On Neurobiological, Neuro-Fuzzy and Statistical Pattern Recognition Techniques

Anupam Joshi
Narendran Ramakrishnan
Elias N. Houstis
Purdue University, enh@cs.purdue.edu
John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
96-004
ON NEUROBIOLOGICAL, NEURO-FUZZY
AND STATISTICAL PATTERN
RECOGNITION TECHNIQUES

Anupam Joshi
Narendran Ramakrishnan
Elias N. Houstis
John R. Rice

Purdue University
Department of Computer Sciences
West Lafayette, IN 47907

CSD TR-96-004
January 1996
On Neurobiological, Neuro-Fuzzy and Statistical Pattern Recognition Techniques*

Anupam Joshi, Narendran Ramakrishnan, Elias N. Houstis and John R. Rice
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 -1398, USA.
Phone: (317)-494-7821, Fax: (317)-494-0739
Email: {joshi,naren,enh,jrr}@cs.purdue.edu

Abstract

In this paper, we propose two new neuro-fuzzy schemes, one for classification and one for clustering problems. The classification scheme is based on Simpson's Fuzzy Min Max method, and relaxes some assumptions he makes. This enables our scheme to handle mutually non exclusive classes. The neuro-fuzzy clustering scheme is a multiresolution algorithm that is modeled after the mechanics of human pattern recognition. We also present data from an exhaustive comparison of these techniques with neural, statistical, machine learning and other traditional approaches to pattern recognition applications. The data sets used for comparisons include those from UCI's machine learning repository. We find that our proposed schemes compare quite well with the existing techniques, and in addition offer the advantage of one pass learning and on line adaptation.

Keywords: Classification, Clustering, Neuro-Fuzzy Systems, Multiresolution, Vision Systems, Overlapping Classes, Comparative Experiments.

*This work was supported in part by NSF awards ASC 9404859 and CCR 9202536, AFOSR award F49620-92-J-0069 and ARPA ARO award DAAH04-94-G-0010
1 Introduction, Background and Related Work

We begin this paper, to paraphrase the popular song, at the very beginning. It does in this case happen to be a very good place to start, considering the interdisciplinary audience that is the target of this issue. Neural Networks represent a computational[34] approach to intelligence, as contrasted with the traditional, more symbolic approaches. The idea of such systems is mostly credited to have arisen out of the work of the psychologist Donald Hebb[20] (and after whom a class of learning techniques is referred to as Hebbian). Despite the pioneering early work of McCulloch & Pitts[37] and Rosenblatt[48], the field was largely ignored through most of sixties and seventies, with researchers in AI mostly concentrating on symbolic techniques. There are many reasons cited for this, ranging from a lack of appropriate computational hardware to the work of Minsky and Papert which showed limitations of a class of neural networks (single layer perceptrons) popular then. A result in part to the failure of Good Old Fashioned AI (GOFAI[5]), and also to the development of VLSI and parallel computing, interest in Neural Networks as an alternate mechanism to investigate, understand, and duplicate intelligence was revived in the mid 80’s. In the past decade, there has been a phenomenal growth in the published literature in this field, and a large number of conferences are now held in the area[34].

While a segment of researchers views neural networks as mechanisms to study intelligence (as evidenced by the title of the famous text by McClelland and Rumelhart[50]—“Explorations in the Microstructure of Cognition”), most of the published literature in the area sees neural networks as a tool to solve problems in science and engineering. Most of these problems involve pattern recognition in one form or another. Neural networks have been used for everything from speech recognition to image recognition to SAR/Sonar data classification to stock market tracking, and so on ad nauseam. These problems involve both classification (supervised learning) and clustering (unsupervised learning). This has lead, over the past 5 years or so, to many researchers trying to investigate the links between neural network based techniques and traditional, statistical pattern recognition techniques. One of the first efforts in this direction was the seminal text by Jain & Sethi[54]. Since then, this topic has aroused considerable interest, and has seen many discussions—some acrimonious, between those who feel that NNs are old wine is new bottles, and those who feel that they represent a new paradigm. As any follower of the newsgroup comp.ai.neural-nets knows, this debate occurs there almost every six months, often triggered by an innocent question from a “newbie”.

However, several scholarly works have appeared in the last few years discussing the links between statistical and ANN based PR. In an excellent overview published in Statistical Science, Cheng & Titterington[8] described some of the relationships that exist between ANNs and SPR techniques. Responses to their article by, amongst others, Ripley[46], Amari[2] and McClelland[36] also commented on these relationships and suggested avenues for potential cross disciplinary work. Warren Sarle[52] has described how some of the simpler neural network models can be described in terms of, and implemented by, standard statistical techniques. Ripley’s work[45, 47] is also along the same lines, and also presents some empirical results comparing MLP networks trained with different algorithms with nonparametric discriminant techniques. Balakrishnan et al. [3] report comparisons of Kohonen feature maps with traditional clustering techniques such as K-means. We refer the reader to these for a scholarly and detailed exposition of the relationships between SPR and ANNs.

An area that has remained relatively unexplored in this interdisciplinary context is the use of ANN techniques that are closely related to biological neural systems. Specifically for pattern recognition, the human visual system can outperform most computer systems on recognition and identification tasks. Part of this capability comes from the human ability to classify and categorize. There have been extensive studies dealing with this subject done by psychologists. The most common models of human abilities use a threefold process. First, some metric of distance is defined on the space of the input (stimuli). Then, an exponentiation is used to convert these to measures of similarity between the stimuli. Finally, a similarity choice model is used to determine the probability of identifying one stimulus with another. We refer the reader to[10] for a detailed exposition. Such work is of increasing interests to computer scientists, especially in the domain of content based lookup of large image databases. However, in the visual pattern recognition domain, a large part of the recognition and identification ability of humans is dependent on the particular visuomotor configurations. Specifically, over the last decade or so, the concept of multiresolution processing has gained considerable interest in the vision community[23]. This technique uses multiple representations of the same input at different resolutions to do processing. These multiple representations are obtained by blurring the image with Gaussian kernels of differing widths. The notion of hierarchical representations also gets support from Neurophysiological data. Enroth-Cugell[13] showed as far back as the 60’s that the retinal processing being done by a Cat’s ganglion cells can be likened to a difference of Gaussians. Marr & Hilderral[35] showed that even for human retinal processing, a similar Laplacian of Gaussian (LOG) operator could be defined. Joshi and Lee[24] showed that a neural network could be trained to produce a connection pattern similar to that found in the retina, and that the mathematical operation performed by such a network would be similar to the LOG operator. Work by Daugman[10] has suggested the use of filter based descriptions. Several studies have shown that there are as many as six
channels tuned to different spatial frequencies that carry different representations of the visual input to the higher layers in the occipital cortex. Another interesting property of the visual system is the increasing size of the receptive fields of the cells as we go up the processing layers in the visual cortex, and up to the IT regions[32, 22]. The receptive field of a cell refers to the region in the photoreceptor layer whose activity influences it. The receptive field of a cell in the Lateral Geniculate Nucleus, for instance, will be larger than that of a retinal ganglion cell.

This kind of an architecture has given rise to several multiresolution based algorithms, implemented in a special parallel architecture referred to as a pyramid. Each processor in a pyramid receives input from some processors in the lower layers, and feeds its output to cells in the upper layer. The most common kind of a pyramid is a non overlapped quad pyramid, where each processor receives input from 4 processors in the layer below it. The reader is referred to [23] for details. Several recent works, including [41], have shown how such a multiresolution based model can successfully account for human visual processing performance. Interestingly, multiresolution approaches are similar to the agglomerative schemes for clustering found in statistics.

In this paper, we propose new Neuro-Fuzzy classification and clustering techniques based on the multiresolution idea. The classification scheme is a modification of the scheme proposed by Simpson[55]. These techniques are described in the next section. We then present a comparison of various statistical, neural and neuro fuzzy techniques for both classification and clustering, including the ones proposed here. The data sets used are representative samples obtained from the Machine Learning repository at the University of California at Irvine. One of the data sets used, which contains overlapping classes, is from our own work dealing with the creation of Problem Solving Environments[17, 26].

2 Neuro-Fuzzy Schemes

2.1 Classification

We have developed a new algorithm for classification, which is a modification of the work done by Simpson[55]. The basic idea is to use fuzzy sets to describe pattern classes. These fuzzy sets are, in turn, represented by the fuzzy union of several n-dimensional hyperboxes. Such hyperboxes define a region in n-dimensional pattern space that contains patterns with full-class membership. A hyperbox is completely defined by its min-point and max-point and also has associated with it a fuzzy membership function (with respect to these min-max points). This membership function helps to view the hyperbox as a fuzzy set and such “hyperbox fuzzy sets” can be aggregated to form a single fuzzy set class. This provides degree-of-membership information that can be used in decision making. The resulting structure fits neatly into a neural network assembly. Learning in the fuzzy min-max network proceeds by placing & adjusting the hyperboxes in pattern space. Recall in the network consists of calculating the fuzzy union of the membership function values produced from each of the fuzzy set hyperboxes.

Simpson’s method assumes that the pattern classes underlying the domain are mutually exclusive and that each pattern belongs to exactly one class. But the pattern classes that characterize problems in many real world domains are frequently not mutually exclusive. For example, consider the problem of classifying geometric figs into classes such as polygon, square, rectangle etc. Note that these classes are not mutually exclusive (i.e., a square should be classified as a square and a rectangle and a polygon). It is possible to apply Simpson’s algorithm to this problem by first ‘reorganizing’ the data into mutually disjoint classes such as ‘rectangles that are not squares’, ‘polygons that are not rectangles’, and ‘polygons’ etc., but this strategy does not reflect the natural overlapping characteristics of the underlying base classes.

Thus Simpson’s algorithm fails to account for a situation where one pattern might belong to several classes. Also, the only parameter in the Simpson’s method is the maximum hyperbox size parameter $\theta$ (The sensitivity parameter $\gamma$ is normally set to a constant so as to produce a moderately quick decrease from full membership to no membership).

In this section, we develop an enhanced scheme that operates with such overlapping and non-exclusive classes. In this process, we introduce another parameter $\delta$ to tune the system. We then study the effect of the parameters $\theta$ and $\delta$ on classification accuracy by applying the method to a real-world problem in scientific computation.

Consider the $k^{th}$ ordered pair $\{A_k, d_k\}$ from the training set. Let the desired output for the $k^{th}$ pattern be $[1, 1, 0, 0, ..., 0]$. Our algorithm considers this as two ordered pairs containing the same pattern $A_k$ but with two pattern classes as training outputs - $d_{k1} = [1, 0, 0, 0, ..., 0]$ and $d_{k2} = [0, 1, 0, 0, ..., 0]$ respectively. In other words, the pattern is associated with both class 1 and class 2. This will cause hyperboxes of both classes 1 and 2 to completely contain the pattern $A_k$. But according to Simpson’s original algorithm, one ordered pair can have complete membership in only hyperboxes of the same class. In other words, the algorithm will perceive this as undesirable overlap and contract the hyperboxes. It will be seen in the course of this section that the contraction step will cause the pattern to have complete membership in neither of the classes. Thus, the above procedure results in the
pattern having equal degrees of membership in both the hyperboxes but is not completely contained in either of them.

Assume that the network is first trained with the desired output as \(d_{k1} = [1, 0, 0, 0, \ldots, 0]\). This results in the \(k^{th}\) pattern \(A_k\) having complete containment in a hyperbox of class 1 (because the 1st bit is set to 1). Then when we train the same pattern with \([0, 1, 0, 0, \ldots, 0]\), a hyperbox of class 2 will be created/expanded to include the \(k^{th}\) pattern. This will result in hyperbox overlap. The hyperbox contraction step detailed below ensures that both the hyperboxes are adjusted so that each of them contain the \(k^{th}\) pattern to the same degree (which will be less than 1).

### Hyperbox Expansion:

Given labeled data of the form \(\{A_h, d_h\}\), find the hyperbox \(b_j\) that represents the class \(d_h\), provides the highest degree-of-membership and allows expansion (if needed). Since we bound the maximum hyperbox size by \(\theta\), the following condition is satisfied:

\[
\frac{1}{n} \sum_{i=1}^{n} (\max(w_{ji}, a_{hi}) - \min(v_{ji}, a_{hi})) \leq \theta
\]

### Overlap Testing:

A dimension-by-dimension comparison between hyperboxes is effected here. This test is conducted between the hyperbox expanded in the previous step and any other hyperbox that represents a different class. Let \(B_j\) be the one expanded in the previous step and \(B_k\) represent another hyperbox of a different class. If at least one of the following conditions is satisfied for a dimension, then we conclude that overlap exists between the hyperboxes.

### Hyperbox Contraction:

If overlap was detected in the \(i^{th}\) dimension, as detailed above, we minimally adjust the \(i^{th}\) dimensions of each of the overlapping hyperboxes. In other words, we try to adjust the hyperboxes so that only one of the min/max points is altered at a time.

Since each pattern can belong to more than one class, we need to define a new way to interpret the output of the fuzzy min-max neural network. In the original algorithm, we locate the node in the output layer with the highest value and set the corresponding bit to 1. All other bits are set to zero. In this way, a hard decision is obtained.

In the modified algorithm, however, we introduce a parameter \(\delta\) and we set to 1 not only the node with the highest output but also the nodes whose outputs fall within a band \(\pm \delta\) of the output value. This results in more than one output node getting included and consequently, aids in the determination of non-exclusive classes. It also allows us to include 'nearby classes' in our decision: Consider the scenario when a pattern gets associated with the wrong class, say Class 1, merely because of its proximity to members of Class 1 that were in the training samples rather than to members of its characteristic class (Class 2). Such a situation can be caused due to a larger incidence of the Class 1 patterns in the training set than the Class 2 patterns or due to a non uniform sampling, since we make no prior assumption on the sampling distribution. In such a case, the \(\delta\) parameter gives us the ability to make a soft decision by which we can associate a pattern with more than one class.

### 2.2 Clustering

Simpson has also presented a related technique for clustering that uses groups of fuzzy hyperboxes to represent pattern clusters. The details are almost analogous to his classification scheme and can be found in [56].

Hyperboxes, defined by pairs of min-max points, and their membership functions are used to define fuzzy subsets of the n-dimensional pattern space. The pattern clusters are represented by these hyperboxes. The bulk of the processing of this algorithm involves the finding and fine-tuning of the boundaries of the clusters. Simpson’s clustering algorithm, however, results in a large number of hyperboxes (clusters) to represent the given data adequately. Also, the clustering performance depends to a large extent on the maximum allowed size of a hyperbox. In other words, \(\theta\), the maximum hyperbox size influences the number of clusters formed, and in turn, the clustering accuracy. Simpson also desires the clusters to be “compact” and hence, performs a compaction procedure that eliminates overlap between hyperboxes in all dimensions. The disadvantage of this is that the algorithm requires more than one run through the data in order to achieve “cluster stability” and hence discourages single-pass clustering.

We propose a multi-resolution scheme, similar to computer vision [23], to partition the data into clusters. The basic idea is to look at the clustering process at differing levels of detail (resolution). For clustering at the base of
the multi-level pyramid, we use Simpson's algorithm. This is looking at the data at the highest resolution. Then, we operate at different "zoom"/resolution levels to obtain the final clusters. At each step up the pyramid, we operate on the clusters from the level below, rather than the original data points. Thus details are lost as we go up the hierarchy. This approach has led to encouraging results from clustering real world data sets.

The parameters of this algorithm are \( \theta \) — the maximum hyperbox size and \( Z \) — the zoom factor. The user specifies the zoom factor as the extent to which the algorithm should "focus" on the data in the pattern space. We also enhance the fuzzy hyperbox data structure as follows: \( M_j \) to contain the "center-of-mass" of the pattern samples represented by the hyperbox, and \( \delta \) the number of pattern samples represented by the hyperbox.

For example, when a hyperbox \( B_j \) is first created for pattern \( x_i \), \( V_j = W_j = x_i \) (i.e., the min and the max point both correspond to the pattern sample). Now, \( M_j \) is set to \( x_i \) as \( x_i \) is the only pattern "represented" by \( B_j \) and \( \delta \) is set to 1. When \( B_j \) is expanded to represent an additional pattern sample \( x_{i+1} \), in addition to \( V_j \) and \( W_j \) getting updated by Simpson's algorithm, we update \( M_j \) and \( \delta \) as follows:

\[
M_j = \frac{\delta M_j + x_{i+1}}{\delta + 1}
\]

\[
\delta = \delta + 1
\]

In other words, \( M_j \) is updated to reflect the new "center-of-mass" of the pattern samples represented by \( B_j \).

Our proposed algorithm operates as follows:

1. Initial clusters are formed from the pattern data by placing and adjusting the hyperboxes. At this stage, the number of clusters equals the number of hyperboxes. In our implementation, we have used Simpson's fuzzy min-max neural network, but any similar technique for such clustering can be used.

2. The bounding box formed by the patterns is calculated and we partition this region based on the zoom factor. In effect, this partitions the total pattern space into several levels of windows/regions. A zoom factor of \( Z \) implies that there exist \( Z \) levels above the bottom of the pyramid. The \( i \)th level above the base level partitions the total region into \( 4^Z \) sub-regions. For example, if we choose a zoom factor of 2, the first level above the base has 16 sub-regions and the next level has 4 sub-regions.

3. We then assume the highest zoom factor (i.e., which causes the window regions to assume the smallest size) and then examine the hyperboxes inside each window. If they are "sufficiently close by", we relabel them so that they indicate the same pattern cluster. The criterion for such combination is a function that depends on \( \theta \), \( Z \), the size of the bounding box and the actual distance between the hyperboxes. A good choice for such a heuristic (after empirical trial and error) was found to be \( d < D/2 + \theta \), where \( d \) is the actual distance between the hyperboxes and \( D \) is the diagonal of the current bounding box. Thus \( D \) represents the effect of the zoom factor \( Z \) on the pattern clustering. \( d \), the distance between hyperboxes is defined as the distance between the centers of masses of the two hyperboxes. In other words, if hyperboxes \( B_i \) and \( B_j \) are candidates for such "combination", then

\[
d = \| M_i - M_j \|
\]

The hyperboxes are combined if the distance condition is satisfied.

4. After we are done with all regions of a zoom factor, we zoom up and view those newly grouped hyperboxes at a higher level. The same procedure is repeated till no more hyperboxes can be relabeled.

5. In effect, we form the hyperbox clusters initially, then we zoom into the pattern space and progressively relabel them, zoom out accordingly and repeat the algorithm.

Another subtle point is deciding on the method to relabel clusters - Does hyperbox \( B_i \) take on the class of \( B_j \) or vice versa? The \( \delta \) parameter of the hyperboxes aid us in this decision. If the \( \delta \) of \( B_i \) is greater than that of \( B_j \), then \( B_j \) assumes the class of \( B_i \) and vice versa.

3 Description of classification techniques

Pattern Classification can be regarded as supervised learning based on inductive inference. The learning algorithm is presented with a sequence of input-output pairs of the form \((x_i, y_i)\), where \( x_i \) is a possible input vector of size \( n \) and \( y_i \) is the output associated with \( x_i \). The objective is to learn the function \( f \) that accounts for these examples. Then, given a 'new' \( x_i \), we can determine the \( y_i \) from \( f \) that most closely replicates the pattern exemplars. (Typically the \( y_i \)'s represent the pattern classes and hence assume values from 1 to \( c \), where \( c \) is the number of classes in the
domain.) We have used several different methods, statistical, neural and other techniques to perform classification. In this section, we describe the methods that were used, omitting details in the cause of brevity. The performance of these algorithms has been evaluated by applying them to several real world data sets. More information about these data sets is given in the next section.

3.1 Naive method

We started out with a naive heuristic, which represented a pattern class as the centroid of all the known exemplars of the class. The characteristic vector for a class is defined as the average, computed element-by-element, of the characteristic vectors of all the class members. That is, the $j^{th}$ element of the characteristic vector $\Psi_j(\cdot) \text{ of a class } C$ is computed as:

$$
\Psi_j(C) = \frac{1}{|C|} \sum_{p \in C} \Psi_j(p)
$$

where $|C|$ denotes the number of pattern examples in class $C$ and $\Psi_j(p)$ represents the characteristic vector of the class member $p$. The distance from a problem $p$ to a class $C$ is defined as the norm of the difference between the two characteristic vectors:

$$
d(p, C) = ||\Psi(p) - \Psi(C)||.
$$

The norm can be chosen as any reasonable distance measure. Then, we say that $p$ belongs to class $C$ if $d(p, C) < 6$ where $6$ is some threshold value that can be adjusted depending on the reliability of the characteristic vectors. This basic technique will serve as a baseline measure.

3.2 Statistical Techniques

The two basic statistical techniques commonly used for pattern classification are regression and discriminant analysis. We used the SAS/STAT routines [53] which implement these algorithms. Below, we describe briefly the basic ideas of these two techniques.

- Regression Models

Regression Analysis [12, 60] determines the relationship between one variable (also called the dependent or response variable) and another set of variables (called the independent variables). This relationship is often described in the form of several parameters. These parameters are adjusted until a reasonable measure of fit is attained. The SAS/STAT REG procedure serves as a general purpose tool for regression by least squares and supports a diverse range of models.

- Discriminant Analysis

Discriminant Analysis [16, 9, 57] uses a function called a discriminant function to determine the class to which a given observation belongs, based on knowledge of the quantitative variables. This is also known as “Classificatory Discriminant Analysis”. The SAS/STAT DISCRIM procedure computes discriminant functions to classify observations into two or more groups. It encompasses both parametric and non-parametric methods. When the distribution of pattern exemplars within each group can be assumed to be multivariate normal, a parametric method is used; if, on the other hand, no reasonable assumptions can be made about the distributions, non-parametric methods are used.

3.3 Classical Machine Learning Algorithms

Several algorithms that have been proposed by the AI community are described next. These include classical decision tree algorithms, native inducers and classical Bayesian classifiers. The implementations used are available in public domain in the MLC++ [28] (Machine Learning Library in C++).

In addition to directly using the techniques presented next, we also tested their performance by combining them with other inducers to improve their behavior etc., We found the most useful of such “wrappers” to be the Feature Subset Selection (FSS) inducer. The FSS inducer operates by selecting a “good” subset of features to present to the algorithm for improved accuracy and performance. The effectiveness of this wrapper inducer is dealt with in a future section.

- ID3

This is a classical iterative algorithm for constructing decision trees from examples [42]. The simplicity of the resulting decision trees is a characteristic of ID3’s attribute selection heuristic. Initially, a small “window" of
the training exemplars are used to form a decision tree and it is then determined if the decision tree so formed correctly classifies all the examples in the training set. If this condition is satisfied, then the process terminates; otherwise, a portion of the incorrectly classified examples is added to the window and then used to "grow" the decision tree. This algorithm is based on the idea that it is less profitable to consider the training set, in its entirety, than an appropriately chosen part of it.

- **HOODG**
  This is a greedy hill-climbing inducer for building decision graphs [27]. It does this in a bottom up manner. It was originally proposed to overcome the disadvantages of decision trees – duplication of subtrees in disjunctive concepts (replication) and partitioning of data into fragments, where a high-arity attribute is tested at each node (fragmentation). Thus, it is most useful in cases where the concepts are best represented as graphs and it is important to understand of the structure of the learned concept. It however, does not cater to unknown values. HOODG suffers from irrelevant or weakly relevant features and also requires discretised data. Thus, it must be used with another inducer and requires procedures like disc-filtering [11].

- **Const**
  This inducer just predicts a constant class for all the exemplars. The majority class present in the training set is chosen as this constant class. Though this approach is very naive, its accuracy is very useful as the baseline accuracy.

- **IB**
  Aha's Instance Based algorithms generate class predictions based only on specific instances [1, 61]. These methods, thus, do not maintain any set of abstractions for the classes. The disadvantage is that these methods have large storage requirements, but these can be significantly reduced with minor sacrifices in learning rate and classification accuracy. The performance also degrades rapidly with attribute noise in the exemplars and hence, it becomes necessary to distinguish noisy instances.

- **C4.5**
  C4.5 is a decision tree cum rule-based system [43]. C4.5 has several options which can be tuned to suit a particular learning environment. Some of these options include varying the amount of pruning of the decision tree, choosing among n "best" trees, windowing, using noisy data and several options for the rule induction program. The most used of these features are windowing and allowing C4.5 to build several trees and retaining the best.

- **Naive-Bayes**
  The Naive-Bayes Inducer [30] computes conditional probabilities of the classes given the instance and picks the class with the highest posterior. Features are assumed to be independent, an assumption that is most likely to be violated, but the algorithm is nevertheless robust in cases where this condition is not met. The probability that the algorithm will induce an arbitrary pair of concept descriptions is calculated and then this is used to compute the probability of correct classification over the instance space. This involves considering the number of training instances, the number of attributes, the distribution of these attributes, and the level of class noise.

- **oneR**
  Holte's one-R [21] is a simple classifier that makes a one-rule which is a rule based on the value of a single attribute. It is based on the idea that very simple classification rules perform well on most commonly used datasets. The minimum number of instances for a discretization interval is recommended to be 6. It is most commonly implemented as a base inducer. Using this algorithm, it is easy to get reasonable accuracy on many tasks by simply looking at one feature. However, it has been claimed to be significantly inferior to C4.5.

- **Aha-IB**
  This is an external system that interfaces with the IB basic inducer. It is basically used for tolerating noisy, irrelevant and novel attributes in conventional instance based learning. It is still a research system and is not very robust. More details about this algorithm can be obtained from [1].

- **Disc-Naive-Bayes**
  Better results to the Naive-Bayes inducer are provided by this algorithm. It achieves this by discretizing the continuous features. This preprocessing step is provided by chaining the disc-filter inducer to the naive-bayes inducer [11, 31].
• OCI-Inducer

This system is used for the induction of multivariate decision trees [39]. Such trees classify examples by testing linear combinations of the features at each non-leaf node in the decision tree. OCI uses a combination of deterministic and randomized algorithms to heuristically “search” for a good tree. It has been experimentally observed that OCI consistently finds much smaller trees than comparable methods using univariate tests.

3.4 Feed Forward Neural-Nets: Gradient Descent Algorithms

Let us suppose that in the classification problem, we represent the \( c \) classes by a vector of size \( c \). A 1 in the \( j^{th} \) position of the vector indicates membership in the \( j^{th} \) class. Our problem now becomes one of mapping the characteristic vector of size \( n \) into the classification vector of size \( c \). Feed forward neural networks have been shown to be effective in this task. Such a neural network is essentially a supervised learning system consisting of an input layer, an output layer and one or more hidden layers, each layer consisting of a number of neurons.

• BackProp

Using the backpropagation algorithm, the weights are then changed in a way so as to reduce the difference between the desired and actual outputs of the neural network. This is essentially using gradient descent on the error surface with respect to the weight values. For more details, see the classic text by Rumelhart & McClelland [49].

• BackProp with Momentum

The second algorithm we consider modifies backpropagation by adding a fraction (the momentum parameter, \( \alpha \)) of the previous weight change during the computation of the new weight change\[14\]. This simple artifice helps moderate changes in the search direction, reduce the notorious oscillation problems common with gradient descent. To take care of the “plateaus”, a “flat spot elimination constant” \( \lambda \) is added to the derivative of \( f \). Typical values of the momentum parameter are \((0 \ldots 1)\) and the flat spot elimination constant \( \lambda \) takes values from 0 to 0.25.

• QuickProp

Quickpropagation (QuickProp)[15], uses information about the curvature (and second derivative) of the error surface to compute the weight change. QuickProp approximates the error surface to be locally quadratic and attempts to jump in one step from the current position directly into the minimum of the quadratic.

• RProp

The final algorithm that we consider is called “Resilient backpropagation” (RProp)[6] because it uses the local topology of the error surface to make a more appropriate weight change. In other words, we introduce a ‘personal update value’ for each weight, which evolves during the learning process according to its local view of the error function. RProp is very powerful and efficient because the size of the weight step taken is no longer influenced by the size of the partial derivative. It is uniquely determined by the sequence of the signs of the derivatives, which provides a reliable hint about the topology of the local error function.

3.5 LVQ

LVQ (Learning Vector Quantization) borrows ideas from classical clustering and vector quantization techniques for signal processing, such as the k-nearest neighbor algorithm. Signal values are approximated by quantized references or ‘codebook’ vectors \( m_i \). Several ‘codebook’ vectors are assigned to each class in the domain, and a new pattern \( x \) is said to belong to the same class to which the nearest \( m_i \) belongs. LVQ determines effective values for the ‘codebook’ vectors so that they define the optimal decision boundaries between classes, in the sense of Bayesian decision theory. The accuracy and time needed for learning depend on appropriately chosen set of codebook vectors and the exact algorithm that modifies the codebook vectors. We have utilized four different implementations of the LVQ algorithm - LVQ1, OLVQ1, LVQ2 and LVQ3. LVQ.PAK [29], a LVQ program training package was used in the experiments.

4 Classification Results

We evaluated the performance of the various classification algorithms described above by applying them to real world data sets. In this section, the results on seven such data sets - IRIS, PYTHIA, Soybean, Glass, Ionosphere, Wine
and ECG - are described. Each of these data sets possess an unique characteristic. For example, the PYTHIA data set contains classes that are not mutually-exclusive, the Soybean data set contains data that have missing features etc., These data sets, with the exception of PYTHIA, were obtained from the Machine Learning Repository at the University of California, Irvine [38], which also contains details about the information contained in these datasets and their characteristics. In this section, we therefore, concentrate on the PYTHIA dataset which comes from our work in scientific computing - the efficient numerical solution of partial differential equations (PDEs) [59, 44, 25, 26]. PYTHIA is an intelligent computational assistant that prescribes an optimal strategy to solve a given PDE. This includes the method to use, the discretization to be employed and the hardware/software configuration of the computing environment. An important step in PYTHIA's reasoning is the categorization of a given PDE problem into one of several classes. The following non-exclusive classes are defined in PYTHIA (the number of exemplars in each class is given in parentheses):

(i) SINGULAR: PDE problems whose solutions have at least one singularity (6).
(ii) ANALYTIC: PDE problems whose solutions are analytic (35).
(iii) OSCILLATORY: PDE problems whose solutions oscillate (34).
(iv) BOUNDARY-LAYER: Problems that depict a boundary layer in their solutions (32).
(v) BOUNDARY-CONDITIONS-MIXED: Problems that have mixed boundary conditions (74).
(vi) SPECIAL: PDE problems whose solutions do not fall into any of the classes (i)-(v).

Each PDE problem is coded as a 32-component characteristic vector and there were a total of 167 problems in the PDE population that belong to at least one of the classes (i) to (vi).

4.1 Results from classification

In this section, we describe results from the classification experiments performed on the seven data sets described above. Each data set is split into two parts - the first part contains approximately 2/3 of the total exemplars. The second part represents the other one-third of the population. In performing these experiments, one part is used for "training" (i.e., in the modeling stage) and the other part is used to measure the "learning" and "generalization" provided by the paradigm (this is called the test data set). Each paradigm described in the previous section was trained using both (i) the first part and the (ii) the second part. For this reason, we refer to (i) as the larger training set and (ii) as the smaller training set. After training, the learning of the paradigm was tested by applying it to the entire data set. Each method previously discussed is operated with a wide range of the parameters that control its operation. We report the results from only the "best" set of parameters. Also, both parts of the data sets are chosen so that they represent the same relative proportion of the various classes as does the entire data set.

In each of these techniques, the number of patterns classified correctly was determined as follows: we first determine the error vector which is the component-by-component difference between the desired output and the actual output. Then, we fix a threshold for the $L_2$ error norm (c) and infer that patterns leading to error vectors with norms above the threshold have been incorrectly classified. We have carried out experiments using threshold values of 0.2, 0.1, 0.05 and 0.005 for each of the techniques.

- Traditional Method

It has been detailed above that the traditional method relies on the definition of an appropriate norm (distance measure) to quantify the distance of a problem $p$ from a class $C$. We have used three definitions of the norm $|| \cdot ||$, namely the norms $|| \cdot ||_1$, $|| \cdot ||_2$ and $|| \cdot ||_\infty$.

It was observed that the traditional method is very naive and averages around 50% accuracy for the datasets considered here. Varying the $L_2$ threshold (c), contrary to expectations, did not lead to a perceptible improvement/decline in the performance of the paradigm. Also norms $|| \cdot ||_1$ and $|| \cdot ||_2$ appear to perform better than $|| \cdot ||_\infty$ as they do a more reasonable task of 'encapsulating' the information in the characteristic vector by a scalar.

- Statistical Routines

The two statistical methods utilized were regression analysis and discriminant analysis. Proc REG performs linear regression and provides the user to chose from one of nine different models. We found the most useful of such models to be STEPWISE, MAXR and MINR. These methods basically differ in the ways in which they include or exclude variables from the model. The STEPWISE model starts with no variables in the model and slowly adds/deletes variables. The process of starting with no variables and slowly adding variables (without
deletion) is called forward selection. The MAXR and MINR provide more complicated versions of forward selection. In MAXR, forward selection is used to fit the best one-variable model, the best two-variable model and so on. Variables are switched so that a factor $R^2$ is maximized. $R^2$ is an indication of how much variation in the data is explained by the model. Model MINR is similar to MAXR, except that variables are switched so that the increase in $R^2$ from adding a variable to the model is minimized.

Then, REG uses the principle of least squares to produce estimates that are the best linear unbiased estimates under classical statistical assumptions. REG was tailored to perform pattern classification as follows: We again assume that the input pattern vector is of size $n$ and the number of classes are $c$. We append the 'class' vector at the end of the input vector to form an augmented vector of size $n + c$. These $n + c$ dimensional pattern samples are input as the regressor variables and the response variable is set to 1. This schema has the advantage that data sets that contain mutually non-exclusive classes do not require any different treatment from the other data sets.

For each regression experiment conducted, an analysis of variance was conducted afterwards. The two most useful results from this analysis are the 'F-statistic' for the overall model and the significance probabilities. The F-statistic is a metric for the overall model and indicates the percentage to which the model explains the variation in the data. The significance probabilities denote the significance of the parameter estimates in the regression equation. From these estimates, the accuracy of the regression was interpreted as follows: For a new pattern sample (size $n$), the 'appropriately' augmented vector is chosen that results in the closest fit i.e., the one which causes the least deviation from the output variable 1. Then the pattern is classified as belonging to the class represented by the augmented vector.

Proc DISCRIM, the other statistical routine discussed previously, performs discriminant analysis and computes various discriminant functions for classifying observations. As no specific assumptions are made about the distribution of pattern samples in each group, we adopt non-parametric methods to derive classification criteria. These methods include the kernel and the $k$-nearest-neighbor methods. The purpose of a kernel is to estimate the group-specific densities. Several different kernels can be used for density estimation - uniform, normal, biweight and triweight etc., - and two kinds of distance measures - Mahalanobis and Euclidean - can be used to determine proximity. We have experimented with each of the above methods and found the uniform kernel with an Euclidean distance measure to be most useful for the data sets described in this paper. This choice of the kernel was found to yield uniformly good results for all the data sets while other kernels led to suboptimal classifications.

Fig. 1 depicts the accuracy achieved by these methods for various data sets using the most accurate model. It can be seen that the DISCRIM procedure consistently outperforms the REG procedure. This can be explained
as follows [53]: DISCRIM obeys a canonical discriminant analysis methodology in which canonical variables are derived from the quantitative data, which are linear combinations of the given variables. These canonical variables summarize "between-class" variation in the same manner in which Principal Components Analysis (PCA) performs total variation. Thus a discriminant criterion is always derived in DISCRIM. In contrast, in the REG procedure, the accuracy obtained is limited by the coefficients of the variables in the regression equation. The measure of fit is thus limited by the efficiency of parameter estimation.

- Classical AI Algorithms

As described earlier, these algorithms are implemented in the Machine Learning library in C++ (MLC++) [28]. The table below depicts the performance of these methods on each of the seven data sets. The values of accuracy indicate the performance on the larger training set with a FSS wrapper inducer.

<table>
<thead>
<tr>
<th>Method</th>
<th>IRIS</th>
<th>PYTHIA</th>
<th>Soybean</th>
<th>Glass</th>
<th>Ion.</th>
<th>Wine</th>
<th>ECG</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID3</td>
<td>94</td>
<td>75.81</td>
<td>91.63</td>
<td>91.78</td>
<td>92.7</td>
<td>95.23</td>
<td>80.75</td>
</tr>
<tr>
<td>HOOKD</td>
<td>94</td>
<td>74.19</td>
<td>87.37</td>
<td>92.38</td>
<td>91.65</td>
<td>94.45</td>
<td>82.17</td>
</tr>
<tr>
<td>Const</td>
<td>98</td>
<td>82.26</td>
<td>96.3</td>
<td>95.78</td>
<td>96.7</td>
<td>95.57</td>
<td>86.00</td>
</tr>
<tr>
<td>IB</td>
<td>96</td>
<td>92</td>
<td>97.2</td>
<td>94.74</td>
<td>94</td>
<td>96</td>
<td>88.65</td>
</tr>
<tr>
<td>C4.5</td>
<td>94</td>
<td>66.13</td>
<td>95.7</td>
<td>93</td>
<td>93.68</td>
<td>90.23</td>
<td>72.73</td>
</tr>
<tr>
<td>Bayes</td>
<td>94</td>
<td>54.84</td>
<td>89.65</td>
<td>93</td>
<td>93.68</td>
<td>90.23</td>
<td>72.73</td>
</tr>
<tr>
<td>oneR</td>
<td>94</td>
<td>74.2</td>
<td>91.76</td>
<td>93</td>
<td>93.68</td>
<td>90.23</td>
<td>72.73</td>
</tr>
<tr>
<td>Aha-IB</td>
<td>94</td>
<td>69.59</td>
<td>96.1</td>
<td>93</td>
<td>93.68</td>
<td>90.23</td>
<td>72.73</td>
</tr>
<tr>
<td>DiscBayes</td>
<td>96</td>
<td>71.29</td>
<td>97.1</td>
<td>93.76</td>
<td>95.75</td>
<td>93.87</td>
<td>73.58</td>
</tr>
</tbody>
</table>

ID3 performs quite well except for the PYTHIA data set which has mutually non-exclusive features. However, its performance is slightly inferior to IB or C4.5. The HOODG base inducer's performance averages around that of the ID3 decision tree algorithm. Also, it does not perform very well on the Soybean and Echocardiogram databases because they contain missing features. It can be seen that the Const inducer achieves a maximum of only around 63% accuracy as it predicts the class which is represented in a majority in the training set. Incidentally, this high performance is achieved for the Ionosphere database which has 63.714% of its samples from the majority class. The IB inducer and C4.5 together account for a majority of the successful classifications. In each case, the highest accuracy achieved by any AI algorithm is realized by either IB or C4.5. However, in the case of the PYTHIA data set, IB falls very short of C4.5's performance which is still not as good as the other algorithms to be discussed in later sections. (The accuracy of C4.5 on PYTHIA is 92% while the best observed accuracy is 95.21%). It can also be observed from the above table that the Naive-Bayes inducer, Aha-IB, oneR classifier and the Disc-Naive-Bayes classifiers fall within a small band of each other. Further, in two out of the 7 data sets considered, the OCI inducer comes up with the second best overall performance.

Training with the smaller training set leads to, as expected, a degradation in the performance of the algorithms. Also, training with the FSS wrapper inducer results in better performance for the C4.5, Naive-Bayes, Disc-Naive-Bayes and the OCI inducers (For instance, the accuracy figures for the PYTHIA dataset with these algorithms are 91, 64, 07.74, 69.35 respectively without the FSS inducer and 92, 06.13, 69.59, 71.29 respectively with the FSS inducer). When the larger training set is used, the FSS inducer improves the performance of only one or two inducers while as much as 5 algorithms give better performance when it is used in conjunction with the smaller training set.

- Feed Forward Neural Networks

As described in the previous section, feed forward networks perform a mapping from the problem characteristic vector to an output vector describing class memberships. For each of the data sets, an appropriately sized network was constructed. The input layer contained as many neurons as the number of dimensions of the data set. The output layer contained as many neurons as the number of classes present in the data. Since the input and output of the network are fixed by the problem, the only layer whose size had to be determined is the hidden layer. Also, since we had no a priori information on how the various input characteristics affect the classification, we chose not to impose any structure on the connection patterns in the network. Our networks were thus fully connected, that is, each element in one layer is connected to each element in the next layer. The number of neurons in the hidden layer was chosen arbitrarily depending on the data set. Care was taken to ensure that the number is large enough to form an adequate "internal representation" of the domain. Also, it should be small enough to permit generalization from the training data. For example, the network that we
chose for the PYTHIA data set is of size 32x10x5. A good heuristic that we utilized was to set the number of hidden layer nodes to be a fraction of the number of features taking care that it does not exceed the number of classes in the domain. Each of the algorithms mentioned in the previous section was trained with five choices of the control parameters and the choice leading to the best performance was considered for performance evaluation. Each network was trained till the weights converged i.e., when subsequent iterations did not cause any significant changes to the weight vector. Again, as mentioned previously, training was done with both the larger training set and the smaller set was used to separately test the network. All simulations were performed using the Stuttgart Neural Network Simulator [14].

The only “free” parameter in the simple backpropagation paradigm was the learning rate \( \eta \) and it was varied in the range [0.1 ... 0.9]. It was observed that the best performance, in terms of classification accuracy, was achieved at \( \eta \) values of 0.8 - 0.95. Increasing \( \eta \) also led to a decrease in convergence time.

In the variant of backpropagation with momentum, the important parameters are the learning rate \( \eta \), the momentum coefficient \( \alpha \) and the flat spot elimination constant \( \lambda \). \( \eta \) was kept at a low value (0.2), because of the overpowering effect of the high momentum term which was found to be “optimal” at the values 0.7, 0.8, 0.9. The ideal value of the flat spot elimination constant was found to be around 0.05.

QuickProp also assumed a low value of the learning rate \( \eta \), approximately 0.2. Also, the parameters \( \mu \), the maximum growth parameter and \( \nu \), the weight decay term influence the performance of QuickProp very much. It was observed that the ideal value of \( \mu \) was in the range [1.75 ... 2] and that for \( \nu \) was either 0.0001 or 0.0002. QuickProp had a very fast convergence rate; even though it got into lots of local minima problems, it was always able to come out of them with very high momentum. Also, the maximum weight changes took place in the first 100 - 200 iterations and the subsequent iterations only served to “fine-tune” the error attained in these initial iterations.

Of all the supervised paradigms for feed forward neural networks studied in this article, RProp provided the best performance for the same number of training iterations. We chose a fixed value of \( \Delta \beta \) because the algorithm refines it iteratively and we set an upper bound 25 on the weight changes \( \Delta_{\text{max}} \). Even though some local minima problems were observed at high values of \( \Delta_{\text{max}} \), an extremely fast convergence rate served to make the network settle to a comfortable error level in about 100 iterations. The best performances were achieved at \( (\Delta_{\beta},\Delta_{\text{max}}) = (0.1,25) \).

Experiments with varying the \( L_2 \) error threshold gave further insights into the functioning of these four algorithms. As the threshold value was decreased, the performance of backpropagation, enhanced backpropagation, and QuickProp methods decline while that of RProp consistently maintains a high value. This can be immediately seen from the following \( L_2 \) error threshold values for the PYTHIA data set:

1. \( \epsilon = 0.005 \) : (47.3%, 72.45%, 74.25% and 95.83%)
2. \( \epsilon = 0.05 \) : (60.41%, 93.41%, 94.61% and 95.83%)
3. \( \epsilon = 0.1 \) : (92.81%, 94.01%, 94.61% and 95.83%)
4. \( \epsilon = 0.2 \) : (92.81%, 94.01%, 94.61% and 95.83%)

Another statistic that we found to be useful when comparing these methods was the mean and median values for the error norms of these algorithms with an appropriately chosen value for the \( L_2 \) error threshold. Again, it was seen that RProp provides the best performance of all the feed forward neural network paradigms. RProp’s median error is nearly negligible. While the mean value describes the average error, the very low median value of RProp shows us that while there are outliers, RProp classifies most of the problem patterns correctly.

Fig. 2 illustrate the performance of feed forward neural networks on the seven data sets. It can be seen that the statistics order these algorithms consistently in the following order of improving accuracy: BProp, BProp with momentum, QuickProp and RProp. The differences in the accuracies between “successive” algorithms (induced by the above ordering) was in the range 1 - 3% except for the PYTHIA data set which resulted in an extremely low performance for BProp, and, conversely, a very high performance for the RProp algorithm. Presumably, this data contained a lot of local minima hence the more sophisticated gradient descent algorithms performed better. Also, RProp was found to be a very good algorithm for most classification purposes. It should be noted that RProp achieved the best/second best performance for 5 out of 7 data sets. Training with the smaller training set instead of the larger leads to the expected degradation of performance.

- LVQ
Figure 2: Performance of feedforward neural networks on seven datasets

The LVQ algorithms mentioned in the previous section were trained as follows - a certain number of codebook vectors were chosen so that their numbers in the respective classes were proportional to their a priori probabilities. The total number of codebook vectors was set at approximately 1/3 of the total number of pattern samples in each data set. Then the algorithms were trained using both the larger and the smaller training sets. An adequate number of iterations was arrived at for each data set that resulted in convergence for both training sets.

The important free parameter in LVQ1 was the learning rate. This was varied from 0.1 to 1 in steps of 0.01. The highest accuracies were attained at a learning rate of 0.05 (this was for an $L_2$ threshold value of 0.005). LVQ1 is used to provide an "initial" solution and other LVQ algorithms can be used to improve the learning done by the LVQ1 algorithm. We adopt this strategy for our experiments.

OLVQ1 was subsequently trained and was found to improve the accuracy earlier obtained by LVQ1. The LVQ2 algorithm depends on the window width parameter $\theta$, the relative 'width' of the window into which the training data must fall. We varied the window width parameter from 0.1 to 0.5 and also the learning rate as mentioned in the LVQ1 experiment. It was observed that the optimal performance was achieved at a window width of around 0.3 and a learning rate of around 0.2. The LVQ3 algorithm can be used for an additional fine-tuning stage in learning. The relative learning rate parameter $\epsilon$ is used (multiplied by the parameter $\alpha$), when both the nearest codebook vectors belong to the same class. Again, as in the LVQ3 experiment, the relative window width parameter determines the "box" into which the training data must fall. Again, a window size of 0.3 was used and the relative learning rate parameter $\epsilon$ was set at 0.1.

The relative performance of the LVQ algorithms for the seven data sets is depicted in Fig. 3. It can be seen that OLVQ1 consistently outperforms all the other LVQ algorithms. Also, in 5 out of the 7 instances, LVQ2's performance was found to be as good as that of OLVQ1. It was observed that though LVQ3 improves the initial codebook, it does not give results better than the OLVQ1 algorithm.

- Neuro-Fuzzy Classifiers

For each of the data sets, the following experiments were conducted:

(i) Effect of $\theta$:

In this set of experiments, the max. hyperbox size was varied continuously and its effect on other variables was studied. In particular, it is observed that when $\theta$ was increased, a lesser number of hyperboxes needed to be formed, i.e., when $\theta$ tends to 1, the number of hyperboxes formed tends to the number of classes in the domain. Also performance on the training set and the test set steadily improved as $\theta$ was decreased. Performance on the training set was, expectedly, better than that on the test set. For instance, an 'optimal' error was found to be achieved at a $\theta$ value of around 0.006 for the IRIS data set and 0.00125 for the PYTHIA dataset.
Figure 3: Performance of LVQ algorithms on seven datasets

\( \theta \) was greater than the 'optimal' value so found, the error increased on both the sets and when \( \theta \) was less, the network overfit the training data so that its performance on the test set started to decline.

(ii) Effect of \( \delta \):
In this experiment, we set \( \theta \) to the optimal value and we vary \( \delta \) by assigning to it the values 0.01, 0.02, 0.05 and 0.09. It is observed that when \( \delta \) was increased, more output nodes tend to get included in the "reading-off" stage so that the overall error increased. For all the datasets, we found a value of 0.01 for \( \delta \) to be appropriate.

(iii) On-line adaptation:
The last series of experiments conducted were to test the fuzzy min-max neural network for its on-line adaptation, i.e., each pattern was incrementally presented to the network and the error on both sets was recorded at each stage. It was observed that the number of hyperboxes formed slowly increases from 1 to the optimal number obtained in Expt.(i). Also, performance on both sets steadily improved to the values obtained in Expt.(i).

Varying the \( L_2 \) error threshold value \( \epsilon \) was found to not alter the accuracy of the fuzzy min-max network. Fig. 4 depicts the performance of Simpson's fuzzy min-max algorithm and the modified algorithm for each of the seven data sets. It can be seen that these algorithms exhibit a difference in performance only in the presence of mutually non-exclusive classes, in this case, the PYTHIA data set. Also, these algorithms appear to achieve high accuracies consistently for all the data sets, much like the RProp algorithm discussed previously.

• Overall Comparison
The first column besides the algorithms describe the number of instances in which it produced the optimal classification. The next column indicates the number of times it was ranked second. The final column indicates the % error range within which it produced the classifications.

It can be seen that the traditional method is very naive as it represents the pattern classes by the centroid of the known samples. It performed very poorly and the highest accuracy achieved by it on a data set is 72%. The statistical routines performed better, with discriminant analysis faring better than regression analysis. It should be noted that more complicated forms of regression, possibly leading to better accuracy, can be applied if more information is known about the data sets. Discriminant analysis is a more natural statistical way to perform pattern classification and its range of error is considerably less than that of regression analysis. Its accuracy was in the range 87 - 95% except for the echocardiogram database which was a particularly difficult...
data set among those considered here. Among the AI algorithms, the best ones discussed here are IB and C4.5. Together they accounted for 4 of the 7 best classifications. Their performance was further enhanced by a Feature Subset Selection Inducer. However, these algorithms did not fare well with the PYTHIA data set which contained mutually non-exclusive classes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Best</th>
<th>Second Best</th>
<th>Range of error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>—</td>
<td>—</td>
<td>24.70-55.20</td>
</tr>
<tr>
<td>Proc REG</td>
<td>—</td>
<td>—</td>
<td>5.62-16.18</td>
</tr>
<tr>
<td>Proc DISCRIM</td>
<td>—</td>
<td>—</td>
<td>1.99-8.27</td>
</tr>
<tr>
<td>ID3</td>
<td>—</td>
<td>—</td>
<td>4.00-20.02</td>
</tr>
<tr>
<td>HOODG</td>
<td>—</td>
<td>—</td>
<td>2.83-21.64</td>
</tr>
<tr>
<td>Const</td>
<td>—</td>
<td>—</td>
<td>32.97-84.16</td>
</tr>
<tr>
<td>IB</td>
<td>3</td>
<td>—</td>
<td>0.00-13.57</td>
</tr>
<tr>
<td>C4.5</td>
<td>1</td>
<td>1</td>
<td>0.00-3.83</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>—</td>
<td>—</td>
<td>1.50-29.70</td>
</tr>
<tr>
<td>oneR</td>
<td>—</td>
<td>—</td>
<td>2.78-41.00</td>
</tr>
<tr>
<td>Aha-IB</td>
<td>—</td>
<td>—</td>
<td>2.78-21.03</td>
</tr>
<tr>
<td>Disc-Naive-Bayes</td>
<td>—</td>
<td>—</td>
<td>1.10-26.24</td>
</tr>
<tr>
<td>OC1</td>
<td>—</td>
<td>2</td>
<td>0.10-24.54</td>
</tr>
<tr>
<td>BProp</td>
<td>—</td>
<td>—</td>
<td>2.35-48.53</td>
</tr>
<tr>
<td>BProp with momentum</td>
<td>—</td>
<td>—</td>
<td>2.35-48.53</td>
</tr>
<tr>
<td>QProp</td>
<td>—</td>
<td>1</td>
<td>0.61-21.58</td>
</tr>
<tr>
<td>RProp</td>
<td>3</td>
<td>2</td>
<td>0.00-2.80</td>
</tr>
<tr>
<td>LVQ1</td>
<td>—</td>
<td>—</td>
<td>5.54-23.52</td>
</tr>
<tr>
<td>OLVQ1</td>
<td>—</td>
<td>—</td>
<td>1.13-15.83</td>
</tr>
<tr>
<td>LVQ2</td>
<td>—</td>
<td>—</td>
<td>1.13-18.58</td>
</tr>
<tr>
<td>LVQ3</td>
<td>—</td>
<td>—</td>
<td>2.34-17.86</td>
</tr>
<tr>
<td>FMMN</td>
<td>1</td>
<td>1</td>
<td>0.55-6.01</td>
</tr>
<tr>
<td>Modified FMMN</td>
<td>1</td>
<td>2</td>
<td>0.55-2.37</td>
</tr>
</tbody>
</table>

Feed forward neural networks, in general, performed quite well, with more complicated training schemes like enhanced backpropagation, Quick Propagation and Resilient Propagation clearly winning over plain error
backpropagation. For higher $L_2$ error threshold values (say 0.2), all these learning techniques gave values close to each other. However, when the $L_2$ error threshold levels were lowered (to, say, 0.005), RProp clearly won out on all the other methods. The same observations can be made by looking at the mean and median of the error values. While the mean for RProp is slightly lower than that of others, the median is significantly lower. This indicates that RProp classifies most patterns correctly with almost zero error, but has few outliers. The other methods have the errors spread more 'evenly', which leads to a degradation in their performance as compared to RProp. RProp also bagged 3 out of the 7 optimal classifications. The variants of the LVQ method (LVQ1, OLVQ1, LVQ2 and LVQ3) that we tried performed about average. While they were better than the naive classifier, their performance was in the 80 - 95% range (for an $L_2$ error threshold value of 0.005). Increasing the $L_2$ error threshold value did not serve to improve the accuracy. Finally, the neuro-fuzzy techniques that we tried out performed quite well. In fact, they performed almost as well as RProp, in terms of % accuracy, mean error and median error. Like RProp, and unlike the other feed forward neural networks, increasing the $L_2$ error threshold did not significantly alter the performance. Considering that unlike RProp, these techniques allow on-line adaptation (i.e., new data do not require retraining on the old data), they are advantageous in this context.

5 Description of clustering techniques

Clustering is another fundamental procedure in pattern recognition. It can be regarded as a form of unsupervised inductive learning that looks for regularities in training exemplars. The clustering problem [4, 16] can be formally described as follows:

**Input** A set of patterns $X = \{x_1, x_2, x_3, \ldots, x_n\}$

**Output** A $c$-partition of $X$ that exhibits categorically homogeneous subsets, where $2 \leq c \leq n$

Different clustering methods have been proposed that represent clusters in different ways - for example, using a representative exemplar of a cluster, a probability distribution over a space of attribute values, necessary and sufficient conditions for cluster membership etc [4]. To represent a cluster by a collection of training exemplars and to "assign" new samples to existing clusters, we use some form of a utility measure. This is normally based on some mathematical property such as distance, angle, curvature, symmetry, intensity that are exhibited by the members of $X$. It has been recognized [51] that no universal clustering criterion can exist and that selection of any such criterion is subjective and depends on the domain of application under question.

5.1 SAS Routines

The SAS/STAT package provides a lot of interesting routines for pattern clustering. It offers both hierarchical clustering and determination of disjoint clusters. There are three basic clustering algorithms provided in SAS/STAT:

- **CLUSTER**
  
  Procedure CLUSTER performs hierarchical clustering of observations using eleven agglomerative methods applied to coordinate data. All of these are based on the usual agglomerative clustering procedure. Initially, each observation starts as an independent cluster. Then, the two closest clusters are merged to form a new cluster that replaces the two old clusters. Merging is discontinued when there are no clusters ‘close enough’ to be combined.

- **FASTCLUS**
  
  The CLUSTER procedure is not appropriate for handling large data sets because the time taken for clustering varies as the cube of the number of observations in practical data sets. The FASTCLUS procedure [19, 33] finds disjoint clusters of observations using a k-means method applied to coordinate data. This efficient algorithm for disjoint clustering is composed of an effective algorithm for finding initial clusters and a standard iterative method for minimizing the sum of squared distances from the cluster means.

- **VARCLUS**
  
  Procedure VARCLUS performs both hierarchical and disjoint clustering by multiple-group component analysis [18]. The set of numeric variables is split into either disjoint/hierarchical clusters. A linear combination of the variables in it is then associated with each cluster. The choice of this linear combination is usually either
the first principal component or the centroid component. Then, VARCLUS tries to minimize the sum across clusters of the variance of the original variables that is explained by the cluster components.

5.2 AutoClass C++

AutoClass C++ [7] is an unsupervised Bayesian system that seeks a maximum posterior probability clustering of the pattern exemplars. It is based on the classical mixture model, supplemented by a Bayesian method to determine the optimal clusters. While the authors of AutoClass C++ emphasize that the discovery of clusters in data is rarely "one-shot", we were interested in determining the accuracy of the so-called "initial approximations" provided by AutoClass C++. The various models provided with this package are the Single Multinomial Model, Single Normal CN Model, Single Normal CM Model and the Multi Normal CN Model. We refer the interested reader to [7] for details.

6 Experimental results from clustering

In this section, we detail the results obtained by applying the above clustering algorithms to the seven real-world data sets discussed previously. The clustering experiments were carried out in the following manner: No constraint is initially set on the number of clusters detected by a particular algorithm. After these clusters are formed, they are "mapped" to the physical clusters known to be present in the data. In other words, each cluster detected is analyzed as to which physical cluster is maximally represented by it. (This means that two or more clusters detected may map to the same physical cluster.) Confusion Matrices are then generated from this mapping data. The rows of the confusion matrix represent the clusters detected by the algorithm. The columns denote the actual clusters known to exist in the data. An entry in the \((i,j)\) position of the table represents the degree to which cluster \(i\) faithfully represents the actual data in cluster \(j\). These matrices determine the number of pattern samples associated with a "wrong" cluster. Thus, the performances of the clustering algorithms are determined by the number of mis-clustered pattern samples.

- **SAS/STAT routines**
  The procedure CLUSTER encompasses a number of models and we found the most appropriate one to be Ward's minimum-variance method (error sum of squares) [58].

  Procedure FASTCLUS is meant for clustering of very large data sets and we noted that it finds reasonable clusters in two or three passes over the data. The parameters for this procedure are the maximum number of clusters and, optionally, the minimum radius of the clusters. FASTCLUS uses a nearest centroid sorting technique in which a set of points called cluster seeds is selected as a first guess of the means of the clusters. Each observation is assigned to the nearest seed to form temporary clusters. The seeds are then replaced by the means of the temporary clusters, and the process is repeated until no further changes occur in the clusters. The above initialization scheme sometimes makes FASTCLUS very sensitive to outliers. VARCLUS, on the other hand, attempts to divide a set of variables into non-overlapping clusters in such a way that each cluster can be interpreted as essentially unidimensional. For each cluster, VARCLUS computes a component that can be either the first principal component or the centroid component and tries to maximize the sum across clusters of the variation accounted for by the cluster components. The one important parameter for VARCLUS is the stopping criterion. We chose the default criterion that stops when each cluster has only a single eigenvalue greater than one. This is most appropriate because it determines the sufficiency of a single underlying factor dimension.

  Fig. 5 presents the results of applying these routines to the seven data sets. It can be seen that VARCLUS falls consistently into the last place and that CLUSTER and FASTCLUS together account for the best clustering results.

- **AutoClass C++ routines**
  The two most useful models in AutoClass C++ were found to be the Single Normal CM Model for data sets that had missing values and the MultiNormal CN Model for other data sets. Fig. 6 depicts the results for the seven data sets. AutoClass utilizes several different search strategies - converge_search_3, converge_search_4 and converge. We found converge_search_3 to be the most useful because the other two methods did substantially worse on the data sets.

- **Neuro-Fuzzy Systems**
Figure 5: Performance of STAT clustering routines on seven datasets

Figure 6: Performance of AutoClass C++ clustering routines on seven datasets
The two hybrid neuro-fuzzy algorithms discussed were Simpson’s fuzzy min-max algorithm and our multiresolution fuzzy clustering algorithm. Fig. 7 depicts the results for the seven data sets. The original fuzzy min-max clustering algorithm performed reasonably well. The clustering accuracy varied very much with the hyperbox size $\theta$. This is because each hyperbox was labeled as a separate cluster and hence, a lower $\theta$ resulted in a better clustering.

Our multiresolution clustering algorithm consistently performed better than Simpson’s. We obtained encouraging results for all the data sets except the PYTHIA data set which contained mutually non-exclusive classes. The neuro-fuzzy scheme does not allow hyperbox clusters to overlap and hence, each pattern sample gets associated with only one cluster. This causes the accuracy to drop down. It was observed that while most data sets require only two levels on the multi-resolution pyramid, the echocardiogram data set needed a 3-level pyramid to obtain the reported accuracy. The clustering accuracy did not vary with the hyperbox size $\theta$ as much as in the case of Simpson’s original fuzzy min-max clustering algorithm. However, a greater accuracy was observed at small values of $\theta$.

### Overall Comparison

The following table summarizes the comparative performance of the various clustering algorithms. It can be readily seen that the fuzzy clustering algorithms and SAS/STAT routines account for a majority of the optimal clusterings. The AutoClass routines also perform well, though they account only for 3 of the best clusterings. Simpson’s fuzzy min-max network, though performing very good clustering, does not obtain the optimal clustering in any of the data sets considered in this paper. It manages to obtain second place for only one of the seven data sets. Our multiresolution algorithm performs very well and accounts for 3 of the best classifications, more than any other algorithm. Also, it provides an error range almost identical to that provided by the SAS/STAT routines. The error ranges of the SAS/STAT and the multiresolution clustering algorithm indicate that these routines perform well on the datasets considered in this paper. This is because our algorithm is similar to the CLUSTER procedure in SAS (using centroid based merging). Unlike CLUSTER, however, our technique has inherent parallelism which can be exploited to reduce the time complexity of the process.

<table>
<thead>
<tr>
<th>Method</th>
<th>Best</th>
<th>Second Best</th>
<th>Range of Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUSTER</td>
<td>1</td>
<td>5</td>
<td>0-2</td>
</tr>
<tr>
<td>FASTCLUS</td>
<td>2</td>
<td>3</td>
<td>0-2</td>
</tr>
<tr>
<td>VARCLUS</td>
<td>1</td>
<td>3</td>
<td>0-2</td>
</tr>
<tr>
<td>AutoClass</td>
<td>2</td>
<td>1</td>
<td>0-3.33</td>
</tr>
<tr>
<td>Simpson’s</td>
<td>-</td>
<td>1</td>
<td>1.12-3</td>
</tr>
<tr>
<td>Agglomerative</td>
<td>3</td>
<td>1</td>
<td>0-1.95</td>
</tr>
</tbody>
</table>
7 Conclusions

In this paper, we have described two hybrid neuro-fuzzy schemes - one for pattern classification and the other for clustering. Both these schemes utilize fuzzy hyperboxes to represent pattern classes. The clustering scheme is motivated by the human visual system. These schemes were extensively compared with traditional, statistical, neural and machine learning algorithms by experimenting with real world data sets. The classification algorithm performs as well as some of the better algorithms discussed here like - C4.5, IB, OC1 and RProp. Besides, this algorithm has the ability to provide on-line adaptation. The clustering algorithm borrows ideas from computer vision to partition the pattern space in a hierarchical manner. It has been found that this simple technique yields very good results. It was seen that the performance of this algorithm is very good on clustering real world data sets. We feel that our clustering scheme provides good support for pattern recognition applications in real-world domains.

References


