ELSEVIER

Fuzzy Sets and Systems 103 (1999) 201–221

FUZZY sets and systems

Soft computing for feature analysis

Nikhil Ranjan Pal *

Machine Intelligence Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700 035, India

Received May 1998

Abstract

With an introduction to soft computing we discuss how the three main ingredients, fuzzy logic, neural networks and genetic algorithms can play significant roles in the design of successful pattern recognition systems. Then we concentrate only on one aspect of pattern recognition, feature analysis, and discuss various methods using fuzzy logic, neural networks and genetic algorithms for feature ranking, selection and extraction including structure preserving dimensionality reduction. Finally, the methods are illustrated with both real and synthetic data. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Soft computing; Feature selection; Connectionist models; Feature attenuators; Dimensionality reduction; Feature ranking

1. Introduction

A computer can complete a job much more efficiently than a human being when the job involves substantial amount of routine computation like inversion of a matrix of large dimension. On the other hand, if the task requires perceptual power or cognitive capability of human beings, the Von Neumann machine is far behind human beings. For example, human beings can recognize shapes of different sizes, orientations even in an occluded environment much more efficiently than by a computer. Computers are good for well structured precisely formulated problems. Typically, human brains are better for solving real world ill-defined, imprecisely formulated problems requiring huge computations. To overcome the limitations of traditional computing paradigm, scientists are in search of new computational approaches that can be used to solve real world problems efficiently. As a result, in the recent past several novel modes of computing have emerged which are collectively known as *soft computing* [33,46].

As to the understanding of the author, a precise definition of soft computing is yet to emerge. However, as of now soft computing may be viewed as a consortium of various computing tools to exploit the tolerance for imprecision and uncertainty to achieve tractability, robustness and low cost [46]. Usually, it attempts to find an approximate solution to a precisely or imprecisely formulated problem. Neuro computing (NC) is one of the major components of soft computing. The other two important constituents are fuzzy logic (FL) and probabilistic reasoning (PR) with PR subsuming belief networks, genetic algorithms and chaotic systems. FL primarily provides a paradigm for dealing with imprecision and approximate reasoning. NC deals with

* E-mail: nikhil@isical.ac.in.

0165-0114/99/$ – see front matter © 1999 Elsevier Science B.V. All rights reserved.
PII: S0165-0114(98)00222-X

learning and curve fitting. PR, on the other hand, deals with probabilistic uncertainty, propagation of belief, etc.

Pattern recognition may involve two types of data. The first category is called the *object* data while the second one is called *relational* data. In object data a pattern or an object (say, a human being, tank, animal etc.) is characterized by a set of measurements like height, weight etc. On the other hand, in relational representation a set of n objects is represented by an $n \times n$ proximity (similarity/dissimilarity) relation. The proximity relation may be computed from the object representation or could be obtained by experts or by some other means. In this paper, we concentrate only on object data.

There are three major tasks of pattern recognition: *Feature analysis*, *Clustering* and *Classification*. Feature analysis is an essential and important step towards designing effective clustering and classification algorithms. Clustering looks for substructures present in a data set, i.e., it partitions the data set into homogeneous groups. For example, in case of a remotely sensed image, the goal of clustering may be to group the pixels based on gray values and properties of neighboring pixels in such a manner that pixels corresponding to each type of surface (land, vegetation, water etc.) form a separate cluster. Note that, for clustering we do not know the actual type of region a pixel corresponds to, but we expect our clustering algorithm to identify such groups. We emphasize that the process of clustering only finds "homogeneous" groups but cannot say which groups correspond to what. A classifier, on the other hand, partitions the feature space so that any unlabeled data point can be assigned the appropriate class label. A classifier is designed using some training data for which the actual class labels are known. For example, to design an analysis system for remotely sensed images we may be given one or more images for which we know the class label of every pixel; in other words, for every pixel we know whether it corresponds to water, land or vegetation etc. Based on these (training) images we can devise a scheme so that when new images come, we can assign possible class label to every pixel considering "similarity" of the pixels in the new image with the pixels in the training images.

For clustering and classification, as mentioned earlier, objects are usually represented by a set of measurements or feature values each of which characterizes some property of the object. Features could also be qualitative values like red, blue, good, bad etc. Success of a clustering algorithm or of a classifier depends heavily on the discriminating power of these features. A given set of measured features, as it is, may not have enough information to discriminate between different classes. For example, the raw gray values of a digital image are not good features for most applications of image processing. In this case we need to extract features like average gray level, standard deviation, gradients etc. each computed over a neighborhood. (We emphasize here that although image processing is a pattern recognition task, often it requires some special operations because of the spatial relation of pixel values on the digital image. In this article we shall not consider these issues). Too many features are not necessarily good. Some of the features (obtained as a result of measurements or computed from measurements) may be redundant causing extra computational overhead; some of the features may again result in confusion in the feature space leading to degradation in the performance of the pattern recognition system. Feature analysis addresses these issues. It consists of two tasks: feature selection and feature extraction. Feature selection deals with choosing some of the measurable quantities which are important for the problem at hand; while feature extraction computes some new attributes (features) based on some selected measurable quantities.

For the sake of completeness we first briefly introduce the three main ingredients, FL, NN and GA, of soft computing and then explain their role in pattern recognition. Since it is almost impossible to provide, in a single paper, the state of the art in the use of soft computing tools to the main three tasks of pattern recognition we decided to concentrate only on feature analysis. We chose feature analysis because it plays a crucial role to the successful design of both classifier and clustering algorithms.

1.1. Artificial neural networks

Although the concept of artificial neural networks (NN) has been inspired by biological neural networks, the heart of this computing paradigm is rooted in different disciplines. Biological neurons are the structural constituents of the brain and they are much slower than silicon logic gates. But inferencing in biological NN is faster than the fastest computer available today. It is believed that the brain compensates for the relatively slower operation by a really large number of neurons with massive interconnections between them. Biological neural networks have the following features:

- They are highly parallel, robust and fault tolerant nonlinear devices.
- They have built-in capability to adapt their synaptic weights to changes in the surrounding environment.
- They can easily deal with imprecise, fuzzy, noisy and probabilistic information.
- They can generalize from known tasks or examples to unknown ones.

One of the main motivations of Artificial NN is to design problem solving devices incorporating some or all of these characteristics [13, 14, 36]. Although the development of neural networks is inspired by models of brains, the purpose is not just to mimic biological neurons, but to use principles from nervous systems to solve complex problem in an efficient manner. The neuro-computing paradigm is different from programmed instruction sequence. Here information is stored in the synaptic connections, not in the main memory. A neuron is an elementary processor with primitive types of operations, like summing the weighted inputs coming to it and then amplifying or thresholding the sum. The computational neuron model proposed by McCulloch–Pitts is a simple binary threshold unit. The jth neuron computes the weighted sum of all its inputs from other units and outputs a binary value, zero or one, depending on whether this weighted sum is greater than equal or less than a threshold θ_i. Thus

$$x_i(t+1) = f\left(\sum_j w_{ij}x_j(t) - \theta_i\right),$$

where

$$f(x) = \begin{cases} 1 & \text{if } x \geqslant 0, \\ 0 & \text{otherwise.} \end{cases}$$

If the synaptic weight $w_{ij} > 0$ (from neuron j to i), then it is called an excitatory connection; if $w_{ij} < 0$, it is viewed as an inhibitory connection. Often the f is replaced with a more general non-linear function. In principle a network of such neurons is capable of doing quite complex tasks.

A neural network is characterized by the network topology, connection strength between pairs of neurons (weights), node characteristics and the status updating rules. The updating or learning rules may be for weights and/or states of the processing elements (neurons). The adaptability of a neural network comes from its capability of learning from "environments". There are several models of NN which are suitable for different tasks [13, 14, 36]. Some examples are: Hopfield Net (suitable for pattern storage and recall and for optimization), Multilayer Perceptron (classifier and function approximator), Self-Organizing Feature Map (basically does clustering but can be used for generating semantic maps and designing classifiers), Learning Vector Quantization (can be used for clustering), and Adaptive Resonance Theory (clustering) network.

In our subsequent discussion, we shall use only the multilayer perceptron (MLP) network and hence an adequate description of it will be provided in an appropriate place.

1.2. *Fuzzy sets*

Fuzzy sets were introduced in 1965 by Zadeh [45] as a new way to represent vagueness in everyday life. They attempt to model human reasoning/thinking process. Fuzzy sets are generalization of crisp sets and have greater flexibility to capture faithfully various aspects of incompleteness or imperfection in information. For an ordinary set, an element either belongs to it or not; while for fuzzy sets, an element can partially belong to the set; for example, a set of TALL persons [20]. Here there is no precise boundary to the set TALL, and therefore, we cannot really isolate a collection of people labeled as tall. Fuzzy sets are conceptual sets, whose semantic is more important than its mathematical characterization. Mathematically, a fuzzy set is nothing but a mapping (known as membership function) from the universe of discourse X to $[0, 1]$; i.e., $\mu : X \to [0, 1]$.

The set TALL can be modeled by a function shown in Fig. 1(a). This is not the only function that can be used to model the fuzzy set TALL. There may be many other functions and they may not even be continuous but all of them should have the general non-decreasing characteristic to keep the membership function consistent with the semantic of the fuzzy set. Such functions are known as S-type membership functions. We shall

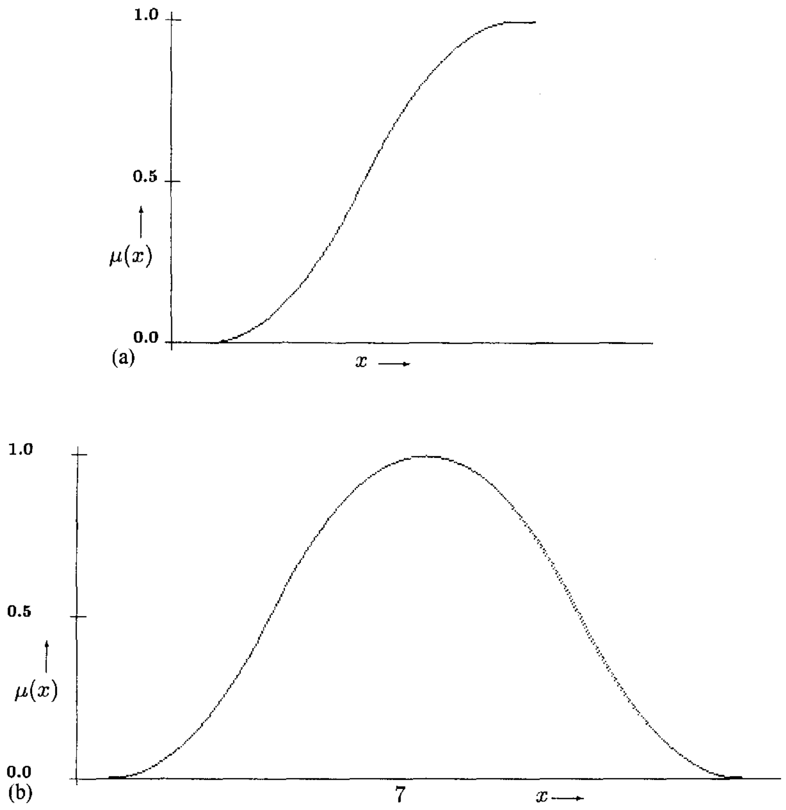

Fig. 1. (a) Fuzzy set TALL. (b) Fuzzy set CLOSE to 7.

show an example of S function in Section 2.2.1. Similarly, fuzzy sets like "CLOSE TO 7" can be modeled by the membership function shown in Fig. 1(b). Again, there could be several other choices. But they should have a membership value of 1 at 7 or over a neighborhood of 7 and then the membership values should decrease as we move away from 7. Such functions are often called Π-type membership functions. These membership functions can be either obtained from an expert or estimated from data.

Since fuzzy sets characterize imprecise properties, they can be effectively used to model vagueness associated with real-life systems. Fuzzy logic is based on the theory of fuzzy sets and approximate reasoning. It is much closer in spirit to human reasoning and natural language than the traditional logical system. Thus, fuzzy logic provides an effective means to model faithfully the approximate and inexact nature of the real world.

1.3. Genetic algorithms

Genetic algorithms [7, 12] (GAs) are another biologically inspired computing tool for optimization. GAs are parallel and randomized search techniques, where a population of solutions evolves over a sequence of generations to possibly a globally optimal solution. Based on a fitness function good solutions are selected for reproduction using two genetic recombination operators: crossover and mutation.

GAs are optimization algorithms which can solve problems resistant to other known optimization methods. They do not require differentiability or continuity of the fitness function. However, if the fitness function is differentiable then this information can be exploited to expedite searching by GAs [3]. Even if the fitness function is not mathematically well structured, they can be used to find an optimal solution. GAs work simultaneously on multiple points in the search space, not on a single point, unlike conventional search techniques. Due to the stochastic characteristic, they have a low chance of getting stuck to a local minimum. Good search algorithms should have the capability of both exploitation and exploration. GAs are believed to support a balanced mixture of both these features. The criterion of "survival of the fittest" provides evolutionary pressure for populations to grow with increasingly fit individuals and thereby exploits good solutions while the crossover and mutation operations enable GAs to explore the entire search space. GAs work on a set of coded solutions not on the solution themselves. Although different coding schemes are possible, binary coding is the most popular. There are many variants, but the basic mechanism of GAs (conventional GAs) consists of the following steps:

1. Start with an initial population (a set of strings/ chromosomes).
2. Evaluate fitness of every string and select candidate strings with probability proportional to fitness value to form the mating pool.
3. Perform crossover and mutation.
4. Repeat steps 2 and 3 until the system ceases to improve, or some stopping criterion is reached.

Each member (which corresponds to a solution of the problem) of the population (i.e., each chromosome) is represented by a fixed length coded string. Selection or reproduction creates the population for the next generation using a probabilistic selection process which offers a string with higher fitness a greater chance of selection. Mutation corresponds to random flipping of one or more bits of an individual string. Mutation increases the diversity in the population and ensures that the probability of attaining any point in the search space is greater than zero. Usually mutation is done with a low probability. The simplest implementation of crossover selects two parents (randomly) from the mating pool and then after choosing a random position each parent string exchanges its tail at that position. The resulting offsprings are included in the population for the next generation. The crossover probability is normally high.

1.4. The role of soft computing in pattern recognition

Any decision making system will have some inputs and some outputs. Usually the inputs are measurements by some sensors. Every measuring instrument has a finite precision. Therefore, with every input value we have an inherent imprecision. For example, if a sensor with two digit precision reads 10.53, then the actual value may not (usually will not) be exactly equal to 10.53 but it is something CLOSE TO 10.53 – a fuzzy concept. Thus fuzzy set is a natural tool to model such vagueness. Now consider another example, an image analysis system for remotely sensed images. For such an image each pixel may represent a surface area of even $20 \times 20\,\mathrm{m}^2$. As a result part of a pixel may correspond to, say, land and while the rest may represent water. Therefore, while segmenting the image if we make a hard decision (either water or land) we are bound to commit some error and then in the later stage of the interpretation we may not be able to recover this mistake. Incorporation of fuzziness (fuzzy segmentation) here can result in a more meaningful and useful system. Depending on the characteristics of the neighboring pixels, the pixel under consideration may be assigned memberships to different classes. Thus we see that fuzzy sets can be used both at the input level and also during processing. Now consider another problem of designing a classifier to discriminate between painters and singers. Suppose we have data for a person who is partially a painter and partially a singer, then our classifier should be able to provide such information. Conventional classifiers cannot provide such details. However, fuzzy sets can be quite effective to model such unsharp decision boundaries between classes, i.e., at the output level of a decision making system. There are several other ways in which FL can be used in pattern recognition, like fuzzy reasoning system for classifier and so on [5, 15, 44].

One of the distinctive features of FL is that it can model the imprecision associated with real-life situations, while given a problem we find a computational neural network model to solve the same. For example, when the input information for a classifier is imprecise or vague, we can use fuzzy sets to model them. On the other hand, given a problem of designing a classifier, we can easily use a multilayer perceptron network, or to find the "homogeneous" sub-groups in a data set we can use Kohonen's self-organizing feature map or Learning Vector Quantization. For some other problems we may need to design problem specific new neural architectures also. Often some of the common architectures can be easily modified to solve some problems for which the original net was not designed. We shall see later how the MLP or its variants can be used for feature analysis although MLP was originally designed to be a classifier. A natural question then arises, what do we gain out of these? Well, we may achieve robustness, parallelism, fault tolerance and often better performance (generalization) than the traditional methods.

Irrespective of whether a problem is modeled using fuzzy logic or a neural network, often finding of solutions becomes equivalent to solving an optimization problem. Some such examples are: choosing an "optimal" architecture for an MLP for a given task, finding the parameters defining membership functions of a fuzzy rule-based system or selecting a small but adequate rule set to solve a problem. Sometimes classical gradient based optimization schemes are not suitable, as in the case of finding an "optimal" set of rules for a fuzzy classifier. Since, GAs, do not require derivatives/continuity of the objective function, in principle, GAs can be used to solve any such optimization problems. Sometimes searching with GAs can be made faster with judicious use of gradient information maintaining the stochastic nature of GAs [3].

Readers should not get the false impression that GAs can solve every problem efficiently. For example, if the searching is done in the real domain and the number of parameters to be identified are reasonably large (sometimes 15 parameters may even be large), GAs may not be an efficient choice. In such a case the guidelines may be, if you have some other reasonably good search technique, use that. For example, use of GAs to optimize the fuzzy c-means [2] objective function even for the IRIS [1] data may not stand in comparison to the usual alternating optimization scheme.

With this background we now concentrate on the feature analysis problem.

2. Feature analysis

Feature analysis may be represented by an implicit or explicit mapping $f: R^p \Rightarrow R^q$ where for feature selection $q < p$; in this case f simply selects some features and for feature extraction (computation of additional features from the given features), $q > p$. One can of course view feature selection as a special case of feature extraction. Often a part of the feature analysis task is called dimensionality reduction. When feature analysis suggests a set of q $(q < p)$ features which has the necessary information to accomplish the task at hand (i.e., the q features can be used in place of the p original features) it may be called dimensionality reduction. These set of features may be obtained by selection or computed from the raw measurements.

We emphasize here the fact that the quality of a feature is dependent on the type of problem or classifier we use to evaluate it. For example, the most important feature for training an MLP may be different from the most important feature for a nearest prototype classifier.

There are many techniques for feature selection. Some of these techniques are based on interclass and intraclass distances [9, 11], some are based on neural networks [8, 28, 39] while some others use genetic algorithms [29, 42]. Similarly, for feature extraction there are several methods including principal component analysis [16, 21, 25, 30, 38, 40].

2.1. Neural networks for feature analysis

Since majority of the connectionist schemes that we are going to present are based on MLP or its variant, for the sake of completeness we provide a brief description of the backpropagation algorithm.

2.1.1. The backpropagation algorithm

A multi-layer perceptron net can be trained to learn the relation between a set of inputs and outputs. Each node of a hidden layer is connected to every node in its immediately preceding and immediately following layers. At each node all incoming signals (weight multiplied by the output of the connecting node in the previous layer) are summed algebraically to give the total input, which is then transformed by a non-linear activation function. The backpropagation (BP) learning algorithm updates the connection weights with a view to minimizing the total square error over the whole training data.

We use the following symbols in our subsequent discussion. Let x_i be the ith component of an input vector $\boldsymbol{x}$ in the training set, O_i^k be the output corresponding to the ith node of the kth hidden layer for the input vector $\boldsymbol{x}$, T_i be the desired output for the ith output node corresponding to input vector $\boldsymbol{x}$, and W_{ij}^k be the weight connecting the jth node of the kth layer to the ith node of the $(k+1)$th layer, $k=0,1,\ldots,n$. Here $k=0$ corresponds to the input layer, n is the total number of hidden layers and $(n+1)$th layer is the output layer. Also let f be the activation function, $f_i'^k$ be the value of the derivative of the activation function at the ith node of the kth hidden layer, ε be the error for the input $\boldsymbol{x}$, and η be the learning rate.

The BP algorithm consists of two passes: the forward pass and the backward or weights adjustment pass. The forward pass computes the output of each node. The output in the first hidden layer is computed as, $O_i^1=f(\sum_j x_j W_{ij}^0)$. The outputs computed by the nodes in the kth $(k=2,\ldots,n)$ layer are given by $O_i^k=f(\sum_j O_j^{k-1}W_{ij}^{k-1})$. The output from the output layer $n+1$ can be written as

$$O_i^{n+1}=f\left(\sum_j O_j^n W_{ij}^n\right)=Z_i. \tag{1}$$

In the backward pass, weights are adapted to minimize $\sum\varepsilon$ using the gradient descent on ε in (2) generated by each input vector in the training set:

$$\varepsilon=\frac{1}{2}\sum_i (E_i)^2=\frac{1}{2}\sum_i (T_i-Z_i)^2. \tag{2}$$

Using the gradient descent method the weight corrections for the output layer can be shown as

$$\Delta W_{ij}^n=\eta\delta_i^{n+1}O_j^n, \tag{3}$$

where

$$\delta_i^{n+1}=E_i f_i'^{n+1}. \tag{4}$$

Similarly, using the chain rules the weight updates for the hidden layers can be written as

$$\Delta W_{ij}^k=\eta\delta_i^{k+1}O_j^k, \tag{5}$$

where,

$$\delta_j^k=f_j'^k\sum_i(\delta_i^{k+1}W_{ij}^k). \tag{6}$$

The incremental changes ΔW_{ij}^k may be summed up over all patterns in the training set and the weights W_{ij}^k may be updated with the resulting sums (batch mode), or the weights may be updated for each pattern (on line mode).

2.1.2. Feature extraction

(a) *Neural nets for PCA.* Principal component analysis (PCA) is a linear orthogonal transform from p-dimensional space to q-dimensional space $(q\leqslant p)$, such that the co-ordinates of the data in the new q-dimensional space are uncorrelated and maximal amount of variance of the original data is preserved by only a small number of co-ordinates [17].

Suppose we have a linear transform from a p-dimensional zero-mean input vector $\boldsymbol{x}=(x_1,x_2,\ldots,x_p)^{\mathrm{T}}$ to a q-dimensional output vector $\boldsymbol{y}=(y_1,y_2,\ldots,y_q)^{\mathrm{T}}$ and $\boldsymbol{y}$ is related to $\boldsymbol{x}$ by the expression $\boldsymbol{y}=W\boldsymbol{x}$ where W is a $q\times p$ matrix, with $q\leqslant p$. PCA sets the q successive rows of W to the q eigenvectors corresponding to the q largest eigenvalues of the input covariance matrix $S=E(\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}})$ Thus, y_1 represents the component of $\boldsymbol{x}$ in the direction of the largest eigenvector of S, y_2 is the component in the direction of the 2nd largest, and so on.

Let W_n and W_{n+1} be the W matrices computed with $X_n=\{\boldsymbol{x}_i\in\mathcal{R}^p\colon\ i=1,2,\ldots,n\}$ and $X_{n+1}=X_n\cup\{\boldsymbol{x}_{n+1}\in\mathcal{R}^p\}$, respectively. Then,

$$\boldsymbol{y}_{n+1}=W_{n+1}\boldsymbol{x}_{n+1}\neq W_n\boldsymbol{x}_{n+1}=\boldsymbol{y}_{n+1}'. \tag{7}$$

As long as $\boldsymbol{x}_{n+1}$ is not widely different from the vectors used to compute W_n, $\boldsymbol{y}_{n+1}\approx\boldsymbol{y}_{n+1}'$. Thus W_n can do a good job of projecting new data points, as long as the data points used to compute W_n adequately represent the population generating $\boldsymbol{x}_i\in\mathcal{R}^p$. Due to unavoidable computational complexity with the conventional approaches, especially when p is very large, neural network approaches for PCA have been widely studied recently. A variety of neural networks and learning algorithms have been proposed for PCA and its variants [22–25,37,38]. Most of them are based on the early work of Oja's one-unit algorithm [23,25]. We discuss here only one of them as a representative.

Rubner's PCA network. The PCA network proposed by Rubner et al. [37,38] consists of an input layer with p nodes and an output layer with q nodes. The two layers are completely interconnected. Let the connection weight between input node i and output node j be denoted by w_{ji}. All the output nodes are hierarchically organized in such a way that the output node i is connected to the output node j with connection strength u_{ij} if and only if $j<i$. The set of weights connecting an output node j to all input nodes forms the

weight vector $\boldsymbol{w}_j$, the transpose of which is the jth row of the weight matrix W. Let $\{\boldsymbol{x}_k=(x_{k1},x_{k2},\ldots,x_{kp})^{\mathrm{T}}, k=1,\ldots,n\}$ be the set of n input vectors with zero-mean and $\{\boldsymbol{y}_k=(y_{k1},y_{k2},\ldots,y_{kq})^{\mathrm{T}}, k=1,\ldots,n\}$ be their corresponding output vectors produced by the network, as computed by

$$y_{kj}=\langle \boldsymbol{w}_j,\boldsymbol{x}_k\rangle+\sum_{l<j}(u_{jl}\times y_{kl}). \tag{8}$$

The weights between the two layers are adjusted upon presentation of an input pattern $\boldsymbol{x}_k$ according to the Hebbian rule,

$$\boldsymbol{w}_j(t+1)=\boldsymbol{w}_j(t)+\eta\boldsymbol{x}_k y_{kj} \quad (j=1,\ldots,q). \tag{9}$$

The lateral weights adapt themselves according to the anti-Hebbian rule,

$$u_{jl}(t+1)=u_{jl}(t)-\mu y_{kj}y_{kl} \quad (l<j), \tag{10}$$

where η and μ are positive learning coefficients. Note that (9) updates a complete weight vector, while (10) updates only one weight.

Often a momentum term is added to each of (9) and (10) to expedite the learning. Rubner and Tavan [38] proved that if the learning parameters η and μ are chosen according to

$$\frac{\eta(\lambda_1-\lambda_p)}{\lambda_1(1+\eta\lambda_p)}<\mu<\frac{2}{\lambda_1} \tag{11}$$

then this learning rule forces the lateral weights to vanish and the activities of the output cells to become uncorrelated. Correspondingly, the weight vectors $\boldsymbol{w}_j$ converge to the eigenvectors of the covariance matrix S. Although in practice, it is difficult to determine the values of η and μ according to the inequality in (11), without computing the eigenvalues, (11) does provide a range for the values of η and μ if λ_1 and λ_p can somehow be estimated.

The PCA network has the same level generalization abilities as that of W computed with the eigenvectors of S and hence is able to project new data as expected when the original data have linear relationship. *However, PCA networks and learning algorithms have some limitations that diminish their attractiveness*: (i) *Standard PCA networks are able to realize only linear input–output mappings.* (ii) *The PCA networks cannot usually separate independent subsignals from their linear mixture.*

To overcome these drawbacks PCA networks containing nonlinear units are gaining attention [18,24]. Also Independent Component Analysis (ICH) has been introduced as an interesting extension of PCA in context with the signal separation problem [6].

(b) *Neural net for Sammon's nonlinear projection.*

Sammon's method: Sammon's [40] nonlinear projection algorithm (SM) attempts to preserve the structure by finding n points in q-space such that their inter-point distances approximate the corresponding inter-point distances in p-space.

Let $X=\{\boldsymbol{x}_k \mid \boldsymbol{x}_k=(x_{k1},x_{k2},\ldots,x_{kp})^{\mathrm{T}}, k=1,2,\ldots,n\}$ be the set of n input vectors and let $Y=\{\boldsymbol{y}_k \mid \boldsymbol{y}_k=(y_{k1},y_{k2},\ldots,y_{kq})^{\mathrm{T}}, k=1,2,\ldots,n\}$ be the unknown vectors to be found. Let $d^*_{ij}=d(\boldsymbol{x}_i,\boldsymbol{x}_j), \boldsymbol{x}_i,\boldsymbol{x}_j\in X$ and $d_{ij}=d(\boldsymbol{y}_i,\boldsymbol{y}_j)$, $\boldsymbol{y}_i,\boldsymbol{y}_j\in Y$, where $d(\boldsymbol{x}_i,\boldsymbol{x}_j)$ be the Euclidean distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Sammon suggested looking for Y minimizing

$$E=\frac{1}{\sum_{i<j}d^*_{ij}}\sum_{i<j}\frac{(d^*_{ij}-d_{ij})^2}{d^*_{ij}}. \tag{12}$$

Sammon used the method of steepest descent for (approximate) minimization of E. Let $\boldsymbol{y}_i(t)$ be the estimate of $\boldsymbol{y}_i$ at the tth iteration $\forall i$. Then $\boldsymbol{y}_i(t+1)$ is given by

$$y_{ij}(t+1)=y_{ij}(t)-\alpha\left[\frac{\partial E(t)}{\partial y_{ij}(t)}\Bigg/\left|\frac{\partial^2 E(t)}{\partial y_{ij}(t)^2}\right|\right], \tag{13}$$

where the non-negative scalar constant α is the step size for gradient search.

With this method we cannot get an explicit mapping function governing the relationship between patterns in p-space and corresponding patterns in q-space. Therefore, it cannot project new points. It also involves a large amount of computation, as every step within an iteration requires the computation of $\frac{1}{2}n(n-1)$ distances. The algorithm becomes impractical for large n. Finally, the algorithm usually gets stuck in a local minimum.

Connectionist implementation of Sammon's method: Jain and Mao [16,21] used the multilayer perceptron network with an error function defined in a different manner for Sammon's projection. The number of input and output nodes are set to p and q, respectively. An MLP needs an error function to drive the backpropagation algorithm. As such for Sammon's method the target value is not known. To realize the Sammon's error, the net is given a pair of data points, say $\mathbf{x}_i, \mathbf{x}_j \in \mathscr{R}^p$ *one after another as input*. Let the corresponding outputs of the net be $\mathbf{y}_i, \mathbf{y}_j \in \mathscr{R}^q$. Once $\mathbf{y}_i, \mathbf{y}_j$ are known, d^*_{ij} and d_{ij} and consequently

$$E_{ij} = \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}} \tag{14}$$

can be defined. Jain and Mao used gradient descent on E_{ij} with a view to minimizing Sammon's error function. The process is repeated with randomly selected pairs till convergence of the net.

It was shown experimentally that the number of nodes required in the hidden layer is to be around nq to get good results. *This method requires a lot of space and training time to get good solutions.*

In [21], a different approach was followed for training so as to take advantage of the nonlinearity of the above network. Initially a PCA network is used to project data and then standard backpropagation algorithm is used to approximate principal components. The weights of this trained MLP are then used to initialize the weights of the Sammon's net.

We offer the following remarks about this implementation: (i) *training time is high*, (ii) *memory usage is high*, (iii) *to try a different (new) architecture, an MLP with the same new architecture should again be trained to approximate principal components for weight initialization i.e., we cannot directly add any extra hidden layer even if it is demanded*; *in fact we cannot even add an extra node and* (iv) *the main purpose of this network is to handle nonlinear data, as, linear data is very well projected by the PCA network, but, even this may not be achieved by the proposed implementation, as shall be seen from the results.*

Another new connectionist scheme: Sammon's algorithm and some of its derivatives work very well for small data sets [10,40]. As mentioned earlier Sammon's method cannot project new data points and is computationally prohibitive for large data sets. These problems can be eliminated, if we can get a mapping function governing the relationship between patterns in p-space and patterns in q-space, by projecting a small representative subset of the data.

We proposed a very simple method [30], which performs better (at least on the examples we tried) than methods given in [16,21] in terms of *time*, *space* and *quality of the projected map*. This method combines the advantages of Sammon's method for projecting small data sets and capabilities of MLP for function approximation. We call this method SAPRONN – Sammon's projection with neural networks.

When we talk about projection of unknown data based on a mapping (explicit or implicit) estimated from a given data set, we implicitly assume that the given data have some structure which future data points are expected to follow. In other words, we can assume that the data points are generated from some time invariant (unknown) probability distribution. Therefore, if we can extract a small but adequate representative sample of the given data set and then estimate the mapping function based on these we can expect to have a good generalization.

In fact, although not explained or stated, this was also the philosophy behind the Jain and Mao's method. It then raises two issues: How to get an adequate but small sample and what do we do with that!

We propose to select a small subset $X^{(S)}$ of representative data points using SRSWOR (simple random sampling without replacement) scheme so that statistical characteristics of X are retained by $X^{(S)}$ based on χ^2 or divergence statistic [43]. Our computational exercise shows that 30% data points are usually enough. Now we run Sammon's algorithm on $X^{(S)}$ to generate $Y^{(S)} \subset \mathscr{R}^q$. Then we use $(X^{(S)}, Y^{(S)})$ to train an MLP. Note that such a trained MLP will capture the structure present in $X^{(S)} \subset \mathscr{R}^p$. Jain and Mao used 50% of the data points first to train a PCA network and used that net to initialize Sammon's network. Unlike Jain and Mao, in our scheme the relation is captured by the pair $(X^{(S)}, Y^{(S)})$ and an MLP simply learns it. In this scheme it is easy to try different NN architectures.

Next we provide a schematic description of the algorithm:

Algorithm SAPRONN ( )

```
{
  Input $X = \{\mathbf{x}_i \in \mathscr{R}^p\colon i = 1, 2, \ldots, n\}$;
  Normalize $\mathbf{x}_i$ to get $\mathbf{x}_i^{(N)}$, $i = 1, 2, \ldots, n\}$;
  Let $X^{(N)} = \{\mathbf{x}_i^{(N)}\colon i = 1, 2, \ldots, n\}$;
  Select a random sample $X^{(S)}$, of size $n_s$ by
  SRSWOR-scheme from $X^{(N)}$
```

```
    using $\chi^2$ or divergence or like Jain & Mao a
    random sample of size $n_s = n/2$;
  Run Sammon_projection with $X^{(S)}$ to get
  $Y^{(S)} = \{y_i^{(S)} \in \mathscr{R}^q: i = 1,2,\ldots,(n_s)\}$
    where $y_i^{(S)}$ corresponds to $x_i^{(S)}$;
  Normalize $y_i^{(S)}$ to get $y_i^{(N)}$, $i = 1,2,\ldots,(n_s)\}$;
  Let $Y^{(N)} = \{y_i^{(N)}: i = 1,2,\ldots,(n_s)\}$;
  Train an MLP with $X^{(S)}$ and $Y^{(N)}$,
  $y_i^{(N)}$ is the target corresponding to $x_i^{(S)}$;
}
```

Use this trained MLP to project the complete data set $X^{(N)}$ and any new data points.

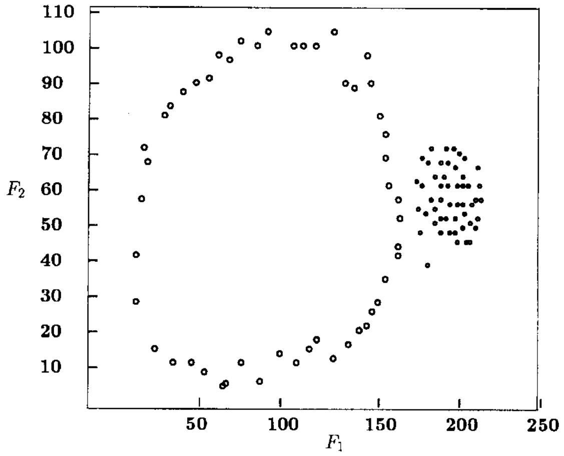

Fig. 2. The scatterplot of a 2-dimensional data set.

2.1.3. *Feature ranking & selection*

When an MLP is reasonably trained we can examine the sensitivity of the net's output with respect to input for finding important features. Based on this philosophy we discuss two methods.

Saliency based feature ranking (*SAFER*): Ruck et al. [39] possibly were the first to propose use of sensitivity of output of the network to its input for ranking of input features. The expression for feature *saliency* measure as proposed by them is

$$\Lambda_j = \sum_{x \in \mathscr{S}} \sum_k \sum_{x_j \in D_j} \left| \frac{\partial o_k(x, W)}{\partial x_j} \right|, \tag{15}$$

where D_j is the set of values for the jth feature that will be sampled and o_k is the output of the kth output node. $\mathscr{S}$ is the training set. The matrix W is an array of all connection weights in the network arranged in some suitable form. They used sum of the absolute values of the derivative as an indicator of the sensitivity of the output of the network with respect to the input feature. Therefore, $\Lambda_j > \Lambda_i$ is assumed to indicate that the importance of the jth feature is more than that of the ith feature.

For evaluating $\partial o_k(x, W)/\partial x_j$ in (15) the chain rule can be used as discussed for the backpropagation algorithm.

To reduce the computational load, Ruck et al. suggested to sample the data at the most important points. The points of greatest importance in the input space are those for which training data exist; hence, the training vectors are used as starting points to sample the input space. For every training vector, each feature is sampled over its range to compute the saliency.

Note that the method of sampling data points in [39] sometimes may mislead the scheme. Let us take a pattern set in two dimension as in Fig. 2.

Fig. 2 has two classes viz. class 1 (left) and class 2 (right). Consider a pattern vector x in the training set from class 1. If the value of feature 1 (F_1) is kept fixed and that of F_2 is varied over its range, some of the points may be generated outside of both classes 1 and 2. The network is neither trained with these pattern points nor do these points belong to any of the two classes. Therefore, incorporation of these points in calculating the feature saliency may mislead the process of ranking. Further details in this regard can be found in [8].

Sensitivity based feature ranking (*SEFER*). After an MLP successfully learns a data set, the weights of the links are expected to be so adjusted that the value of a redundant (less importance) feature will not influence the output vector much. Lesser the importance of a feature in discriminating between classes, lower would be the influence of its value on the output of the network. SEFER is banked on this concept [8].

Using the trained MLP, for every feature q we compute a feature quality index, FQI_q and then rank the features according to FQI_q. To compute FQI_q we proceed as follows: For each training data point x_i, $i = 1,2,\ldots,n$ we set x_{iq} to zero. Let this modified data point be denoted by $\boldsymbol{x}_i^{(q)}$; i.e., $x_{ij}^{(q)} = x_{ij}\ \forall j \neq q$ and $x_{iq}^{(q)} = 0$. Setting the qth component to zero is equivalent to delinking the qth input node and hence delinking all connections associated directly with the qth input node. Thus, the impact of the qth feature will not reach any node of the network. Let the output vectors obtained for $\boldsymbol{x}_i$ and $\boldsymbol{x}_i^{(q)}$ be $\boldsymbol{o}_i$ and $\boldsymbol{o}_i^{(q)}$, respectively. Note that $\boldsymbol{o}_i$ is *not* the target output corresponding to $\boldsymbol{x}_i$, but the *actual* output that is obtained for $\boldsymbol{x}_i$ from the trained net. For a less important feature, the output vectors $\boldsymbol{o}_i$ and $\boldsymbol{o}_i^{(q)}$ are not expected to differ much. Any function of $\boldsymbol{o}_i$ and $\boldsymbol{o}_i^{(q)}$ that can measure this variation between the two can be used as an index for feature ranking. A very simple choice would be to define

$$FQI_q = FQI_q^t = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{o}_i - \boldsymbol{o}_i^{(q)}\|_t$$
$$= \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{c} |\boldsymbol{o}_{ij} - \boldsymbol{o}_{ij}^{(q)}|^t\right)^{1/t}. \tag{16}$$

Here $t > 1$ and c is the number of classes. After computing FQI_q^ts for all p features, they can be ranked according to their importance as $q_1, q_2, \ldots, q_p$ when $FQI_{q1}^t \geqslant FQI_{q2}^t \geqslant \cdots \geqslant FQI_{qp}^t$.

Another interesting choice could be to use the symmetric divergence function of Kullback FQI_D,

$$FQI_D = \sum_{j=1}^{c} (\boldsymbol{o}_{ij} - \boldsymbol{o}_{ij}^{(q)}) \log(\boldsymbol{o}_{ij}/\boldsymbol{o}_{ij}^{(q)}). \tag{17}$$

Note that, FQI_D cannot be called entropy as o_{ij}s are not probability and it is not a metric also.

If the problem is to select k $(k < p)$ best features (feature selection), best from the point of view of discrimination between classes, the feature set $\{q_1, q_2, \ldots, q_k\}$ may not be the optimal set. But $q_1, q_2, \ldots, q_k$ will definitely represent a good subset of features. However, the best set of k features can be obtained by evaluating FQI setting every possible subset of k features to zero.

In SEFER we find the output of the net after removing a feature and then measuring the deviation of this output from the learnt output but not from the target output. We have not considered the target output because the network might not have been able to learn the target output to a desirable level. It is more logical to consider the sensitivity with respect to what has been learnt by the network. Moreover, setting a feature value to zero is equivalent to assuming absence of that feature. Thus, it is a conservative approach. SAFER ranks individual features but cannot select the best subset of $k < n$ features. But SEFER can rank the features individually as well as select the best subset of $k < n$ features.

An attenuator based feature selection (*AFES*). In a standard multilayer perceptron network, the effect of some features (inputs) can be eliminated by multiplying them with zero and the rest with unity before they propagate into the network. They can be made effective again by changing these “multipliers” or “feature attenuators” from zero to unity. The binary version of these “attenuators” can be further generalized into continuous “attenuation functions”, whose range is [0, 1]. In AFES the inputs are attenuated by their corresponding “attenuation functions” before they pass into the network. Parameters of the attenuation functions are also trained by the gradient descent method along with the connection weights [28]. At the end of the training input features with a high attenuation can be eliminated.

In addition to the symbols introduced earlier, we use the following:

Let F be the attenuation function, F_i' be the derivative of the attenuation function associated with the ith input node, M_i be the argument of the attenuation function associated with the ith input node, μ be the learning rate of the attenuator, and A_i be the attenuation of the ith feature $= 1 - F(M_i)$.

In the forward pass corresponding to an input vector $\boldsymbol{x}$ we get the attenuated vector $\boldsymbol{x}'$ after the attenuation has occurred for each input feature (x_i) as,

$$x_i' = F(M_i)x_i. \tag{18}$$

To realize (18) we may assume that the ith node in the input layer has an activation function $f_i^o(x) = xF(Mi)$, with tunable M_i, $i = 1, 2, \ldots, p$. Thus, the output O_i^1 for the first hidden layer becomes,

$$O_i^1 = f\left(\sum_j x_j' W_{ij}^0\right). \tag{19}$$

Propagating the signal further into the network we have,

$$O_i^k = f\left(\sum_j O_j^{k-1} W_{ij}^{k-1}\right), \tag{20}$$

where $k = 2,\dots,n+1$. The final output, as before, is given by O_i^{n+1}.

In the backward pass, weights and parameters of the attenuation functions are adapted with a view to minimizing $\sum \varepsilon$ using the gradient descent on ε generated for each input vector in the data set.

As before,

$$\varepsilon = \frac{1}{2}\sum_i (E_i)^2. \tag{21}$$

The weights of the network are adjusted exactly in the same manner as described earlier except for the minor changes given below.

$$\Delta W_{ij}^0 = \eta\delta_i^1 x_j' = \eta\delta_i^1 x_j F(M_j). \tag{22}$$

The learning rule for the attenuators (M_i) can be shown [28] to be

$$\Delta M_i = \mu x_i F_i' \sum_j (W_{ji}^0 \delta_j^1). \tag{23}$$

Here the attenuation for the ith feature is given by, $A_i = 1.0 - F(M_i)$.

If A_i is close to 1.0 i.e., when $F(M_i)$ is close to zero, $x_i F(M_i)$ will have values close to zero. Under such a situation the feature will not pass into the network. On the other hand, when A_i is close to 0.0 ($F(M_i)$ nearly equal to 1.0), $x_i F(M_i)$ will have values close to x_i, and hence, the feature passes almost unattenuated into the network. The training starts with all attenuation functions set to almost zero value, i.e., $A_i = 100\%$. Thus, at the beginning of the training, practically none of the features is allowed to pass into the network. As the network trains, it selectively allows only some important features to be active by increasing their attenuator values as dictated by the gradient descent. The training can be stopped when the network has classified satisfactorily, i.e., the number of mis-classifications has gone down to a tolerable value and/or the error is low. Features with high attenuation may be eliminated from the feature set.

2.2. *Fuzzy sets for feature analysis*

There are not many attempts to feature analysis using fuzzy logic. We just illustrate here a few such approaches for feature ranking. Let $X = \{x_1, x_2, \dots, x_n\} \subset R$ be the universe of discourse and a fuzzy set $\mathscr{A} = \{\mu_{\mathscr{A}}(x_i)/x_i \mid x_i \in X;\ i = 1,2,\dots,n;\ \mu_{\mathscr{A}} \in [0,1]\}$ be defined on X where $\mu_{\mathscr{A}}(x_i)$ denotes the membership of x_i to $\mathscr{A}$. A measure of fuzziness for $\mathscr{A}$ can be defined as [27]

$$H(\mathscr{A}) = k\sum_{i=1}^{n} f(\mu_{\mathscr{A}}(x_i)), \tag{24}$$

where k is a constant and the function $f(\cdot)$ can be defined in various ways [27]. One can obtain the fuzziness measure suggested by Deluca–Termini using

$$f(\mu_{\mathscr{A}}(x_i)) = -\mu_{\mathscr{A}}(x_i)\ln(\mu_{\mathscr{A}}(x_i)) - (1-\mu_{\mathscr{A}}(x_i))\ln(1-\mu_{\mathscr{A}}(x_i)) \tag{25}$$

in Eq. (24). $H(\mathscr{A})$ with (25) is also called entropy of the fuzzy set. Thus, the entropy becomes,

$$H(\mathscr{A}) = \frac{1}{n\ln 2}\sum_{i=1}^{n}\{-\mu_{\mathscr{A}}(x_i)\ln(\mu_{\mathscr{A}}(x_i)) - (1-\mu_{\mathscr{A}}(x_i))\ln(1-\mu_{\mathscr{A}}(x_i))\}, \tag{26}$$

where $k = 1/n\ln 2$ is the normalization factor. Pal and Chakraborty used Eq. (26) for feature ranking [32]. $H(\mathscr{A})$ attains the maximum value when $\mathscr{A}$ is most fuzzy, i.e., when $\mu_{\mathscr{A}}(x_i) = 0.5\ \forall i$ and it attains the minimum value when $\mu_{\mathscr{A}}(x_i) = 0$ *or* $1\ \forall i$. Pal and Chakraborty used S-type and π-type [32] membership functions for modeling of μ. Let us consider only the standard S-function, defined as

$$\mu_{\mathscr{A}}(x_i; a,b,c) = \begin{cases} 0, & x_i \leqslant a, \\ 2\left[\dfrac{x_i - a}{c - a}\right]^2, & a \leqslant x_i \leqslant b, \\ 1 - 2\left[\dfrac{x_i - c}{c - a}\right]^2, & b \leqslant x_i \leqslant c, \\ 1, & x_i \geqslant c \end{cases} \tag{27}$$

in the interval $[a,c]$ with $b = (a+c)/2$. The parameter b is known as the crossover point for which $\mu_{\mathscr{A}}(x; a,b,c) = 0.5$.

Let $X = \{x_1, x_2, \ldots, x_n\} \subset R^p$, be the given data set where each x_i is from one of the c classes, i.e., each x_i has a class label $\mathscr{C}_j$ which it comes from.

Let av, max and min be the average, maximum and the minimum value of x_{qj}, respectively, of the jth feature for class k.

Define [32]

$$b = (x_{qj})_{\mathrm{av}}, \tag{28}$$

$$c = b + \max\{|(x_{qj})_{\mathrm{av}} - (x_{qj})_{\max}|, \\ |(x_{qj})_{\mathrm{av}} - (x_{qj})_{\min}|\}, \tag{29}$$

and

$$a = 2b - c. \tag{30}$$

Compute H (in (26)) of the class $\mathscr{C}_j$ for the qth feature using (28)–(30). Now for the qth feature of class k if each x_{qj} is equal to b, H will be maximum and equal to 1; H tends to zero as x_{qj} moves away from b towards either c or a. The higher the value of H, the greater would be the number of samples having $\mu(x) \approx 0.5$ and hence greater would be the tendency of the samples to cluster around its mean value, resulting in less (internal) scatter within the class. If we pool together the classes $\mathscr{C}_j$ and $\mathscr{C}_k$ and compute the mean, maximum and minimum values of the qth feature over all $(n_j + n_k)$ samples where n_r $(r = j, k)$ is the number of samples in class $\mathscr{C}_r$, H for the pooled sample would decrease as the goodness of feature increases. This is because, for a good feature, the samples from both classes should be away from the overall mean, i.e., most of the points will have $\mu(x) \approx 0$ *or* 1. The feature evaluation index for feature q, (FEI_q), can thus be defined as [32]

$$FEI_q = \frac{H_{qjk}}{H_{qj} + H_{qk}}, \tag{31}$$

where H_{qjk} is the value of the entropy for feature q after pooling the classes $\mathscr{C}_j$ and $\mathscr{C}_k$; H_{qj}, H_{qk} are those for the feature q computed for $\mathscr{C}_j$ and $\mathscr{C}_k$, respectively. The lower the value of FEI_q, the better is, therefore, the quality of the qth feature in characterizing and discriminating classes $\mathscr{C}_j$ and $\mathscr{C}_k$. Instead of using only one feature q, FEI can be calculated even for a set of features [31]. In this case, we need to use a multidimensional membership function [34]. Note that, instead of H in (26) any measure of fuzziness [27] can be used.

This method can be used to assess features for a pair of classes only. It may happen that a feature p is good in discriminating $\mathscr{C}_i$ and $\mathscr{C}_j$, while feature q may be a better discriminator for classes $\mathscr{C}_k$ and $\mathscr{C}_l$. Further, it may happen that some other feature r is, on an average, a better discriminator for all the classes $\mathscr{C}_i$, $\mathscr{C}_j$, $\mathscr{C}_k$ and $\mathscr{C}_l$ taken together. Thus, with FEI it may be difficult to assess the goodness of a feature keeping in view all classes taken together.

To get around this problem, Pal [31] extended his earlier work to define the average importance of a set of features $\mathscr{S}$ as

$$(FEI)^{\mathrm{av}}_{\mathscr{S}} = \sum_j \sum_k W_j W_k (FEI)^{(jk)}_{\mathscr{S}}, \tag{32}$$

where $W_j = n_j/n_t$, $W_k = n_k/n_t$, $n_t = \sum_j n_j$, $j, k = 1, 2, \ldots, c$; $k \neq j$, are weight factors.

Here the weights are nothing but the a priori probabilities of different classes. Hence, $(FEI)^{\mathrm{av}}$ depends on the number of points in a class and this may not be desirable. Preferably, $(FEI)^{\mathrm{av}}$ should depend only on the structure of the classes but not on the number of points in a class. Suppose $n_j + n_k = \psi$ (a constant) for two different pairs of classes. Here $W_j W_k$ attains the maximum value when $n_j = n_k = \psi/2$. Thus, $(FEI)^{\mathrm{av}}$ is biased towards equiprobable classes but this is not desirable. With a view to relaxing this bias a new index, called *overall feature evaluation index* or *OFEI* is defined in [8].

The objective of *OFEI* is to account for some of the issues just discussed. Feature q will be good if it can discriminate every pair of the c classes. Therefore, the goodness of a feature q increases as H_{qjk} $(j, k=1, 2, \ldots, c$ and $j \neq k)$ decreases and H_{qj} $(j=1, 2, \ldots, c)$ increases; i.e., $\sum_{j,k=1, j\neq k}^{c} H_{qjk}$ decreases and $\sum_{j=1}^{c} H_{qj}$ increases. Thus, the overall feature evaluation index for feature q $(OFEI_q)$ can be defined as

$$OFEI_q = \frac{\sum_{j,k=1, j\neq k}^{c} H_{qjk}}{\sum_{j=1}^{c} H_{qj}}. \tag{33}$$

If *OFEI* is low, we can expect the associated feature to be better. It may happen that $H_{qij} < H_{rij}$ but $H_{qkl} > H_{rkl}$, i.e., feature q is more important to discriminate classes i and j than feature r but the converse is true for classes k and l. Since Eq. (33) considers all possible pairs of classes, $OFEI_q$ will reflect the overall (average) discriminating power of the feature q. Note that, $OFEI_q$ does not directly depend on the size of a class.

2.3. Genetic algorithms for feature analysis

There have been a few attempts to solve the feature selection problem using GA. Siedlecki and Sklansky [42] used k-NN rule to find a small subset of features for which the classifier's performance does not deteriorate below a specified level. They did this by constructing a GA chromosome consisting of a binary string whose length equaled the number of features. If a bit is "1", that feature is selected for evaluating the performance of the classifier.

Kelly and Davis [19] and Punch et al. [35] solved the same feature selection problem using GA. Unlike Siedlecki and Sklansky they multiplied each feature by a real-valued weight and then used that weighted feature for computing distances required for implementation of k-NN classifier. GAs have been used to learn these weights. Features with high values for the learned weights are considered important features and vice-versa.

These methods cannot be used to select a fixed (given) number of good features i.e., say q, good features. The algorithm may terminate at a point where the total number of 1's in the solution string may not be equal to q in [42]; while for other two methods [19,35] those q features having highest weights can be selected. But this can create another problem. Suppose, there is a feature which is more or less constant for all classes. For this feature whatever be the weight the classifier performance will not change. Since GAs are probabilistic search techniques, the algorithm might terminate at a point with high weight for this indifferent feature and thereby indicating a false importance.

In order to maintain a fixed number of 1's in a chromosome, yet keeping the evolutionary characteristic of GA, Pal et al. [29] proposed a new crossover operator, named self-crossover. Unlike conventional crossover, self-crossover alters the genetic information within a *single* potential string selected randomly from the mating pool to produce an offspring. This is done in such a manner that the stochastic and evolutionary characteristics of GAs are preserved.

Let

$$S = 00010010011001011011 \tag{34}$$

be a string of length 20 selected from the mating pool. For self-crossover, first we select a random position p $(0 < p < L)$ and generate two substrings s_1 and s_2: $s_1 =$ bits 1 through p of S and $s_2 =$ bits $p+1$ through L of S. Now we select two random positions p_1, $0 \leqslant p_1 \leqslant p$ and p_2, $0 \leqslant p_2 \leqslant (L-p)$. Then four substrings are generated as follows:

$$s_{11} = \text{bits 1 through } p - p_1 \text{ of } s_1,$$
$$s_{12} = \text{bits } (p - p_1 + 1) \text{ through } p \text{ of } s_1,$$
$$s_{21} = \text{bits 1 through } L - p - p_2 \text{ of } s_2,$$
$$s_{22} = \text{bits } (L - p_2 + 1) \text{ through } L \text{ of } s_2.$$

Using operations similar to crossover we generate $S^1 = s_{11} \mid s_{22}$ and $S^2 = s_{21} \mid s_{12}$. Finally, the self-crossovered offspring of S is generated as $S_1 = S^1 \mid S^2$. It is easy to see that number of 1's in S and S_1 is the same. Let us now explain it with the example string S in (34).

A random position, $p = 9$, is selected for splitting the string into two substrings (s_1, s_2) as follows:

$$s_1 = 000100100 \quad \text{and} \quad s_2 = 11001011011.$$

Now two random positions, $p_1 = 4$ and $p_2 = 7$, are selected for s_1 and s_2, respectively. After splitting s_1 and s_2 at 4th and 7th position, respectively, we get,

$$s_{11} = 00010; \quad s_{12} = 0100; \quad s_{21} = 1100;$$

and

$$s_{22} = 1011011.$$

Now two new substrings S^1 and S^2 are then obtained as

$$S^1 = 000101011011 \quad \text{and} \quad S^2 = 11000100.$$

Finally, the offspring (S_1) is generated by concatenating S^1 and S^2 as

$$S_1 = 00010101101111000100.$$

Thus, self-crossover exchanges substrings s_{12} and s_{22}. If the parent string consists of all 0's or all 1's, the offspring generated through self-crossover will resemble its parent because of the underlying constraint on the total number of 1's in the string. If we do not start GA with a all "1" or all "0" string, GA with self-crossover technique, will never generate such strings as offsprings. Self-crossover evolves to new offsprings as iterations go on.

It can be easily shown that self-crossover (without mutation) can generate any target string [29]. However, the result does *not* say that there is no need for mutation in GA with self-crossover technique. It simply says that for problems like TSP, use of self-crossover without mutation can generate all possible valid solution strings. For problems like feature selection, data editing for NN classifier where we want to select a good subset of features or data points of a prefixed cardinality, self-crossover without mutation is sufficient. Conventional mutation for such problems may produce invalid solutions, i.e., it may generate a substring of arbitrary cardinality, not equal to the prefixed cardinality.

At the first sight, it might appear that self-crossover is a parallel random search, but this is not the case for two reasons. Self-crossover is done only on a randomly selected subset of strings and self-crossover does not alter the substring s_{11}. It exchanges, only s_{22} and s_{12}. Consequently, through selection & crossover the evolutionary characteristics of GA are preserved. The similarity between the parents and offsprings will be more if we take $p_1 = p_2 = p'$ (say) = a random number selected between 1 and $\mathrm{Min}(p, L - p)$; i.e., $0 < p_1 = p_2 = p' < \mathrm{Min}(p, L - p)$. In this case, the bits in positions 0 through p' and bits from $p + 1$ through $L - p'$ will remain unaltered. Consequently, the evolutionary pressure will be more.

Let us denote the p features as $F_1, F_2, \ldots, F_p$. We have to select a set of q features, say $\{F_{i1}, F_{i2}, \ldots, F_{iq}\} \subset \{F_1, F_2, \ldots, F_p\}$ such that the selected feature subset can do different pattern recognition jobs well. To use GA for feature selection we need an objective (fitness) function to guide the feature selection process. The fitness function should reflect the performance of the reduced data set for different pattern recognition tasks. For an unlabeled (where class information is not available) data set the fitness function may reflect the performance of a clustering algorithm; while for labeled data (where class information is available) the fitness function may be defined to measure the performance of a classifier. Here we consider the latter case and the fitness function is defined to be the performance of the nearest prototype (NP) classifier. Thus the fitness function f is given by $f(F_{i1}, F_{i2}, \ldots, F_{iq}, Y_q, V) =$ No. of correct classification, where $Y_q = \{y_1, y_2, \ldots, y_n\}$, $y_i \in R^q$ and the kth component of y_i, i.e., y_{ik} is equal to some lth component x_{il} of $x_i \in R^p$; $V = \{v_1, v_2, \ldots, v_c\}$, $v_i \in R^q$ is the set of q dimensional prototypes defined by

$$v_i = \frac{1}{|\mathscr{C}_i|} \sum_{k \in \mathscr{C}_i} y_k, \tag{35}$$

where c is the number of classes and $\mathscr{C}_i$ denotes the ith class. Note that the prototypes may be generated in many other ways.

A feature subset is now represented by a binary string of length p. A set of M binary strings of length p and cardinality k is taken as the initial population where the cardinality of a binary string is defined as the total number of 1's in the string. If the ith position of the string contains a "1" then the ith feature is selected for the chosen subset. Thus, a string of cardinality k denotes a feature subset of size k. Now the iterations of GA are continued with self-crossover, evaluation and selection with probability proportional to fitness. The entire process is repeated for a desired number of times or till we find no improvement in the fitness value for several generations.

3. Results

We present our results summarized into two subsections, one for feature ranking and selection, and the other for dimensionality reduction. For feature selection and ranking we have implemented the algorithms discussed on several data sets including both synthetic as well as real data sets, but we report here only results on two of them, *Crude-oil* and *Mango-leaf* for the feature selection algorithms and to show the effectiveness of the feature extraction algorithm we consider a synthetic data set, *Sphere-Shell* and the well known *IRIS* data.

Crude-oil [17] has five features and 56 data points and Mango-leaf [4] has eighteen features and 166 data points. Both have three classes. The Sphere-Shell [22], on the other hand, consists of 1000 points in 3-space. 500 points are selected randomly within a hemisphere of radius $r1$ and rest 500 from a shell defined by two hemispheres of radius $r2$ and $r3$, such that

Table 1
Results with SAFER, SEFER, $(\text{FEI})^{\text{av}}$ and OFEI on Crude-oil

Feature no./ Feature made 0	SAFER			SEFER	$(\text{FEI})^{\text{av}}$	OFEI
	Run 1	Run 2	Run 3			
1	1	1	1	1	5	5
2	5	5	5	5	3	3
3	4	3	2	4	4	4
4	2	4	4	2	2	2
5	3	2	3	3	1	1

Table 2
Results with AFES for Crude-oil for 6000 iterations

Features	M_i	$A_i \cdot 100$
1	*5.82	*0.29
2	5.52	99.60
3	*6.12	*0.22
4	*4.90	*0.74
5	*6.62	*0.13
Misclassification	2	
Error	0.023	

Table 3
Results of conventional MLP on Crude-oil

Features	Iteration	Misclassifications	Error
All	5000	3	0.0289
1, 2, 3, 4	5000	4	0.0315
1, 2, 3, 5	5000	2	0.0170
1, 2, 4, 5	5000	2	0.0156
*1, 3, 4, 5	5000	0	0.0002
2, 3, 4, 5	5000	1	0.0154
1, 3, 5	5000	4	0.0411
2, 3, 5	5000	9	0.0703
3, 5	5000	12	0.1053
2, 5	5000	7	0.0764
4, 5	5000	7	0.0655

$r1 < r2 < r3$. IRIS [1,17] is a four dimensional data with 150 points in three classes.

3.1. *Results on feature ranking and selection*

In our implementation all features are normalized to the same scale by a transformation. For each feature x' the transformed value x is obtained as $x = (x' - k1)/(k - k1)$, where $k1 = \min_i \min_j \{x'_{ij}\}$ and $k = \max_i \max_j \{x'_{ij}\}$. Note that this transformation does not change the structure of the classes as it is only a change of scale and origin of the entire data.

For the MLP based algorithms we used the *standard sigmoid* for both attenuator and activation functions. "On-line" method was employed for training. One complete pass through the data was considered to be one epoch or iteration. The initial values of the "attenuation functions" should be ideally zero, this was practically achieved by setting M_i to -5.0 which corresponds to $F(M_i) = 0.006699$, i.e., an attenuation of 99.33%. A feature is considered important if its attenuation is low.

We compared our results with Ruck et al.'s scheme for which we provided rankings for three typical runs. The network architectures, learning rates and the number of iterations were kept the same for all schemes. We authenticated our results by running the conventional MLP with different feature subsets.

3.1.1. *Results for Crude-oil*

For Crude-oil [17] an architecture with six nodes in a single hidden layer is found to be adequate for an MLP.

Table 1 reports the results obtained from three typical runs of SAFER and also the ranks obtained by SEFER, FEI and OFEI. The second feature has consistently been ranked the last for all experiments that we conducted with SAFER and SEFER, thus indicating that it is the least important feature. Later we shall see that AFES conforms to this but the ranking of other features does not agree with that of AFES. For a few runs not reported in Table 1 both methods are found to rank feature 3 as the least important one; while both of the fuzzy indices produce significantly different ranking.

Results of a typical run of AFES on Crude-oil are given in Table 2. In Table 2 (and also in Table 3) asterisks (*) are used to indicate features with low attenuation. Table 2 reveals that features 1, 3, 4 and 5 are the important ones as their attenuations are very low at the end of the training. Several different initializations gave the *same* final result. For the run shown in Table 2 the initialization was such that feature 2 was activated first, however, later it was eliminated which tells us that this feature is causing confusion and is a harmful one. These results were ratified by running the standard MLP on various feature subsets. We report this in Table 3.

From Table 3, we find that only the feature subset (1, 3, 4, 5) can result in zero misclassification just in 1000 iterations and it is the best subset of features of size four. Among the combinations involving four features, the one lacking feature 5 has a higher error than the rest, indicating the importance of this feature in comparison to the rest. This is also reflected by the lowest value of the attenuation factor for feature 5 in Table 2.

Feature sets involving 4 and 5 have given a better performance than the rest, thereby indicating that feature 5 is the most important one followed by feature 4. The absence of feature 2 shows improvement in the performance of the classifier indicating its deleterious contribution. The column labeled Error in Table 3 represents the average square error per class.

3.1.2. Results on Mango-leaf

This data set [4] has 18 features corresponding to three kinds of Indian mangoes. The results of SAFER, SEFER and $(FQI)^{av}$ and OFEI are presented in Table 4. Ranking produced by three runs of Ruck et al.'s method and that by others are different but are highly correlated. We shall see that these rankings are significantly different from the results suggested by AFES.

Table 5 shows the results obtained by AFES which are found to be consistent over several initializations. Always features 2 and 3 had the minimum attenuations in comparison to the rest. Though feature 3 consistently gave a lower attenuation than that of feature 2 for all runs, the difference in attenuations was small. Feature 6 and feature 9 were consistently close to each other and always next to features 2 and 3. Most of the times feature 6 was found to have a lower attenuation than that of feature 9. However, feature 9 gave lower values of attenuation than that of feature 6 for some initializations. Like other data sets for Mango-leaf also we ran the conventional MLP with different subsets of features and it is reported in Table 6.

For most runs of SAFER the following four features 17, 6, 12 and 18 (listed in order) are found to be the most important. To authenticate this we ran conventional MLP (Table 6) with feature subsets (17), (17, 6), (17, 6, 12) and (17, 6, 12, 18). We found that the most important feature subset (2, 3) suggested by AFES is much better than (6, 17). Similarly, (2, 3, 6) is a much better choice than (6, 12, 17). Similar experiments with the ranks suggested by others showed that AFES is the best among the three methods discussed.

Table 4
Results with SAFER, SEFER, $(FEI)^{av}$ and FQI on Mango-leaf

Feature no./ Feature made 0	SAFER			SEFER	$(FEI)^{av}$	FQI
	Run 1	Run 2	Run 3			
1	17	16	14	9	7	7
2	11	11	12	5	10	9
3	13	12	16	6	16	16
4	6	6	4	1	3	1
5	9	9	10	13	1	2
6	2	2	1	7	13	13
7	16	18	17	17	6	5
8	18	17	18	18	5	4
9	8	8	2	2	9	12
10	14	15	8	4	12	11
11	7	7	7	8	18	17
12	3	3	9	10	14	15
13	5	5	13	11	17	18
14	12	13	11	15	4	6
15	10	10	5	3	2	3
16	15	14	15	16	11	10
17	1	1	3	12	15	14
18	4	4	6	14	8	8

We also used GA with self-crossover for feature selection. When the cardinality of the chromosomes were fixed at 2, 3 and 4 the feature subsets selected by the scheme are $\{9, 14\}$, $\{9, 13, 14\}$ and $\{9, 13, 14, 17\}$, respectively. These feature subsets are found to be quite good in terms of number of misclassifications produced by nearest prototype classifiers designed on them.

3.2. Results on dimensionality reduction

3.2.1. For Sphere-Shell and IRIS

For feature extraction or structure preserving dimensionality reduction algorithms, as mentioned earlier, we used two data sets: Sphere-Shell and IRIS.

Table 7 shows that for IRIS original Sammon's method requires much less time than either of Jain and Mao's method and SAPRONN, with SAPRONN requiring about half time of Jain and Mao's method. In fact, for any small data set Sammon's method is expected to perform better than the neural

Table 5
Results of the multiplier based method on Mango-leaf

Iterations	1000		2000		3000		5000	
Features	M_i	$A_i{\cdot}100$	M_i	$A_i{\cdot}100$	M_i	$A_i{\cdot}100$	M_i	$A_i{\cdot}100$
1	*1.75	*14.75	*4.03	*1.74	*4.22	*1.45	*5.06	*0.63
2	*4.52	*1.08	*5.21	*0.54	*5.84	*0.29	*6.70	*0.12
3	*3.37	*3.32	*5.44	*0.43	*6.06	*0.23	*6.83	*0.11
4	−4.76	99.15	−2.47	92.24	−2.94	94.96	−3.70	97.58
5	−4.98	99.32	−4.63	99.03	−3.44	96.89	*4.76	*0.84
6	−4.97	99.31	−3.59	97.31	*5.19	*0.56	*6.65	*0.13
7	−4.99	99.33	−4.99	99.33	−4.99	99.33	−4.99	99.33
8	−4.99	99.33	−5.0	99.32	−5.00	99.33	−4.99	99.33
9	*3.03	*4.62	*6.51	*0.15	*6.70	*0.12	*6.68	*0.13
10	−4.77	99.16	−4.80	99.18	−1.95	87.50	−3.62	97.40
11	−4.97	99.31	*1.00	*26.86	*3.48	*3.00	*4.61	*0.98
12	−4.96	99.31	−4.08	98.34	*4.36	*1.27	*6.05	*0.24
13	−4.97	99.31	−4.61	99.01	−3.20	96.1	*5.55	*0.39
14	−4.99	99.32	−4.84	99.22	−4.62	99.02	−3.87	97.96
15	−4.89	99.25	−3.83	97.87	−2.21	90.13	−3.41	96.79
16	−4.99	99.33	−4.98	99.32	−4.97	99.31	−4.93	99.28
17	−4.98	99.32	−2.52	92.58	*4.57	*1.03	*6.10	*0.22
18	−4.99	99.33	−4.86	99.23	−4.56	98.97	*3.25	*3.72
Misclassifications	24		21		15		11	
Error	0.162		0.114		0.096		0.078	

Table 6
Results of conventional MLP on Mango-leaf

Features taken	Iterations	Misclassifications	Error
All	5000	13	0.1301
2, 3	5000	25	0.1301
3, 9	5000	34	0.1476
3, 6	5000	35	0.1697
2, 3, 6	5000	18	0.1047
2, 3, 9	5000	20	0.1134
2, 3, 6, 9	5000	19	0.0609
2, 3, 6, 9, 12	5000	19	0.1007
2, 3, 6, 9, 17	5000	19	0.1025
2, 3, 6, 9, 12, 17	5000	19	0.1023
2, 3, 6, 9, 12, 13, 17	5000	15	0.0893
7, 8	5000	48	0.1998
7, 8, 2	5000	47	0.1868
2, 3, 9, 17	5000	19	0.1091
17	5000	48	0.5953
6, 17	5000	48	0.5669
6, 12, 17	5000	28	0.4219
6, 12, 17, 18	5000	31	0.4130

Table 7
CPU time (seconds) for various methods

Data	Sammon's method	Jain and Mao's method	SAPRONN
IRIS	205.6	1288.5	589.0
Sphere-Shell	8995.8	7611.3	5508.6

Table 8
Values of Sammon's error for various methods

Data	Sammon's method	Jain and Mao's method	SAPRONN
IRIS	0.00659	0.01252	0.06173
Sphere-Shell	0.00089	0.09	0.03

implementation in terms of both CPU time and Sammon's error. For IRIS the scatterplot of the two dimensional projections are similar and hence not displayed here. For Sphere-Shell SAPRONN required much less CPU time than both Sammon's method and Jain and Mao's scheme, and the performance (in terms of Sammon's error) of SAPRONN is quite comparable to that of Jain's method (Table 8). For Sphere-Shell SAPRONN works better than the method of Jain and Mao. This is also revealed by Fig. 3. In Jain's method some of the projected points

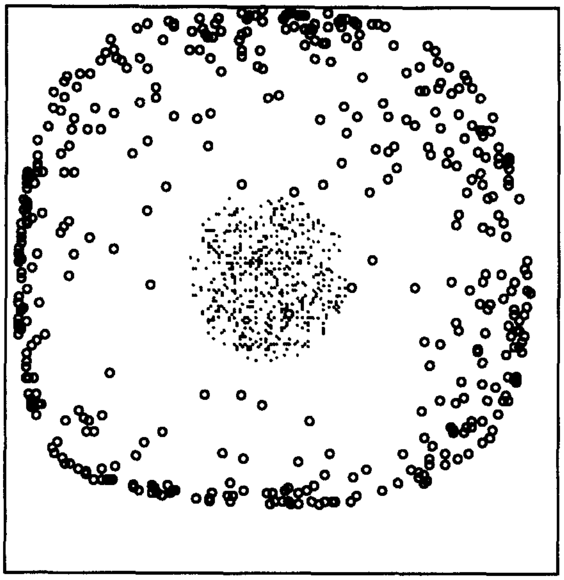

Fig. 3. Sphere-Shell by SAPRONN.

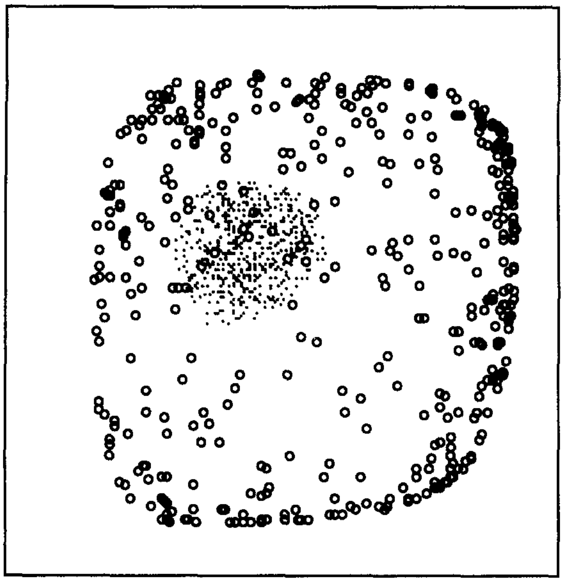

Fig. 5. Sphere-Shell by Jain and Mao Algorithm.

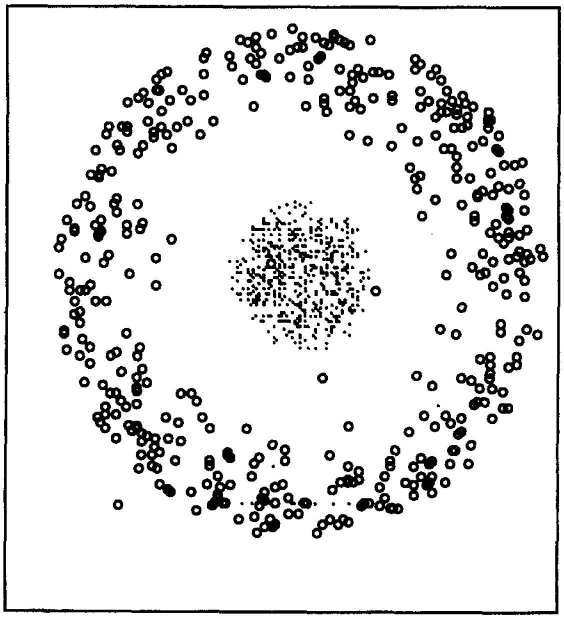

Fig. 4. Sphere-Shell by Sammon's algorithm.

corresponding to the outer shell got mixed up with projected points corresponding to central hemisphere (circle represents points from the shell while dot (.) indicates points in the hemisphere).

4. Conclusions and discussion

We discussed the main ingredients of soft computing and explained how they can help in the design of effective pattern recognition systems. We presented several methods based on soft computing for feature analysis. In particular, we discussed how GA, fuzzy logic and NN can be used for feature ranking and selection. Of the various feature selection/ranking schemes, AFES is found to be the best. To avoid the computational overhead of Sammon's method and to realize a dimensionality reduction system with predictability, SAPRONN integrates the tools of statistics, Sammon's function and neural networks in a novel manner. We also presented a few other methods for neural realization of Sammon's scheme. Most of the methods have been illustrated with synthetic as well as real data.

An interesting area where further investigation could be done for feature extraction (particularly for dimensionality reduction) would be the use of neuro-fuzzy approaches. As is well known, MLP picks up one of many possible generalizations (equivalently settles to one of several local minima) which may not be the desirable one. Consequently, even when Jain and Mao's method or SAPRONN works well for the data used to train the net, it can seriously fail for new data points. This chance of very bad generalizations can possibly be reduced drastically with the help of multi-layered neuro-fuzzy architectures. If the neuro-fuzzy system, maintains the logical reasoning structure of a fuzzy reasoning system yet exploits the features of connectionist models, the chance of very bad generalization is reduced significantly.

Acknowledgements

The author gratefully acknowledges Dr. S. Mitra of this Institute for her valuable suggestions to improve the manuscript and Mr. E. Vijaykumar of Vedika International Pvt. Ltd., Calcutta for his help in producing some of the computational results.

References

[1] E. Anderson, The irises of the Gaspe Peninsula, Bull. Amer. IRIS Soc. 59 (1935) 2–5.

[2] J.C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum, New York, 1981.

[3] D. Bhandari, N.R. Pal, S.K. Pal, Directed mutation in genetic algorithms, Inform. Sci. 79 (1994) 251–270.

[4] A. Bhattacharjee, Some aspects of mango (Mangifora Indica L) leaf growth features in varietal recognition, Master's Thesis, Calcutta University, Calcutta, 1986.

[5] S.L. Chiu, Extracting fuzzy rules for pattern classification by cluster estimation, in: Proc. 5th Internat. Fuzzy Systems Assoc., World Congress (IFSA'95), Sao Paulo, Brazil, 1–4 July 1995.

[6] P. Comon, Independent component analysis – a new concept?, Signal Processing 36 (3) (1994) 287–314.

[7] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

[8] R. De, N.R. Pal, S.K. Pal, Feature analysis: neural network and fuzzy set theoretic approaches, Pattern Recognition 30 (10) (1997) 1579–1590.

[9] P.A. Devijver, J. Kittler, Pattern Recognition, A Statistical Approach, Prentice-Hall, London, 1982.

[10] D.H. Foley, J.W. Sammon, An optimal set of discriminant vectors, IEEE Trans. Comput. 24 (1978) 271–278.

[11] K. Fukunaga, W.L.G. Koontz, Application of the Karhunen–Loeve expansion to feature selection and ordering, IEEE Trans. Comput. 19 (1970) 311–318.

[12] D.E. Goldberg, Genetic Algorithms: Search, Optimization and Machine Learning, Addison-Wesley, New York, 1989.

[13] S. Haykin, Neural networks – a comprehensive foundation, Macmillan, New York, 1994.

[14] J. Hertz, A. Krogh, R.G. Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley, Redwood City, CA, 1991.

[15] H. Ishibuchi, K. Nozaki, N. Yamamoto, H. Tanaka, Selecting fuzzy if–then rules for classification problem using genetic algorithms, IEEE Trans. Fuzzy Systems 3 (3) (1995) 260–270.

[16] A.K. Jain, J. Mao, Artificial neural networks for nonlinear projection of multivariate data, in: Proc. IEEE Internat. Joint Conf. on Neural Networks, vol. 3, 1992, pp. 59–69.

[17] R.A. Johnson, D.W. Wichern, Applied Multivariate Statistical Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[18] J. Karhunen, J. Joutsensalo, Representation and separation of signals using nonlinear PCA type learning, Neural Networks 7 (1994) 113–127.

[19] J.D. Kelly, Jr., L. Davis, A hybrid genetic algorithm for classification, in: Internat. Joint Conf. on Artificial Intelligence, Darling Harbour, Sydney, Australia, vol. 2, 1991, pp. 645–650.

[20] G.J. Klir, B. Yuan, Fuzzy Sets and Fuzzy Logic – Theory and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1995.

[21] J. Mao, A.K. Jain, Artificial neural networks for feature extraction and multivariate data projection, IEEE Trans. Neural Networks 6 (2) (1995) 296–317.

[22] H. Niemann, Linear and nonlinear mapping of patterns, Pattern Recognition 12 (1980) 83–87.

[23] E. Oja, A simplified neuron model as a principal component analyzer, J. Math. Biol. 15 (1982) 267–273.

[24] E. Oja, J. Karhunen, Nonlinear PCA: algorithms and applications, Report A18, September 1993, Laboratory of Computer and Information Science, Helsinki University of Technology, Espoo, Finland.

[25] E. Oja, Neural networks, principal components, and subspaces, Internat. J. Neural Systems 1 (1989) 61–68.

[26] E. Oja, Principal components, minor components and neural networks, Neural Networks 5 (1992) 927–936.

[27] N.R. Pal, J.C. Bezdek, Measuring Fuzzy Uncertainty, IEEE Trans. Fuzzy Systems 2 (2) (1994) 107–118.

[28] N.R. Pal, K. Chintalapudi, A connectionist system for feature selection, Neural, Parallel Sci. Comput. 5 (3) (1997) 359–381.

[29] N.R. Pal, S. Nandi, M.K. Kundu, Self-crossover: a new genetic operator and its application to feature selection, Int. J. System Sci. 29 (2) (1998) 207–212.

[30] N.R. Pal, E. Vijay Kumar, Neural networks for dimensionality reduction, in: Kasabov et al. (Eds.), Progress in Connectionist-Based Information Systems, Proc. 4th Internat. Conf. on Neural Information Processing, ICONIP'97, vol. 1, Springer, New York, 1997, pp. 221–224.

[31] S.K. Pal, Fuzzy set theoretic measures for automatic feature evaluation: ii, Inform. Sci. 64 (1992) 165–179.

[32] S.K. Pal, B. Chakraborty, Fuzzy set theoretic measures for automatic feature evaluation, IEEE Trans. Systems Man Cybernet. 16 (1986) 754–760.

[33] S.K. Pal, N.R. Pal, Soft computing: goals, tools and feasibility, J. IETE 42 (4–5) (1996) 195–204.

[34] S.K. Pal, P.K. Pramanik, Fuzzy measures in determining seed points in clustering, Pattern Recognition Lett. 4 (1986) 159–164.

[35] W.F. Punch, E.D. Goodman, M. Pei, L.C. Shun, P. Hovland, R. Embody, Further research on feature selection and

classification using genetic algorithms, in: Proc. 5th Internat. Conf. on Genetic Algorithms, University of Illinois at Urbana-Champaign, 1993, pp. 557–564.

[36] R. Rosenfeld, J. Anderson (Eds.), Neuro Computing, MIT Press, Cambridge, MA, 1988.

[37] J. Rubner, K. Schulten, Development of feature detectors by self organization, Biol. Cybernet. 62 (1990) 193–199.

[38] J. Rubner, P. Tavan, A self-organizing network for principal component analysis, Europhys. Lett. 10 (1989) 693–698.

[39] D.W. Ruck, S.K. Rogers, M. Kabrisky, Feature selection using a multilayer perceptron, J. Neural Network Comput. (1990) 40–48.

[40] J.W. Sammon Jr., A nonlinear mapping for data structure analysis, IEEE Trans. Computers C-18 (1969) 401–409.

[41] B. Schachter, A nonlinear mapping algorithm for large databases, Comput. Graphics Image Process. 7 (1978) 271–278.

[42] W. Siedlecki, J. Sklansky, A note on genetic algorithms for large-scale feature selection, Pattern Recognition Lett. 10 (1989) 335–347.

[43] N.R. Pal, E.V. Kumar, Two efficient connectionist schemes for structure preserving dimensionality reduction, IEEE Trans. Neural Networks 9 (6) (1998).

[44] R.R. Yager, D.P. Filev, Generation of fuzzy rules by mountain clustering, J. Int. Fuzzy Systems 2 (1994) 209–219.

[45] L.A. Zadeh, Fuzzy sets, Inform. and Control 8 (1965) 338–353.

[46] L.A. Zadeh, Fuzzy logic and soft computing: issues, contention and perspection, Proc. 3rd Internat. Conf. on Fuzzy Logic, Neural Nets and Soft Computing, Iizuka, Japan, 1994, pp. 1–2.