On Neurobiological, Neuro-Fuzzy, Machine Learning, and Statistical Pattern Recognition Techniques

Anupam Joshi, *Member, IEEE,* Narendran Ramakrishman, *Member, IEEE,* Elias N. Houstis, and John R. Rice

***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 nonexclusive 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 the machine learning repository at the University of California, Irvine. We find that our proposed schemes compare quite well with the existing techniques, and in addition offer the advantages of one-pass learning and on-line adaptation.**

***Index Terms*— Pattern recognition, classification, clustering, neuro-fuzzy systems, multiresolution, vision systems, overlapping classes, comparative experiments.**

I. Introduction, Background, and Related Work

We begin this paper, to paraphrase the popular song, *at the very beginning* in consideration of the interdisciplinary audience that is the target of this issue. Neural networks (NN's) represent a *computational* [36] approach to intelligence as contrasted with the traditional, more symbolic approaches. The idea of such systems is due to the work of the psychologist D. Hebb [20] (and after whom a class of learning techniques is referred to as Hebbian). Despite the pioneering early work of McCullouch and Pitts [39] and Rosenblatt [51], the field was largely ignored through most of 1960's and 1970's, with researchers in artificial intelligence (AI) mostly concentrating on symbolic techniques. Reasons for this could be the lack of appropriate computational hardware or the work of Minsky and Papert which showed limitations of a class of NN's (single layer perceptrons) popular then. The failure of good old-fashioned AI (GOFAI) [5], the development of very large-scale integration (VLSI) and parallel computing revived interest in NN's in the mid 1980's as an alternate mechanism to investigate, understand, and duplicate intelligence. 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 [36].

Some researchers view NN's as mechanisms to study intelligence (e.g., the famous text by McClelland and Rumelhart [53]), but most literature in the area sees NN's as a tool to solve problems in science and engineering. Most of these problems involve pattern recognition (PR) in one form or another—everything from speech recognition to image recognition to SAR/Sonar data classification to stock market tracking, and so on. The paper by Jain *et al.* [24] elaborates upon this viewpoint. These problems involve both classification (*supervised learning*) and clustering (*unsupervised learning*). Recently, many researchers have investigated the links between NN-based techniques and traditional statistical pattern recognition techniques. One of the first efforts in this direction was the seminal text by Jain and Sethi [57]. Since then, this topic has aroused considerable interest and has seen many discussions—some acrimonious, between those who feel that NN's are old wine in new bottles, and those who feel that they represent a new paradigm. As any follower of the news group comp.ai.neural-nets knows, this debate occurs there almost every six months, often triggered by an innocent question from a "newbie."

In addition, several works have given scholarly discussions of these links—see the excellent overview of Cheng and Titterington [8]. Responses to their article by, among others, Amari [2], McClelland [38], and Ripley [49], also commented on these relationships and suggested avenues for potential cross disciplinary work. Sarle [55] has described how some of the simpler NN models can be described in terms of, and implemented by, standard statistical techniques. Ripley's work [48], [50] along the same lines presents some empirical results comparing 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. Duin [13] makes interesting observations on techniques used to compare classifiers.

An area that has remained relatively unexplored in this interdisciplinary context is the use of NN techniques that are closely related to biological neural systems. The human visual system can out perform most computer systems on pattern recognition and identification tasks and part of this capability comes from the human ability to classify and

Manuscript received January 11, 1996; revised June 29, 1996. This work was supported in part by NSF Grants ASC 9404859 and CCR 9202536, AFOSR Grant F49620-92-J-0069, and ARPA ARO Grant DAAH04-94-G-0010.

A. Joshi is with the Department of Computer Engineering and Computer Science at the University of Missouri, Colombia, MO 65211 USA.

N. Ramakrishnan, E. N. Houstis, and J. R. Rice are with the Department of Computer Sciences, Purdue University, West Lafayette, IN 47907 USA.

Publisher Item Identifier S 1045-9227(97)00241-5.

categorize. Extensive studies by psychologists have suggested a threefold process to model human abilities. 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 [43] for a detailed exposition. Such work is of increasing importance 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 *wetware* configurations. Specifically, the use of multiresolution processing has attracted much interest from the vision community [25]. This technique uses multiple representations of the same input at different resolutions which are obtained by blurring the image with Gaussian kernels of differing widths. The notion of hierarchical representations also gets support from neuro physiological data. Enroth–Cugell [14] showed as far back as the 1960's that the retinal processing being done by a Cat's ganglion cells can be likened to a difference of Gaussians. Marr and Hildereth [37] showed that even for human retinal processing, a similar Laplacian of Gaussian (LOG) operator could be defined. Joshi and Lee [26] showed that an NN 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 is similar to the LOG operator. Daugman [10] suggested the use of Gabor 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 infero temporal (IT) regions [22], [34]. The receptive field (the region in the photoreceptor layer whose activity influences it) of a cell in the lateral geniculate nucleus, for instance, will be larger than that of a retinal ganglion cell.

This kind of view has given rise to multiresolution-based algorithms, implemented in a special pyramid like parallel architecture. 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 pyramid is a nonoverlapped quad pyramid, where each processor receives input from four processors in the layer below it [25]. Several recent works, including [44], 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 [58]. 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 of 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], [28].

II. Neuro-Fuzzy Schemes

A. Classification

We have developed a new algorithm for classification [47], which is a modification of a technique proposed by Simpson [58]. 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 contain 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 an NN assembly. Learning in the fuzzy min–max network proceeds by placing and 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. This system can be represented as a three-layer feedforward NN with a single pass fuzzy algorithm for determining weights.

Initially, the system starts with an empty set (of hyperboxes). As each pattern sample is "taught" to the fuzzy min–max network, either an existing hyperbox (of the same class) is expanded to include the new pattern or a new hyperbox is created to represent the new pattern. The latter case arises when we do not have an already existing hyperbox of the same class or when we have such a hyperbox but which cannot expand any further beyond a limit θ set on such expansions.

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 figures into classes such as polygon, square, rectangle etc., Note that these classes are not mutually exclusive (i.e., a square is 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 Simpson's method is the maximum hyperbox size parameter θ—this denotes the limit beyond which a hyperbox cannot expand to "enclose" a new pattern.

(There is a sensitivity parameter γ which is normally set to a constant so as to produce a moderately quick gradation from full membership to no membership). In this section, we develop an enhanced scheme that operates with such overlapping *and* nonexclusive classes. In the process, we introduce another parameter δ to tune the system.

Consider the kth ordered pair $\{A_k, d_k\}$ from the training set, where A_k is the kth pattern sample and d_k is the class vector denoting membership of A_k in the various classes (a "1" denotes membership and a "0" represents an absence of membership). Assume, for example, that the desired output for the kth pattern (A_k) be $[1, 1, 0, 0, \cdots, 0]$. Our algorithm considers this as two ordered pairs containing the same pattern A_k but with two pattern classes as training outputs—$d_{k1} = [1, 0, 0, 0, \cdots, 0]$ and $d_{k2} = [0, 1, 0, 0, \cdots, 0]$, 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, unlike Simpson's algorithm. Thus, we *allow* hyperboxes to overlap if the problem domain so demands.

Since each pattern can belong to more than one class, a new way to interpret the output of the fuzzy min–max NN needs to be defined. In the original algorithm, one locates the node in the output layer with the highest value and sets the corresponding bit to one. All other bits are set to zero, obtaining a hard decision.

In the modified algorithm, however, we introduce a parameter δ and we set to one *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 nonexclusive classes. It also allows our algorithm to handle "nearby classes." 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 nonuniform sampling, since we make no prior assumption on the sampling distribution. In such a case, the δ parameter gives us the ability to make a soft decision by which we can associate a pattern with more than one class.

B. 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 [59].

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, θ, 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 multiresolution scheme, similar to computer vision [25], 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 multilevel 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 treat the clusters from the level below as points at this level. As we go up the hierarchy, therefore, we view the original data with decreasing resolution. This approach has led to encouraging results from clustering real world data sets.

The parameters of this algorithm are θ—the maximum hyperbox size and Z—the zoom factor which represents the number of resolution levels. 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 η 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 η is set to one. 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 η as follows:

$$\begin{aligned} M_j &= \frac{\eta M_j + x_{i+1}}{\eta + 1} \\ \eta &= \eta + 1. \end{aligned}$$

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 NN, 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 ith level above the base level partitions the total region into $4^{(Z-i+1)}$ subregions. For example, if we choose a zoom factor of two, the first level above

the base has 16 subregions and the next level has four subregions.

3) We assume the highest zoom factor (i.e., which causes the window regions to assume the smallest size) and examine the centers of masses of 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 θ, 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. The distance between two hyperboxes is defined as the distance between their centers of mass. In other words, if hyperboxes B_i and B_j are candidates for such "combination," then

$$d = ||M_i - M_j||_2.$$

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 these newly grouped hyperboxes at a higher level. The same procedure is recursed through till no more hyperboxes can be relabeled.

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 η parameter of the hyperboxes aid us in this decision. If the η of B_i is greater than that of B_j, then B_j assumes the class of B_i and vice versa.

III. Description of Some Other 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 an 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 one to c, where c is the number of classes in the domain.) We have used several different methods, statistical, neural, and others, to perform classification and compare results. In this section, we describe the methods that were used, omitting details for the sake 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. Some interesting observations on techniques used to compare classifiers can be found in [13].

A. Traditional Method

We started out with a traditional 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 jth element of the characteristic vector $\Psi(\cdot)$ 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 $d(p, C) < \delta$ where δ 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 of classification accuracy.

B. 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++ [30] (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 tress from examples [45]. 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 [29]. 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 discretized 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], [64]. 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 [46]. 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.

Bayes: The Bayes inducer [32] computes conditional probabilities of the classes given the instance and picks the class with the highest posterior. Features are assumed to be independent 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. 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-Bayes: Better results to the 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], [33].

OC1-Inducer: This system is used for the induction of multivariate decision trees [42]. Such trees classify examples by testing linear combinations of the features at each nonleaf node in the decision tree. OC1 uses a combination of deterministic and randomized algorithms to heuristically "search" for a good tree. It has been experimentally observed that OC1 consistently finds much smaller trees than comparable methods using univariate tests.

C. Statistical Techniques

The two basic statistical techniques commonly used for pattern classification are regression and discriminant analysis. We used the SAS/STAT routines [56] which implement these algorithms. Below, we describe briefly the basic ideas of these two techniques.

Regression Models: Regression analysis [12], [63] 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. For methods of regression using logistic models, we used the SAS/STAT LOGISTIC procedure.

Discriminant Analysis: Discriminant analysis [9], [16], [60] 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 nonparametric 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, nonparametric methods are used.

D. Feedforward 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 one in the jth position of the vector indicates membership in the jth class. Our problem now becomes one of mapping the characteristic vector of size n into the classification vector of size c. Feedforward NN's have been shown to be effective in this task. Such a NN 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.

Backpropagation: Using the backpropagation (BP) algorithm, the weights are then changed in a way so as to reduce the difference between the desired and actual outputs of the NN. 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 and McClelland [52].

BP with Momentum: The second algorithm we consider modifies BP by adding a fraction (the momentum parameter, α) of the previous weight change during the computation of the new weight change [65]. 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" λ is added to the derivative of f. Typical values of the momentum parameter are

$(0 \cdots 1)$ and the flat spot elimination constant λ takes values from 0–0.25.

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.

E. LVQ Algorithms

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 an 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, [31] a LVQ program training package was used in the experiments.

IV. 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, ECG and wine—are described. Each of these data sets possess an unique characteristic. The IRIS data set, for instance, contains three classes—one is linearly separable from the others while the other two are not linearly separable from each other. 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 of the University of California at Irvine [41], 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 (PDE's) [27], [28], [47], [62]. 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 nonexclusive classes are defined in PYTHIA (the number of exemplars in each class is given in parentheses).

1) *Singular:* PDE problems whose solutions have at least one singularity (6).
2) *Analytic:* PDE problems whose solutions are analytic (35).
3) *Oscillatory:* PDE problems whose solutions oscillate (34).
4) *Boundary-layer:* Problems that depict a boundary layer in their solutions (32).
5) *Boundary-conditions-mixed:* Problems that have mixed boundary conditions (74).
6) *Special:* PDE problems whose solutions do not fall into any of the classes 1) through 5).

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 1) through 6).

A. 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 two-thirds 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 1) the first part and the 2) the second part. For this reason, we refer to 1) as the larger training set and 2) as the smaller training set. After training, the learning of the paradigm was tested by applying it to the portion of the data set that it has not encountered before. This is the "generalization" accuracy. (The recall accuracy is computed by considering only the portion of the data set used for "training"). Each method previously discussed was operated with a wide range of the parameters that control its behavior. We report the results from only the "best" set of parameters and due to space considerations, we provide only the generalization accuracy. 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 (ϵ) and infer that patterns leading to error vectors with norms above the threshold have been incorrectly classified. We have carried out experiments

TABLE I
THE PERFORMANCE (% ACCURACY IN CLASSIFICATION) OF THE TEN CLASSICAL AI ALGORITHMS

Algorithm	IRIS	PYTHIA	Soybean	Glass	Ionosphere	ECG	Wine
ID3	93.0	74.8	90.5	91.8	92.7	80.7	95.2
HOODG	93.0	73.2	85.8	92.4	91.6	82.2	94.4
Const	30.0	38.7	13.0	16.8	63.7	33.3	39.9
IB	97.0	80.0	96.3	95.8	96.7	86.0	96.6
C4.5	95.0	91.0	97.2	94.7	94.0	88.6	96.0
Bayes	93.0	66.0	95.7	93.0	93.7	72.7	90.2
oneR	93.0	54.0	89.6	93.0	93.7	72.7	90.2
Aha-IB	93.0	72.2	91.8	93.0	93.7	72.7	90.2
Disc-Bayes	93.0	60.5	96.1	93.0	93.7	72.7	90.2
OC1	95.0	70.4	97.1	93.8	95.7	73.6	93.9

TABLE II
THE PERFORMANCE (% ACCURACY IN CLASSIFICATION) OF 13 ALGORITHMS

Algorithm	IRIS	PYTHIA	Soybean	Glass	Ionosphere	ECG	Wine
Proc REG	83	77	85	90	89.57	70.1	90.2
Proc LOGISTIC	92	87	87.29	91.19	93.46	75.29	95.88
Proc DISCRIM	90	86.27	91.37	92.78	92.22	72.98	94.79
Bprop	78.535	47.3	87.795	86.57	94.35	83.39	95.18
Bprop with momentum	80.65	72.45	92	93.27	94.35	84.44	97.632
QuickProp	83.79	74.25	93.88	95.17	95.51	87.65	98
RProp	95.2	95.83	94.7	95.23	96.23	89.57	100
LVQ1	81.63	77.06	73.68	83.7	91.16	76.52	96.2
OLVQ1	91.7	80	83.76	90.63	95.57	80.72	96.2
LVQ2	91.75	79.79	78.62	90.63	95.57	80.72	96.2
LVQ3	86.59	79.26	79.34	84.19	94.36	77.17	96
Simpson's algo.	95.7	88	95.17	95.13	95	88.75	100
Modified algo.	95.7	95.21	95.17	95.13	95	88.75	100

using threshold values of 0.2, 0.1, 0.05, and 0.005 for each of the techniques.

The performance data (% accuracy) are given in Tables I and II. The % accuracy is defined as follows: The algorithm is selected "good" parameters are chosen for it as it is trained on part of the set. The parameters are then used to classify the other part of the set. We report the percent of these classifications that are correct (accurate).

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 (ϵ), 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.

Classical AI Algorithms: As described earlier, these algorithms are implemented in the machine learning library in C++ (MLC++) [30]. Table I shows the performance of these methods on each of the seven data sets. The values of accuracy indicate the performance when training with the larger training set, and with an FSS wrapper inducer.

ID3 performs quite well except for the PYTHIA data set which has mutually nonexclusive 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 91% while the best observed accuracy is 95.83%.) It can also be observed from the above table that the Bayes inducer, Aha-IB, oneR classifier, and the disc-Bayes classifiers fall within a small band of each other. Further, in two out of the seven data sets considered, the OC1 inducer comes up with the second best overall performance.

Training with the smaller training set leads to, as expected, a slight degradation in the performance of the algorithms. Also, training with the FSS wrapper inducer results in better performance for the C4.5, Bayes, disc-Bayes and the OC1 inducers (For instance, the accuracy figures for the PYTHIA dataset with these algorithms are 90, 64.1, 58.35, and 68.12%, respectively, without the FSS inducer and 91, 66, 60, 46, and 70.37%, 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 many as five algorithms give better performance when it is used in conjunction with the smaller training set.

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 one. This schema has the advantage that data sets that contain mutually nonexclusive 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 one. Then the pattern is classified as belonging to the class represented by the augmented vector.

The LOGISTIC procedure, on the other hand, fits linear logistic regression models by the method of maximum likelihood. Like REG, it performs stepwise regression with a choice of forward, backward, and stepwise entry of the variables into the models.

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 nonparametric 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. While the k-NN classifier has been know to give good results in some cases [40], we 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.

See Table II for the performance of these methods. It is seen that the DISCRIM and LOGISTIC procedures consistently out perform the REG procedure. This can be explained as follows [56]: 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. The LOGISTIC procedure is more sophisticated, in its use of link functions that model the "response probability" by logistic terms.

Feedforward NN's: As described in the previous section, feedforward 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. There have been several heuristics proposed to determine an appropriate number of hidden-layer nodes. 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 32 × 10 × 5. 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 significantly 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 until 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. All simulations were performed using the Stuttgart neural-network simulator [65].

The only “free” parameter in the simple back propagation paradigm was the learning rate η and it was varied in the range $[0.1 \cdots 0.9]$. It was observed that the best performance, in terms of classification accuracy, was achieved at η values of 0.8–0.95. Increasing η also led to an decrease in convergence time.

In the variant of BP with momentum, the important parameters are the learning rate η, the momentum coefficient α and the flat spot elimination constant λ. η 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, and 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 η, approximately 0.2. Also, the parameters μ, the maximum growth parameter and ν, the weight decay term influence the performance of Quickprop very much. It was observed that the ideal value of μ was in the range $[1.75 \cdots 2]$ and that for ν 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 feedforward NN’s studied in this article, Rprop provided the best performance for the same number of training iterations. We chose a fixed value of Δ_0 because the algorithm refines it iteratively and we set an upper bound 25 on the weight changes $\Delta_{\max}$. Even though some local minima problems were observed at high values of $\Delta_{\max}$, 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 $(\Delta_0, \Delta_{\max}) = (0.1, 25)$.

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 BP, enhanced BP, and Quickprop methods decline, while that of Rprop consistently maintains a high value.

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 feedforward NN 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.

See Table II for the performance of feedforward NN’s on the seven data sets. It can be seen that the statistics order these algorithms consistently in the following order of improving accuracy: BP, BP 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 BP, 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 five out of seven data sets. Training with the smaller training set instead of the larger leads to the expected degradation of performance.

LVQ Algorithms: 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 one-third 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 0.1–1 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 i.e., the relative “width” of the window into which the training data must fall. We varied the window width parameter from 0.1–0.5 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 ϵ is used (multiplied by the parameter α), 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 ϵ was set at 0.1.

The performance of the LVQ algorithms for the seven data sets is given in Table II. It can be seen that OLVQ1 consistently out performs all the other LVQ algorithms. Also, in five out of the seven 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.

1) *Effect of θ*: In this set of experiments, the max hyperbox size was varied continuously and its effect on other variables were studied. In particular, it is observed that when θ was increased, a lesser number of hyperboxes needed to be formed, i.e., when θ tends to one, 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 θ 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 θ value of around 0.005 for the IRIS data set and 0.003 25 for the PYTHIA dataset. When θ was greater than the "optimal" value so found, the error increased on both the sets and when θ was less, the network *overfit* the training data so that its performance on the test set started to decline.
2) *Effect of δ*: In this experiment, we set θ to the optimal value and we vary δ by assigning to it the values 0.01, 0.02, 0.05, and 0.09. It is observed that when δ 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 δ to be appropriate.
3) *On-Line Adaptation:* The last series of experiments conducted were to test the fuzzy min–max NN 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 one to the optimal number obtained in Item 1). Also, performance on both sets steadily improved to the values obtained in Item 1).

Varying the L_2 error threshold value ϵ was found to not alter the accuracy of the fuzzy min–max network. Table II gives 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 nonexclusive 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. Table II summarizes the classification accuracies of these algorithms.

B. Overall Comparison

Table III provides an overall comparison of the 24 classification algorithms used in this experimental study. 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, compared to the best algorithm.

It is seen that the traditional method using the centroid of the known samples performs very poorly, and the highest accuracy achieved by it on a data set is 61%. The statistical routines performed better, with discriminant analysis faring better than simple forms of regression analysis. Regression using logistic functions performed as well as discriminant 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 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 four of the seven 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 nonexclusive classes.

TABLE III
SUMMARY OF THE RELATIVE PERFORMANCE OF THE 24 CLASSIFICATION ALGORITHMS. THE COUNTS FOR THE BEST AND SECOND BEST PERFORMANCE ARE GIVEN ALONG WITH THE RANGE OF ERROR OBSERVED IN THE CLASSIFICATIONS

Method	Best	Second Best	Range of error
Traditional	—	—	35.00 69.20
Proc REG	—	—	5.80–19.47
Proc LOGISTIC	—	—	4.61–14.28
Proc DISCRIM	—	—	3.02–16.59
ID3	—	—	4.00–21.03
HOODG	—	—	3.40–22.63
Const	—	—	33.00-84.2
IB	3	—	0.00–15.83
C4.5	1	1	0.00–4.83
Bayes	—	—	1.50–29.83
oneR	—	—	2.80–41.83
Aha-IB	—	—	2.80–23.63
Disc-Bayes	—	—	1.10–35.33
OC1	—	2	0.10–25.43
BProp	—	—	2.35–48.53
BProp with momentum	—	—	2.35–23.38
QProp	—	1	0.63–21.58
RProp	3	2	0.00–2.50
LVQ1	—	—	3.80–23.52
OLVQ1	—	—	1.13 15.83
LVQ2		—	1.13–18.58
LVQ3	—	—	2.34–17.86
Simpson's algo.	1	1	0.00–7.83
Modified algo.	1	2	0.00–2.03

Feedforward NN's, in general, performed quite well, with more complicated training schemes like enhanced BP, Quickprop, and Rprop clearly winning over plain error BP. 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 counted for three out of the seven 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 feedforward NN's, 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.

V. 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 $X = \{x_1, x_2, x_3, \cdots, x_n\}$.
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, as well as necessary and sufficient conditions for cluster membership, etc. [4]. Various algorithms for clustering data are also described in [23]. 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, and intensity, which are exhibited by the members of X. It has been recognized [54] 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.

A. 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], [35] 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. 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.

B. 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 multinormal CN model.

VI. 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 misclustered 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) [61].

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

TABLE IV
The Performance (% Accuracy in Clustering) of the Six Clustering Algorithms

Algorithm	IRIS	PYTHIA	Soybean	Glass	Ionosphere	ECG	Wine
CLUSTER	89.33	86.23	90.23	88.317	92.3	91.573	84.09
FASTCLUS	89.33	85.03	90.23	89.25	92.87	91.573	84.09
VARCLUS	89.33	85.63	89.9	88.317	92.023	92.13	84.09
AutoClass	88	85.03	90.55	86.916	92.87	91.57	84.84
Simpson's algo.	90	82.635	89.57	87.38	91.737	91.01	82.57
Multiresolution algo.	91.33	85.03	90.88	87.85	92.87	91.01	84.09

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 nonoverlapping 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.

Table IV 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. Table IV 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.

Nuero–Fuzzy Systems: The two hybrid neuro–fuzzy algorithms discussed were Simpson's fuzzy min–max algorithm and our multiresolution fuzzy clustering algorithm. Table IV gives 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 θ. This is because each hyperbox was labeled as a separate cluster and hence, a lower θ 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 nonexclusive 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 multiresolution pyramid, the echocardiogram data set needed a three-level pyramid to obtain the reported accuracy. The clustering accuracy did not vary with the hyperbox size θ 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 θ.

TABLE V
Summary of the Relative Performance of the Six Clustering Algorithms. The Counts for the Best and Second Best Performance Are Given Along w[w]ith the Range of Error Observed in the Clustering

Method	Best	Second Best	Range of Error
CLUSTER	1	5	0–2
FASTCLUS	2	3	0–2
VARCLUS	1	3	0–2
AutoClass	2	1	0–3.33
Simpson's	—	1	1.12–3
Multiresolution	3	1	0–1.95

A. Overall Comparison

Table V 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 three 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 three 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.

VII. 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. Our detailed experiments also indicate that regardless of the underlying paradigm, the more sophisticated methods tended to out perform the simpler ones. Moreover, the best methods from each paradigm perform about as well as one another, with minor variations depending on the nature of the data. Our neuro–fuzzy techniques are important in this respect, since they tend to be amongst the best performing methods, and have the added advantage of single-pass learning.

References

[1] D. W. Aha, "Tolerating noisy, irrelevant attributes in instance-based learning algorithms," *Int. J. Man-Machine Studies,* vol. 36, no. 1, pp. 267–287, 1992.

[2] S. Amari, "Neural networks: A review from a statistical perspective—Comment," *Statist. Sci.,* vol. 9, no. 1, pp. 31–32, 1994.

[3] P. V. Balakrishnan, M. C. Cooper, V. S. Jacob, and P. A. Lewis, "A study of the classification capabilities of neural networks using unsupervised learning: A comparison with k-means clustering," *Psychometrika,* vol. 59, no. 4, pp. 509–525, 1994.

[4] J. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms.* New York: Plenum, 1981.

[5] M. Boden, *The Philosophy of Artificial Intelligence.* Oxford, U.K.: Oxford Univ. Press, 1990.

[6] H. Braun and M. Riedmiller, "Rprop: A fast and robust backpropagation learning strategy," in *Proc. ACNN,* 1993.

[7] P. Cheeseman and J. Stutz, "Autoclass: A Bayesian classification system" in *Proc. 5th Int. Conf. Mach. Learning.* San Mateo, CA: Morgan Kaufmann, 1988, pp. 55–64.

[8] B. Cheng and D. M. Titterington, "Neural networks: A review from a statistical perspective," *Statist. Sci.,* vol. 9, no. 1, pp. 2–54, 1994.

[9] W. W. Cooley and P. R. Lohnes, *Multivariate Data Analysis.* New York: Wiley, 1971.

[10] J. Daugman, "Pattern and motion vision without Laplacian zero crossings," *J. Opt. Soc. Amer. A,* vol. 5, pp. 1142–1148, 1988.

[11] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Machine Learning: Proc. 12th Int. Conf.,* 1995 [Online]. Available ftp://starry.stanford.edu/pub/ronnyk/disc.ps

[12] N. Draper and H. Smith, *Applied Regression Analysis.* New York: Wiley, 1981.

[13] R. P. W. Duin, "A note on comparing classifiers," *Pattern Recognition Lett.,* vol. 1, pp. 529–536, 1996.

[14] C. Enroth-Cugell and J. Robson, "The contrast sensitivity of retinal ganglion cells of the cat," *J. Physiol.,* vol. 187, pp. 517–522, 1966.

[15] S. E. Fahlman, "Faster-learning variations on backpropagation: An empirical study," in *Proc. 1988 Connectionist Models Summer School,* T. J. Sejnowski, G. E. Hinton, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1988.

[16] R. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics,* vol. 7, no. 2, pp. 179–188, 1936.

[17] E. Gallopoulos, E. Houstis, and J. R. Rice, "Computer as thinker/doer: Problem-solving environments for computational science," *IEEE Computa. Sci. Eng.,* vol. 1, no. 2, pp. 11–23, 1994.

[18] H. H. Harman, *Modern Factor Analysis.* Chicago, IL: Univ. Chicago Press, 1976.

[19] J. A. Hartigan, *Clustering Algorithms.* New York: Wiley, 1975.

[20] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory.* New York: Wiley, 1949.

[21] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine Learning,* vol. 11, pp. 63–90, 1993.

[22] D. O. Hubel, *Eye, Brain, and Vision.* New York: Sci. Amer. Library, 1988.

[23] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

[24] A. K. Jain and J. Mao, "Neural networks and pattern recognition," in *Computational Intell. Imitating Life,* J. M. Zurada, R. J. Marks, II, and E. G. Robinson, Eds. Piscataway, NJ: IEEE Press, 1994, pp. 194–212.

[25] J. M. Jolion and A. Rosenfel, *A Pyramid Framework for Early Vision.* Boston, MA: Kluwer, 1994.

[26] A. Joshi and C. H. Lee, "Backpropagation learns Marr's operator," *Biol. Cybern.,* vol. 70, 1993.

[27] A. Joshi, S. Weerawarana, and E. N. Houstis, "The use of neural networks to support 'intelligent' scientific computing," in *Proc. Int. Conf. Neural Networks, World Congr. Computa. Intell.,* Orlando, FL, vol. IV, 1994, pp. 411–416.

[28] A. Joshi, S. Weerawarana, N. Ramakrishnan, E. N. Houstis, and J. R. Rice, "Neuro-fuzzy support for problem solving environments," *IEEE Computa. Sci. Eng.,* Spring 1996.

[29] R. Kohavi, "Bottom-up induction of oblivious, read-once decision graphs: Strengths and limitations," in *Proc. 12th Nat. Conf. Artificial Intel.,* 1994, pp. 613–618 [Online]. Available FTP://starry.stanford.edu/pub/ronnyk/aaai94.ps

[30] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger, "MLC++: A machine learning library in C++," in *Tools With Artificial Intelligence.* Washington, D.C.: IEEE Comput. Soc. Press, 1994, pp. 740–743 [Online]. Available FTP://starry.stanford.edu/pub/ronnyk/mlc/toolsmlc.ps

[31] T. Kohonen, J. Kangas, J. Laaksoonen, and K. Torkolla, "LVQ-PAK learning vector quantization program package," Lab. Comput. Inform. Sci., Rakentajanaukio, Finland, Tech. Rep. 2C, 1992.

[32] P. Langley, W. Iba, and K. Thompson, "An analysis of Bayesian classifiers," in *Proc. 10th Nat. Conf. Artificial Intell.* Cambridge, MA: MIT Press, 1992, pp. 223–228.

[33] P. Langley and S. Sage, "Induction of selective Bayesian classifiers," in *Proc. 10th Conf. Uncertainty in Artificial Intell.* Seattle, WA, 1994, pp. 399–406.

[34] M. Livingstone and D. O. Hubel, "Segregation of form, color, movement, and depth: Anatomy, physiology, and perception," *Sci.,* vol. 240, pp. 740–749, 1988.

[35] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probability,* 1967, pp. 281–297.

[36] R. J. Marks, II, "Intelligence: Computational versus artificial," *IEEE Trans. Neural Networks,* vol. 4, 1993.

[37] D. Marr and E. Hilderth, "The theory of edge detection," in *Proc. Roy. Soc. London, B,* vol. 207, 1980, pp. 187–217.

[38] J. L. McClelland, "Comment—Neural networks and cognitive science: Motivations and applications," *Statist Sci.,* vol. 9, no. 1, pp. 42–45, 1994.

[39] W. S. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bull. Math. Biophys.,* vol. 5, pp. 115–133, 1943.

[40] D. Michie, D. J. Spiefelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification.* New York: Ellis Horwood, 1994.

[41] P. M. Murphy and D. W. Aha, "Repository of machine learning databases," Univ. California, Irvine, 1994 [Online]. Available http://www.ics.uci.edu/ mlearn/MLRepository.html

[42] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for the induction of oblique decision trees," *J. Artificial Intell. Res.,* vol. 2, pp. 1–33, 1994.

[43] R. M. Nosofsky, "Tests of a generalized MDS-choice model of stimulus identification," Indiana Univ. Cognitive Sci. Program, Bloomington, IN, Tech. Rep. 83, 1992.

[44] Z. Pizlo, A. Rosenfeld, and J. Epelboim, "An exponential pyramid model of the time course of size processing," *Vision Res.,* vol. 35, pp. 1089–1107, 1995.

[45] J. R. Quinlan, "Induction of decision trees," *Machine Learning,* vol. 1, pp. 81–106, 1986.

[46] ———, *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann, 1993.

[47] N. Ramakrishnan, A. Joshi, S. Weerawarana, E. N. Houstis, and J. R. Rice, "Neuro-fuzzy systems for intelligent scientific computing," in *Proc. Artificial Neural Networks Eng. ANNIE '95,* 1995, pp. 279–284.

[48] B. D. Ripley, "Statistical aspects of neural networks," in *Proc. Neural Networks Chaos—Statist. Probabilistic Aspects.* London: Chapman and Hall, 1993, pp. 40–123.

[49] ———, "Neural networks: A review from a statistical perspective—Comment," *Statist. Sci.,* vol. 9, no. 1, pp. 45–48, 1994.

[50] B. D. Ripley, "Neural networks and related methods for classification," *J. Roy. Statist. Soc.,* vol. 56, 1994.

[51] F. Rosenblatt, *Principles of Neurodynamics.* New York: Spartan, 1962.

[52] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* D. E. Rumelhart and J. L. McClelland, Eds., vol. I. Cambridge, MA: MIT Press, 1986.

[53] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge, MA: MIT Press, 1986.

[54] E. Ruspini, "A new approach to clustering," *Inf. Cont.,* vol. 15, pp. 22–32, 1969.

[55] W. S. Sarle, "Neural networks and statistical models," in *Proc. 19th Annu. SAS Users Group Int. Conf.,* 1994.

[56] *SAS/STAT User's Guide: Version 6.* Cary, NC: SAS Instit. Inc., 1990.

[57] I. K. Sethi and A. K. Jain, *Artificial Neural Networks and Statistical Pattern Recognition.* Amsterdam, The Netherlands: North Holland, 1991.

[58] P. K. Simpson, "Fuzzy min–max neural networks—Part 1: Classification," *IEEE Trans. Neural Networks,* vol. 3, pp. 776–786, 1992.

[59] ——, "Fuzzy min–max neural networks–Part 2: Clustering," *IEEE Trans. Fuzzy Syst.,* vol. 1, pp. 32–45, 1993.

[60] M. M. Tatsouka, *Multivariate Analysis.* New York: Wiley, 1971.

[61] J. H. Ward, "Hierarchial grouping to optimize an objective function," *J. Amer. Statist. Assoc.,* vol. 58, pp. 236–244.

[62] S. Weerawarana, E. N. Houstis, J. R. Rice, A. Joshi, and C. E. Houstis, "PYTHIA: A knowledge-based system for intelligent scientific computing," ACM Trans. Math. Software, vol. 22, to appear.

[63] S. Weisberg, *Applied Linear Regression.* New York: Wiley, 1985.

[64] D. Wettschreck, "A study of distance-based machine learning algorithms," Ph.D. dissertation, Oregon State Univ., Corvallis, 1994.

[65] A. Zell, N. Mache, R. Hubner, G. Mamier, M. Vogt, K. Herrmann, M. Schmalzl, T. Sommer, A. Hatzigeorgiou, S. Doring, and D. Posselt, "SNNS: Stuttgart neural-network simulator," Inst. Parallel Distributed High-Performance Syst., Univ. Stuttgart, Germany, Tech. Rep. 3/93, 1993.

Anupam Joshi (S'87–M'89) received the B.Tech degree in electrical engineering from the Indian Institute of Technology, Delhi, in 1989, and the Ph.D. degree in computer science from Purdue University, West Lafayette, IN, in 1993.

From August 1993 to August 1996, he was a member of the Research Faculty at the Department of Computer Sciences at Purdue University. He is currently an Assistant Professor of Computer Engineering and Computer Science at the University of Missouri, Columbia. His research interests include artificial and computational intelligence, concentrating on neuro-fuzzy techniques, multiagent systems, computer vision, mobile and networked computing, and computer mediated learning. He has done work in using AI/CI techniques to help create Problem Solving Environments for Scientific Computing.

Dr. Joshi is a Member of the IEEE Computer Society, ACM, and Upsilon Pi Epsilon.

Narendran Ramakrishnan (M'96) received the M.E. degree in computer science and engineering from the Anna University, Madras, India. He is working toward the Ph.D. degree at the Department of Computer Sciences at Purdue University, West Lafayette, IN.

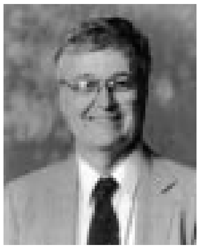
He has worked in the areas of computational models for pattern recognition and prediction. Prior to coming to Purdue, he was with the Software Consultancy Division of Tata Consultancy Services, Madras, India. His current research addresses the role of intelligence in problem solving environments for scientific computing.

Mr. Ramakrishnan is a Member of the IEEE Computer Society, ACM, ACM SIGART, and Upsilon Pi Epsilon.

Elias N. Houstis received the Ph.D. degree from Purdue University, West Lafayette, IN, in 1974.

He is a Professor of Computer Science and Director of the computational science and engineering program at Purdue University. His research interests include parallel computing, neural computing and computational intelligence for scientific applications. He is currently working on the design of a problem solving environment called PDELab for applications modeled by partial differential equations and implemented on a parallel virtual machine environment.

Dr. Houstis is a Member of the ACM and the International Federation for Information Processing (IFIP) Working Group 2.5 (Numerical Software).



John R. Rice received the Ph.D. degree in mathematics from the California University of Technology, Pasadena, in 1959.

He joined the faculty of Purdue University, West Lafayette, IN, in 1964, and was Head of the Department of Computer Sciences there from 1983 to 1996. He is W. Brooks Fortune Professor of Computer Sciences at Purdue University. He is the author of several books on approximation theory, numerical analysis, computer science, and mathematical and scientific software.

Dr. Rice founded the *ACM Transactions on Mathematical Software* in 1975 and remained as its Editor until 1993. He is a Member of the National Academy of Engineering, the IEEE Computer Society, IMACS, and SIAM. He is a Fellow of the ACM and AAAS.