non-leader region in L34157.

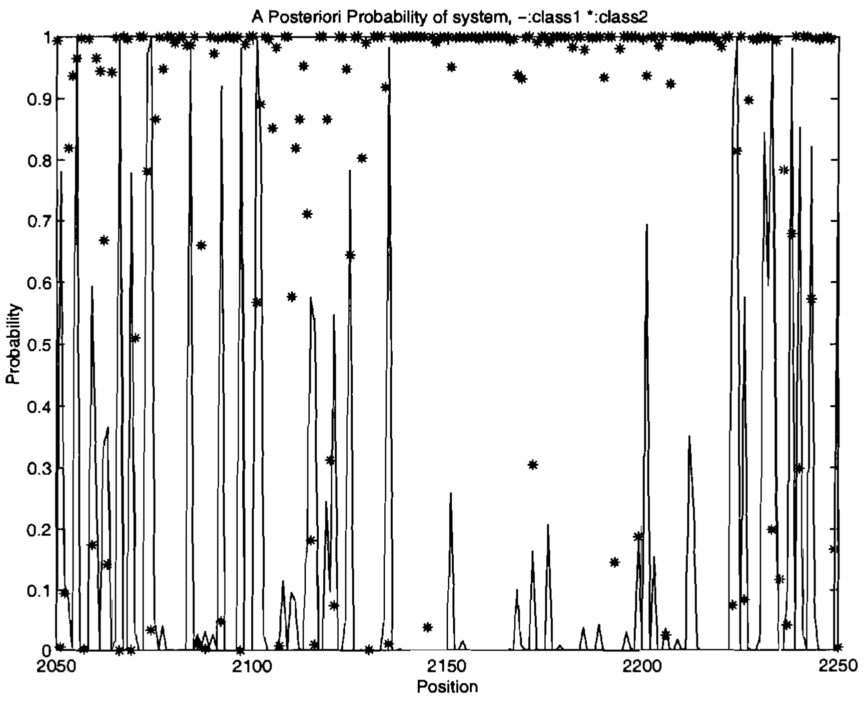


Fig. 5.11 Detailed result for leader region in L34157.

5.5 Conclusions

In this chapter, we introduced another new rule extraction scheme. Neural networks together with decision trees were used based on parallel self-organizing hierarchical neural network paradigm. Neural networks were known to be more accurate in classification than decision trees and decision trees are easier to extract interpretable rules than neural networks. We utilized the advantages of both algorithms. Neural networks and decision trees were constructed in parallel, and the outputs from the both systems were combined for final decisions. If the results from neural networks and decision trees disagree, decision is postponed to next stages like in PSHNN. By using both systems in that way, synergy effect was made. The proposed system shows better performance not only in terms of classification accuracy but also in terms of rule extraction.

6. CONCLUSIONS

In this thesis, we present neural network and hybrid neural network-decision tree methods for rare event detection and rule extraction. We developed two sample stratification techniques for detecting rare events by neural networks. First method used stratifying coefficients which are multiplied by the weighted sum of the derivatives during the backward pass of training. Second method used bootstrap aggregating. After training neural networks with multiple sets of bootstrapped examples of the rare event classes and subsampled examples of common event classes, we perform multiple voting for classification. These two schemes make rare events have better chance of being included in the sample for training and improve the classification accuracy of neural networks. The experimental performance of the two schemes using human DNA as well as two other data sets indicates that proposed schemes have the potential of significantly improving accuracy of neural networks to recognize rare events.

The second part of the thesis is on the development of rule extraction algorithms from neural networks and decision trees. For neural networks to gain more popular user acceptance and to enhance their overall utility as learning and generalization tools, it is desirable to add explanation capability. The suitability of each approach depends on the network type and architecture, complexity, the application nature, inputs, and the required transparency level. Specifically it is much helpful in the field of genomic sequence research since researchers need some explanation of the results from neural networks to utilize them for their further research. In order to make this possible, we specifically developed new rule extraction algorithms using parallel self-organizing hierarchical neural networks and decision trees. These methods are able to extract meaningful rules for each data set, where no pre-existing rules are available. The rules extracted by the proposed schemes are efficient, comprehensible and powerful. Like the training phase for classification, it is very hard to find techniques for extracting rules from neural networks for rare events. It is important to mention that obtaining all possible combinations of rules is NP-hard and a feasible alternative is often to extract key rules that cover most of the concepts of the application domain like our proposed schemes. The schemes we have developed force the overall system to be highly accurate in classification while generating simple rules.

For future research, we continue to investigate more efficient detection methods for rare events. One of them will be on acceptable minimum data size for rare event detection. We cannot make up for any small amount of data. Under a certain level of scarcity, we cannot obtain the desired results even though we use the proposed techniques. Another research direction will be towards investigating the relationship between the performance and the number of bootstrap replicates. We cannot increase the number of replicates for better performance without considering complexity problems. We want to find the reasonable bound for the number of replicates considering the performance and the complexity of neural networks.

More progress is also needed in determining when an adequate set of rules has been extracted. Another important issue that needs to be investigated is how the extracted rules can be used for knowledge refinement and truth maintenance of domain knowledge. Finally, NNDT together with stratification scheme is a strong candidate for better performance.

LIST OF REFERENCES

[1] E. M. Crowley, K. Roeder, and M. Bina, "A statistical model for locating regulatory regions in genomic DNA," *J.Mol.Biol.*, vol. 268, pp. 8-14, 1997.

[2] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," Psychological Rev., vol. 65, pp. 368-408, 1958.

[3] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in 1960 *IRE* WESCON Conv. Rec., New York IRE, 1960, pp. 96-104.

[4] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Parallel Distributed Processing, vol. 1: Foundations, Cambridge, MA: MIT Press, 1986.

[5] C. R. Cantor, "Orchestraing the human genome project," Science., vol. 248, pp. 49-51, 1990.

[6] R.J Robbins, "Challenges in the human genome project," ZEEE Trans. Engineering in Biology and Medicine, vol. 34, no. 7, pp. 25-34, 1992.

[7] M. Reczko and S. Suhai, "Applications of artificial neural networks in genome research," in Computational Methods in Genome Research, 1984, pp. 191-207.

[8] G. D. Stormo, T. D. Schneider, L. M. Gold, and A. Ehrenfeucht, "Use of the perceptron algorithm to distinguish translational initiation sites in E. coli," Nucl. Acids Res., vol. 10, pp. 2997-3011, 1982.

[9] A. V. Lukashin, V. V. Anshelevich, B. R. Amirikyan, A. I. Gragerov, and M. D. Frank-Kamenetskii, "Neural network models for promoter recognition," J. of Biomol. *Strucy. &* Dyn., vol. 6, pp. 1123-1134, 1989.

[10] B. Demeler and G. Zhou, "Neural network optimization for Escherichia coli promoter prediction," Nucl. Acids Res., vol. 19, pp. 1593-, 1991.

[11] M. Kudo, Y. Lida, and M. Shimbo, "Syntactic pattern analysis of 5' splice site sequences of mRNA precursors in higher eucaryote genes," *CABIOS*, vol. 3, pp. 319-324, 1987.

[12] K. Nakata, M. Kaneshisa, and C. DeLisi, "Prediction of splice juction in mRNA sequences," Nucl. Acids Res., vol.13, pp. 5327-5340, 1985.

[13] S. Brunak, J. Engelbrecht, and S. Knudsen, "Prediction of human mRNA donor and acceptor sites from the DNA sequences," *J.Mol.Biol.*, vol.. 220, pp. 49-65, 1991.

[14] E. C. Uberbacher and R. Mural, "Locating protein-coding regions in human DNA sequences by multiple sensor-neural network approach," *Proc.Natl.Acad.Sci.*, vol. 88, pp. 11261-11265, December 1991.

[15] R. Farber, A. Lapides, and K. Sirotkin, "Determination of eukaryotic protein coding regions using neural networks and information theory," *J.Mol.Biol.*, vol. 226, pp. 471-479, 1992.

[16] N. Qian and T. J. Sejnowski, "Predicting the secondary structure of globular proteins using neural network models," *J.Mol.Biol.* vol. 202, pp. 865-884, 1988.

[17] H. Bohr, J. Bohr, S. Brunak, J. M. R. Cotterill, B. Lautrup, L. Norskov, H. O. Olsen, S. B. Perterson, "Protein secondary structure and hoinology by neural networks," FEBS Lett., vol. 214, pp. 228-233, 1988.

[18] B. Rost, C. Sander, "Prediction of protein secondary structure at better than 70 percent accuracy," *J.Mol.Biol.* vol. 232, pp. 584-599, 1993.

[19] J. D. Hirst and M. J. E. Sternberg, "Prediction of ATP-binding motifs: A comparison of a perceptron-type neural network and a consensus sequence method," ***Protein Engineering,*** vol. 4, pp. 615-623, 1991.

[20] E. A. Ferran and P. Ferrara, 'Topological maps of protein sequences," *Biol.Cybern.* vol. 65, pp. 451-458, 1991.

[21] P. M. Hahn and M. C. Jeruchim, "Developments in the theory and application of importance sampling," ***IEEE Trans. Commun.*** vol. 34, no. 7, pp. 15-719, 1986.

[22] D. J. Monro, O . K. Ersoy, M. R. Bell, and J. S. Sadowsky, "Neural network learning of low-probability events," ***IEEE Trans. Aerosp. Electron. Syst.*** vol. 3, no. 3, pp. 898-910, July 1996.

[23] W. Choe, O. K. Ersoy, and M. Bina, "Detection of rare events by neural networks," ***in Proceedings of the Artificial Neural Networks in Engineering Conference,*** vol. 8, November 1998, pp. 5-10.

[24] M. C .Jeruchim, P. Balaban, and K. S. Shanmugan, ***Simulation of Communications Systems,*** New York, NY: Plenum, 1992.

[25] A. M. Zoubir and B. Boashash, "The bootstrap and its application in signal processing," ***IEEE Signal Processing Magazine,*** pp. 56-76, January 1999.

[26] B. Efron, ***The jackknife, the Bootstrap, and other Resampling Plans,*** SIAM NSF-CBMS, Monograph 38, 1982.

[27] N. Nilsson, *Learning Machines,* New York, NY: McGraw-Hill, 1965.

[28] L. K. Hansen and P. Salamon, "Neural network ensembles," *ZEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 12, no. 10, pp. 993-1001.,October 1990.

[29] H. Valafar and O. K.Ersoy, "Parallel, self-organizing, consensual neural network," School of Electrical Engineering, Purdue University, Tech. Rep. TR-EE 90-56, 1990.

[30] J. A. Benediktsson, J. R. Sveinsson, O. K. Ersoy, and P. H. Swain, "Parallel consensual neural networks," *ZEEE Trans. Neural Networks,* vol. 8, no. 1, pp. 54-64, 1997.

[31] D. W. Opitz and J. W. Shavlik, "Generating accurate and diverse members of a neural networks ensemble," *Advances in Neural Information Processing System 8,* MIT Press, 1996.

[32] K. Tumer and J. Ghosh, "Theoretical foundations of linear and order statistics combiners for neural pattern classifiers," The Computer and Vision Research Center, The University of Texas at Austin, Tech. Rep. TR-95-02-98, 1995.

[33] Y. Freund, and R. E. Schapire, "A decision-theoretic generalization of on-line learning and application to boosting," in *Computational Learning Theory: Second European Conference, EuroCOLT '95,* 1995, pp. 23-27.

[34] J. Jurka, ***Repbase** update.* Genetic Information Research Institute, http://www.girinst.org/~server/repbase.html, 1997.

[35] C. Yan, C. Chen, and L. Chang, "Modeling of watershed flood forecasting with time series artificial neural network algorithm," in Proc. Water Resources Engineering Conference, vol. 1, pp. 903-908, 1996.

[36] K. Fukunaga, Introduction to Statistical Pattern Recognition, 2nd ed. San Diego, CA: Academic Press, 1990.

[37] R. Andrews, J. Diederich, and A. Tickle, "A survey and critique of technique for extracting rules from trained artificial neural networks," Neurocomputing Research Center, QUT, Australia, Tech. Rep., 1995.

[38] L. M. Fu, "Rule generation from Neural Networks," IEEE *Trans.* Sysems, Man, and Cybernetics, vol. 24, no. 8, pp. 1114-1124, August 1994.

[39] G. G. Towell and J. W. Shavlik, "The extraction of refined rules from knowledge-based neural networks," Machine Learning, vol. 13, no. 1, pp.7 1-101, 1993.

[40] R. Setiono and H.Liu, "Symbolic representation of neural networks," IEEE Computer, pp. 71-77, March 1996.

[41] J. R. Quinlan, C4.5: Programs for Machine Learning, San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[42] K. Saito and R. Nakano, "Medical diagnostic expert systenn based on DPD model," in Proceedings of IEEE International Conference on Neural Networks, vol. 1, 1988, pp. 255-262.

[43] S. Sestito and T. Dillon, "Automated knowledge acquisition of rules with continuously valued attributes," In Proceedings of *Twelfth* International

Conference on Expert Systems and their Applications *(AVIGNON)*, pp. 645-656, May 1995.

[44] M. W. Craven and J. W. Shavlik, "Using sampling and queries to extract rules from trained neural networks," In Machine Learning: *Proceedings* of the Eleventh International Conference, 1994.

[45] **A.** B. Tickle, M. Orlowski, and J. Diederich, "DEDEC:decision detection by rule extraction from neural networks," Neurocomputing Research Center, QUT, September 1994.

[46] S. Horikawa, T. Furuhashi, and Y. Uchikawa, "On fuzzy modeling using fuzzy neural networks with back-propagation algorithm," IEEE Trans. Neural Networks, vol. 3, no. 5, pp. 801-806, September 1992.

[47] C. W. Omlin and C. L. Giles, "Extraction of rule from discrete-time recurrent neural networks," Neural Networks, vol. 9, no. 1, 1996.

[48] 0 . K. Ersoy and D. Hong, "Parallel self-organizing hierarchical neural networks," IEEE Trans. Neural Networks, vol. 1, no. 2, pp. 167-178, 1990.

[49] S-W. Deng and 0 . K. Ersoy, "Parallel self-organizing hierarchical neural networks with continuous inputs and outputs," IEEE Trans. Neural Networks, vol. 6, no. 5, pp. 1037-1044, 1995.

[50] L. Breiman, J. Friedman, R. Olshen, and C. Stone, Classification and Regression Trees, Wadsworth International Group, 1984.

[51] C. E. Brodley, and P. E. Utgoff, Linear machine decision trees. Dept. of Computer Science, University of Massachusetts at Amherst, Tech. Rep. 10, 1991.

[52] J. A. Benediktsson and Philp H. Swain, " Consensus theoretic classification methods, " *IEEE* Trans. Systems Man and *Cybernetics*, vol. 22, no. 4, pp. 688-704, July/August 1992.

[53] C. Berenstein, L. N. Kanal, and D. Lavine, "Consensus rule," in Uncertainty in Artificial Intelligence, L. N. Kanal and J. F. Lemmer (eds.), New York: North Holland, 1986.

[54] R. F. Brodely, Studies in Mathematical Group Decision Theory, Ph.D. Thesis, University of California, Berkely, 1979.

[55] C. Genest and J.V.Zidek, "Combining probability distributions: A critique and annotated bibliography," Statistical Science, vol. 1 no. 1, pp. 114-118, 1986.

[56] R. L. Winkler, "Combining probability distributions from dependent information sources," Management Sciences, vol. 27, pp. 479-488, 1981.

[57] R. A. Jacobs, "Methods for combining experts' probability assesments," Neural Computation, vol. 3, pp. 867-888, 1995.

[58] D. H. Wolpert, "Stacked generalization," Neural Networks, vol. 5, pp. 241-259, 1992.

[59] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter, "The multilayer perceptron as an approximation to a bayes optimal discriminant function," *IEEE* Trans. Neural Networks, vol. 1, no. 4, pp. 296-298, December 1990.

[60] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," ZEEE Trans. on Systems, Man and Cybernetics, vol. 21, no. 3, pp. 660-674, 1991.

[61] J. R. Quinlan, "Induction of decision trees," Machine Learning, pp.81-106, 1986.

[62] I. Taha and J. Gosh, "Symbolic interpretation of artificial neural networks," University of Texas at Austin, Tech. Rep. TR-97-01-106, 1996.