Reduced Polynomial Neural Swarm Net for Classification Task in Data Mining

B. B. Misra, S. Dehuri, P. K.Dash, and G. Panda

***Abstract*—In this paper, we proposed a reduced polynomial neural swarm net (RPNSN) for the task of classification. Classification task is one of the most studied tasks of data mining. In solving classification task of data mining, the classical algorithm such as Polynomial Neural Network (PNN) takes large computation time because the network grows over the training period (i.e. the partial descriptions (PDs) in each layer grows in successive generations). Unlike PNN our proposed network needs to generate the partial description for a single layer. Particle swarm optimization (PSO) technique is used to select a relevant set of PDs as well as features, which are then fed to the output layer of our proposed net which contain only one neuron. The selection mechanism used here is a kind of wrapper approach. Performance of this model is compared with the results obtained from PNN. Simulation result shows that the performance of RPNSN is encouraging for harnessing its power in data mining area and also better in terms of processing time than the PNN model.**

I. INTRODUCTION

Data classification [1,26,28] is a core issue in data mining, pattern recognition, and forecasting. The goal of classification is to assign a new object to a class from a given set of predefined classes based on the attribute values of this object. Furthermore, classification is based on some discovered model, which forms an important piece of knowledge about the application domain. There has been wide range of machine learning and statistical methods for classification task. One of the popular and widely used techniques is feed-forward neural network (FNN) [2]. Although such FNNs can learn to classify wide range of problem domain well, the classification model cannot be comprehensible due to large number of synaptic connections. To achieve high classification accuracy in FNN framework, one has to provide a well defined structure of FNNs, such as, the number of input nodes, hidden and output neurons, and assume a proper set of relevant features. Trial and error methodology is used to arrive at such kind of structures, which is computationally expensive. Similarly other methods like rule extraction and decision tree, which provide comprehensible rule, are based on the trade-off between the complexity and the classification accuracy of the model.

B. B. Misra is with the College of Engineering Bhubaneswar, Orissa, India, (phone: 0919437400307; fax: ;e-mail: misra_bijan@yahoo.co.in)

S. Dehuri is with Fakir Mohan University, Balasore, Orissa, India, (e-mail: satchi.lapa@gmail.com)

G. Panda, is with the National Institute of Technology, Rourkela, Orissa, India, (e-mail: ganapatipanda@gmail.com).

P. K. Dash, is with the College of Engineering Bhubaneswar, Orissa, India, (e-mail: director@ceb.org).

To alleviate the shortcomings of FNNs, Polynomial Neural Network (PNN) based on Group Method of Data Handling (GMDH) approach suggested by Ivakhenko [3, 4, and 27] can be used for classification purposes. The approach is based on evolutionary strategy, where PNN generates populations or layers of neurons/simulated units/partial descriptions (PDs) and then trains and selects those neurons, which provide the best classification. During learning the PNN model grows the new population of neurons and the number of layers & the complexity of the network increases [24, 25] while a predefined criterion is met. Such models can be comprehensively described by a set of short-term polynomials thereby developing a PNN classifier. Coefficients of PNN can be estimated by least square fitting.

The network architecture grows depending on the number of input features, PNN model selected, number of layer required, and the number of PD's preserved in each layer. In turn the architecture becomes very complex, requires huge memory and computation time. In our approach, we take only one layer of PNN model. We select an optimal set from the PD's generated in the first layer along with the input features using PSO technique. This optimal set of features is fed to a single Perceptron like model of ANN. The weights of the single perceptron like model are also optimized by PSO.

The rest of the paper is organized as follows. Section II describes the basics of PNN. In Section III, PSO is discussed. The analysis and design of RPNSN architecture is given in Section IV. In Section V, a simulation result of the model is presented. Section VI summarizes this paper.

II. BASICS OF PNN

A. PNN Architecture

The GMDH belongs to the category of inductive self-organization data driven approaches. It requires small data samples and is able to optimize models' structure objectively. Relationship between input–output variables can be approximated by Voters functional series, the discrete form of which is Kolmogorov-Gabor Polynomial[4]

$$y = C_0 + \sum_{k1} C_{k1} x_{k1} + \sum_{k1k2} C_{k1k2} x_{k1} x_{k2} + \sum_{k1k2k3} C_{k1k2k3} x_{k1} x_{k2} x_{k3} + \ldots . \quad (1)$$

where C_k denotes the coefficients or weights of the Kolmogorov-Gabor polynomial & **x** vector is the input variables. This polynomial can approximate any stationary random sequence of observations and it can be solved by either adaptive methods or by Gaussian equations. This

978-1-4244-1823-7/08/$25.00©2008 IEEE

polynomial is not computationally suitable if the number of input variables increase and there are missing observations in input dataset. Also it takes more computation time to solve all necessary normal equations when the input variables are large.

A new algorithm called GMDH is developed by Ivakhnenko [4, 6] which is a form of Kolmogorov-Gabor polynomial. He proved that a second order polynomial i.e

$$y = a_0 + a_1 x_i + a_2 x_j + a_3 x_i x_j + a_4 x_i^2 + a_5 x_j^2 , \quad (2)$$

which takes only two input variables at a time and can reconstruct the complete Kolmogorov-Gabor polynomial through an iterative procedure. The GMDH method belongs to the category of heuristic self-organization methods, where the approaches like black-box concepts, connectionism and induction concepts can be applied [5]. The black-box method is a principal approach to analyze systems on the basis of input-output samples. And the method of connection and induction can be thought of as representation of complex functions through network of elementary functions. Thus the GMDH algorithm has the ability to trace all input-output relationship through an entire system that is too complex. The GMDH-type Polynomial Neural Networks are multilayered model consisting of the neurons/active units /Partial Descriptions(PDs) whose transfer function is a short-term polynomial described in equation (2). At the first layer *L*=1, an algorithm, using all possible combinations by two from *m* inputs variables, generates the first population of PDs. Total number of PDs in first layer is $n = m(m-1)/2$. The outputs of each PDs in layer *L*=1 is computed by applying the equation (2). Let the outputs of first layer are denoted as $y_1^1, y_2^1, \ldots\ldots, y_n^1$ .The vector of coefficients of the PDs are determined by least square estimation approach.

The architecture of a PNN with four input features is shown in Fig. 1

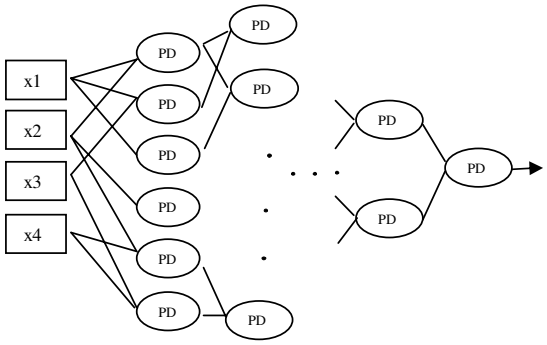

Fig. 1: Basic PNN model

The details of PNN model development and least square estimation technique is explained below.

Let the input and output data for training is represented in the following manner

$$\begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1m} & y_1 \\ x_{21} & x_{22} & \ldots & x_{2m} & y_2 \\ . & . & . & . & . \\ x_{n1} & x_{n2} & \ldots & x_{nm} & y_n \end{bmatrix}$$

In general, it is expressed as

$(X_i, y_i) = (x_{1i}, x_{2i}, \ldots, x_{mi}, y_i)$, where *i =1, 2, 3, … ,n.*

The input and output relationship of the above data by PNN algorithm can be described in the following manner:

$y = f(x_1, x_2, x_3, \ldots, x_m)$,

where m is the number of features in the dataset.

Number of PDs, K in each layer depends on the number of input features M as below

$$K = \binom{M}{2} = M(M-1)/2$$

The input index of features (p, q) to each PD, may be generated using the following algorithm

1. Let layers is l.
2. Let k=1,
3. FOR i =1 to m-1
4. FOR j = i+1 to m
5. Then PD_k^1 will receive input from the features
6. p=i; & q=j;
7. k=k+1;
8. END FOR
9 END FOR

Let consider the equations for the first PD of layer1, which receives input from feature 1 and 2.

$$d_1 = y_1 - (c_{11} + c_{12}x_{11} + c_{13}x_{12} + c_{14}x_{11}x_{12} + c_{15}x_{11}^2 + c_{16}x_{12}^2$$

$$d_2 = y_2 - (c_{11} + c_{12}x_{21} + c_{13}x_{22} + c_{14}x_{21}x_{22} + c_{15}x_{21}^2 + c_{16}x_{22}^2$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$d_n = y_n - (c_{11} + c_{12}x_{n1} + c_{13}x_{n2} + c_{14}x_{n1}x_{n2} + c_{15}x_{n1}^2 + c_{16}x_{n2}^2$$

This equation in general may be written as

$$d_i = y_i - (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2$$

where (i) i=1, 2, ..., n.
(ii) j=1, 2, ..., k
(iii) k=m(m-1)/2

The equations for the least square are

$$\Pi = d_1^2 + d_2^2 + \ldots + d_n^2$$
$$= \sum_{i=1}^{n} d_i^2$$
$$= \sum_{i=1}^{n} \left[y_i - (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2) \right]^2$$

To minimize the error, we get the first derivatives of Π in terms of all the unknown variables (i.e. the coefficients).

$$\begin{bmatrix} \frac{\partial \Pi}{\partial c_{j1}} = 2\sum_{i=1}^{n}\left[y_i - (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2\right] = 0 \\ \frac{\partial \Pi}{\partial c_{j2}} = 2\sum_{i=1}^{n}x_{ip}\left[y_i - (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2\right] = 0 \\ \frac{\partial \Pi}{\partial c_{j3}} = 2\sum_{i=1}^{n}x_{iq}\left[y_i - (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2\right] = 0 \\ \frac{\partial \Pi}{\partial c_{j4}} = 2\sum_{i=1}^{n}x_{ip}x_{iq}\left[y_i - (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2\right] = 0 \\ \frac{\partial \Pi}{\partial c_{j5}} = 2\sum_{i=1}^{n}x_{ip}^2\left[y_i - (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2\right] = 0 \\ \frac{\partial \Pi}{\partial c_{j6}} = 2\sum_{i=1}^{n}x_{iq}^2\left[y_i - (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2\right] = 0 \end{bmatrix}$$

On expanding the above equations, we get

$$\begin{bmatrix} \sum_{i=1}^{n}y_i = c_{j1}\sum_{i=1}^{n}1 + c_{j2}\sum_{i=1}^{n}x_{ip} + c_{j3}\sum_{i=1}^{n}x_{iq} + c_{j4}\sum_{i=1}^{n}x_{ip}x_{iq} + c_{j5}\sum_{i=1}^{n}x_{ip}^2 + c_{j6}\sum_{i=1}^{n}x_{iq}^2 \\ \sum_{i=1}^{n}x_{ip}y_i = c_{j1}\sum_{i=1}^{n}x_{ip} + c_{j2}\sum_{i=1}^{n}x_{ip}^2 + c_{j3}\sum_{i=1}^{n}x_{ip}x_{iq} + c_{j4}\sum_{i=1}^{n}x_{ip}^2x_{iq} + c_{j5}\sum_{i=1}^{n}x_{ip}^3 + c_{j6}\sum_{i=1}^{n}x_{ip}x_{iq}^2 \\ \sum_{i=1}^{n}x_{iq}y_i = c_{j1}\sum_{i=1}^{n}x_{iq} + c_{j2}\sum_{i=1}^{n}x_{ip}x_{iq} + c_{j3}\sum_{i=1}^{n}x_{iq}^2 + c_{j4}\sum_{i=1}^{n}x_{ip}x_{iq}^2 + c_{j5}\sum_{i=1}^{n}x_{ip}^2x_{iq} + c_{j6}\sum_{i=1}^{n}x_{iq}^3 \\ \sum_{i=1}^{n}x_{ip}x_{iq}y_i = c_{j1}\sum_{i=1}^{n}x_{ip}x_{iq} + c_{j2}\sum_{i=1}^{n}x_{ip}^2x_{iq} + c_{j3}\sum_{i=1}^{n}x_{ip}x_{iq}^2 + c_{j4}\sum_{i=1}^{n}x_{ip}^2x_{iq}^2 + c_{j5}\sum_{i=1}^{n}x_{ip}^3x_{iq} + c_{j6}\sum_{i=1}^{n}x_{ip}x_{iq}^3 \\ \sum_{i=1}^{n}x_{ip}^2y_i = c_{j1}\sum_{i=1}^{n}x_{ip}^2 + c_{j2}\sum_{i=1}^{n}x_{ip}^3 + c_{j3}\sum_{i=1}^{n}x_{ip}^2x_{iq} + c_{j4}\sum_{i=1}^{n}x_{ip}^3x_{iq} + c_{j5}\sum_{i=1}^{n}x_{ip}^4 + c_{j6}\sum_{i=1}^{n}x_{ip}^2x_{iq}^2 \\ \sum_{i=1}^{n}x_{iq}^2y_i = c_{j1}\sum_{i=1}^{n}x_{iq}^2 + c_{j2}\sum_{i=1}^{n}x_{ip}x_{iq}^2 + c_{j3}\sum_{i=1}^{n}x_{iq}^3 + c_{j4}\sum_{i=1}^{n}x_{ip}x_{iq}^3 + c_{j5}\sum_{i=1}^{n}x_{ip}^2x_{iq}^2 + c_{j6}\sum_{i=1}^{n}x_{iq}^4 \end{bmatrix}$$

We know that,

$XA = Y$

$X^T XA = X^T Y$

$A = (X^T X)^{-1} X^T Y$

Here,

$$X = \begin{bmatrix} n & \sum_{i=1}^{n}x_{ip} & \sum_{i=1}^{n}x_{iq} & \sum_{i=1}^{n}x_{ip}x_{iq} & \sum_{i=1}^{n}x_{ip}^2 & \sum_{i=1}^{n}x_{iq}^2 \\ \sum_{i=1}^{n}x_{ip} & \sum_{i=1}^{n}x_{ip}^2 & \sum_{i=1}^{n}x_{ip}x_{iq} & \sum_{i=1}^{n}x_{ip}^2x_{iq} & \sum_{i=1}^{n}x_{ip}^3 & \sum_{i=1}^{n}x_{ip}x_{iq}^2 \\ \sum_{i=1}^{n}x_{iq} & \sum_{i=1}^{n}x_{ip}x_{iq} & \sum_{i=1}^{n}x_{iq}^2 & \sum_{i=1}^{n}x_{ip}x_{iq}^2 & \sum_{i=1}^{n}x_{ip}^2x_{iq} & \sum_{i=1}^{n}x_{iq}^3 \\ \sum_{i=1}^{n}x_{ip}x_{iq} & \sum_{i=1}^{n}x_{ip}^2x_{iq} & \sum_{i=1}^{n}x_{ip}x_{iq}^2 & \sum_{i=1}^{n}x_{ip}^2x_{iq}^2 & \sum_{i=1}^{n}x_{ip}^3x_{iq} & \sum_{i=1}^{n}x^{ip}x_{iq}^3 \\ \sum_{i=1}^{n}x_{ip}^2 & \sum_{i=1}^{n}x_{ip}^3 & \sum_{i=1}^{n}x_{ip}^2x_{iq} & \sum_{i=1}^{n}x_{ip}^3x_{iq} & \sum_{i=1}^{n}x_{ip}^4 & \sum_{i=1}^{n}x_{ip}^2x_{iq}^2 \\ \sum_{i=1}^{n}x_{iq}^2 & \sum_{i=1}^{n}x_{ip}x_{iq}^2 & \sum_{i=1}^{n}x_{iq}^3 & \sum_{i=1}^{n}x_{ip}x_{iq}^3 & \sum_{i=1}^{n}x_{ip}^2x_{iq}^2 & \sum_{i=1}^{n}x_{iq}^4 \end{bmatrix}$$

$$A = \begin{bmatrix} c_{j1} \\ c_{j2} \\ c_{j3} \\ c_{j4} \\ c_{j5} \\ c_{j6} \end{bmatrix}, \text{ and } \quad Y = \begin{bmatrix} \sum_{i=1}^{n}1y_i \\ \sum_{i=1}^{n}x_{ip}y_i \\ \sum_{i=1}^{n}x_{iq}y_i \\ \sum_{i=1}^{n}x_{ip}x_{iq}y_i \\ \sum_{i=1}^{n}x_{ip}^2y_i \\ \sum_{i=1}^{n}x_{iq}^2y_i \end{bmatrix}$$

After obtaining the values of the coefficients with the testing dataset, we estimate the target

$$\hat{y}_i = (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x_{iq}^2)$$

If the error level is not up to our desired value, we construct next layer of PNN by taking the output of the previous layer

$$z_j = (c_{j1} + c_{j2}x_{ip} + c_{j3}x_{iq} + c_{j4}x_{ip}x_{iq} + c_{j5}x_{ip}^2 + c_{j6}x)$$

and apply the same procedure.

$$d_i = y_i - (c_{j1} + c_{j2}z_{ip} + c_{j3}z_{iq} + c_{j4}z_{ip}z_{iq} + c_{j5}z_{ip}^2 + c_{j6}z_{iq}^2$$

This process is repeated till error decreases. Overall framework of the design procedure of the GMDH-type PNN comes as a sequence of the following steps.

Step 1: Determine system's input variables

Step 2: Form training and testing data

Step 3: Choose a structure of the PNN

Step 4: Determine the number of input variables and the order of the polynomial forming a partial description (PD) of data.

Step 5: Estimate the coefficients of the PD

Step 6: Select PDs with the best classification accuracy

Step 7: Check the stopping criterion

Step 8: Determine new input variables for the next layer

The layers of PNN models are grown as per the algorithm described above. The residual error between the estimated output and the actual output is calculated at each layer. If the error level is within the tolerable limit then the growth of the model is stopped and the final model is derived taking into account only those PDs that contribute to obtain the best result. Otherwise the next layer is grown. It is observed that the error level decreases rapidly at the first layers of PNN network and relatively slower near to optimal number of layers, and further increasing the number of layers causes increasing the value of error level because of over-fitting [8,9,10]. Thus in our simulation the number of layers in the model increases one-by-one until the stopping rule i.e. the tolerable error level is met at the layer r. Subsequently we take a desired PNN model of nearly optimal complexity from (*r-1*)th layer. Hence we preserve only those PDs that contribute to the better result. From the simulation it is seen that the output of best two PDs of previous layer not necessarily yields the best result in the next layer. Hence, PDs that give better result in a layer are preserved for building the next layer.

The PNN model is tested with the test data set. We have adopted the 2-fold cross-validation strategy to test. Each set of the datasets is used for training and testing. Two third of a set is taken for training a model and one third is used for validation of the model. Then both the train and the test data sets are exposed to the model for classification. The average of the percentage of correct classification for test data set using PNN model are considered for comparison.

III. Particle Swarm Optimization

A. PSO basics

The implicit rules adhered to by the members of bird flocks and fish schools, that enable them to move synchronized, without colliding, resulting in an amazing choreography, was studied and simulated by several scientists [11,12]. In simulations, the movement of the flock was an outcome of the individuals' (birds, fishes etc.) efforts to maintain an optimum distance from their neighboring individuals [13].

The social behavior of animals, and in some cases of humans, is governed by similar rules [14]. However, human social behavior is more complex than a flock's movement. Besides physical motion, humans adjust their beliefs, moving, thus, in a belief space. Although two persons cannot occupy the same space of their physical environment, they can have the same beliefs, occupying the same position in the belief space, without collision. This abstractness in human social behavior is intriguing and has constituted the motivation for developing simulations of it. There is a general belief, and numerous examples coming from nature enforce the view, that social sharing of information among the individuals of a population may provide an evolutionary advantage. This was the core idea behind the development of PSO [13].

B. PSO Algorithm

PSO's precursor was a simulator of social behavior that was used to visualize the movement of a birds' flock. Several versions of the simulation model were developed, incorporating concepts such as nearest-neighbor velocity matching and acceleration by distance [13,15]. When it was realized that the simulation could be used as an optimizer, several parameters were omitted, through a trial and error process, resulting in the first simple version of PSO [13].

PSO is similar to EC techniques in that, a population of potential solutions to the problem under consideration is used to probe the search space. However, in PSO, each individual of the population has an *adaptable velocity* (position change), according to which it moves in the search space. Moreover, each individual has a *memory*, remembering the best position of the search space it has ever visited [13]. Thus, its movement is an aggregated acceleration towards its best previously visited position and towards the best individual of a topological neighborhood. Since the ''acceleration'' term was mainly used for particle systems in Particle Physics [16], the pioneers of this technique decided to use the term *particle* for each individual, and the name *swarm* for the population, thus, coming up with the name *Particle Swarm* for their algorithm [15].

Two variants of the PSO algorithm were developed, one with a global neighborhood, and the other with a local neighborhood. According to the global variant, each particle moves towards its best previous position and towards the best particle in the whole swarm. On the other hand, according to the local variant, each particle moves towards its best previous position and towards the best particle in its restricted neighborhood [13]. In the following paragraphs, the global variant is exposed.

Suppose that the search space is D-dimensional, then the i^{th} particle of the swarm can be represented by a D-dimensional vector, $x_i = (x_{i1}, x_{i2}, ..., x_{iD})$. The *velocity* (position change) of this particle, can be represented by another D-dimensional vector $v_i = (v_{i1}, v_{i2}, ..., v_{iD})$. The best previously visited position of the i^{th} particle is denoted as $p_i = (p_{i1}, p_{i2}, ..., p_{iD})$. Defining g as the index of the best particle in the swarm (i.e., the g^{th} particle is the best), and let the superscripts denote the iteration number, then the swarm is manipulated according to the following two equations:

$$v_{id}^{n+1} = v_{id}^{n} + cr_1^n \left(p_{id}^n - x_{id}^n\right) + cr_2^n \left(p_{gd}^n - x_{id}^n\right) \quad (3)$$

$$x_{id}^{n+1} = x_{id}^{n} + v_{id}^{n+1} \quad (4)$$

where $d = 1, 2, \ldots, D$; $i = 1, 2, \ldots, N$, and N is the size of the swarm; c is a positive constant, called *acceleration constant*; $r1$, $r2$ are random numbers, uniformly distributed in [0, 1]; and $n = 1, 2, \ldots$, determines the iteration number.

Equations (3) and (4) define the initial version of the PSO algorithm. Since there was no actual mechanism for controlling the velocity of a particle, it was necessary to impose a maximum value *Vmax* on it. If the velocity exceeded this threshold, it was set equal to *Vmax*. This parameter proved to be crucial, because large values could result in particles moving past good solutions, while small values could result in insufficient exploration of the search space. This lack of a control mechanism for the velocity resulted in low efficiency for PSO, compared to EC techniques [17]. Specifically, PSO located the area of the optimum faster than EC techniques, but once in the region of the optimum, it could not adjust its velocity step size to continue the search at a finer grain.

The aforementioned problem was addressed by incorporating a weight parameter for the previous velocity of the particle. Thus, in the latest versions of the PSO, Equations (3) and (4) are changed to the following ones [18,19,20]:

$$v_{id}^{n+1} = \chi\left(wv_{id}^{n} + c_1 r_1^n \left(p_{id}^n - x_{id}^n\right) + c_2 r_2^n \left(p_{gd}^n - x_{id}^n\right)\right) \quad (5)$$

$$x_{id}^{n+1} = x_{id}^{n} + v_{id}^{n+1} \quad (6)$$

where w is called *inertia weight*; $c1$, $c2$ are two positive constants, called *cognitive* and *social* parameter respectively; and χ is a *constriction factor*, which is used, alternatively to w to limit velocity.

In the local variant of PSO, each particle moves towards the best particle of its neighborhood. For example, if the size of the neighborhood is 2, then the i^{th} particle moves towards the best particle among the $(i-1)^{th}$, the $(i+1)^{th}$ and itself.

The PSO method appears to adhere to the five basic principles of swarm intelligence, as defined by [13]:
(a) *Proximity*, i.e., the swarm must be able to perform simple space and time computations;
(b) *Quality*, i.e., the swarm should be able to respond to quality factors in the environment;
(c) *Diverse response*, i.e., the swarm should not commit its activities along excessively narrow channels;
(d) *Stability*, i.e., the swarm should not change its behavior every time the environment alters; and finally
(e) *Adaptability*, i.e., the swarm must be able to change its behavior, when the computational cost is not prohibitive.

Indeed, the swarm in PSO performs space calculations for several time steps. It responds to the quality factors implied by each particle's best position and the best particle in the swarm, allocating the responses in a way that ensures diversity. Moreover, the swarm alters its behavior (state) only when the best particle in the swarm (or in the neighborhood, in the local variant of PSO) changes, thus, it is both adaptive and stable [13].

C. The parameters of PSO

The role of the *inertia weight w*, in Equation (5), is considered critical for the PSO's convergence behavior. The inertia weight is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter *w* regulates the trade-off between the global (wide-ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e., fine-tuning the current search area. A suitable value for the inertia weight *w* usually provides balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. Initially, the inertia weight was constant. However, experimental results indicated that it is better to initially set the inertia to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions [19, 20]. Thus, an initial value around 1.2 and a gradual decline towards 0 can be considered as a good choice for *w*.

The parameters $c1$ and $c2$, in Equation (5), are not critical for PSO's convergence. However, proper fine-tuning may result in faster convergence and alleviation of local minima. An extended study of the acceleration parameter in the first version of PSO, is given in [22]. As default values, $c1 = c2 = 2$ were proposed, but experimental results indicate that $c1 = c2 = 0.5$ might provide even better results. Recent work reports that it might be even better to choose a larger cognitive parameter, $c1$, than a social parameter, $c2$, but with $c1 + c2 <= \rightsquigarrow 4$ [23].
The parameters $r1$ and $r2$ are used to maintain the diversity of the population, and they are uniformly distributed in the range [0, 1]. The constriction factor χ controls on the magnitude of the velocities, in a way similar to the *Vmax* parameter, resulting in a variant of PSO, different from the one with the inertia weight.

IV. RPNSN Architecture

While simulating the PNN model it is observed that the number of partial descriptions generated in each layer grows very fast. As a result lot of time is consumed in generating the PDs. In general the PDs giving poor performance are rejected. But still then a substantial number of PDs needs to be preserved to get better result in subsequent layers. It is observed that always PDs giving best result do not combine to yield improvised result. Very often one PD giving better result combined with other PD giving inferior result to improvise its performance in subsequent layer. Therefore it is always essential to preserve substantial number of PDs in a hope of getting better result in subsequent layers. In turn huge amount of memory and running time is needed for the process of generation of a model for a dataset [21].

Each PD tries to approximate the input output relation of the dataset. In the suggested model, we have developed PDs for one layer. Along with the output of the first layer, we have considered the original features of the dataset. The PSO technique is used to select the optimal number of input to the output layer.

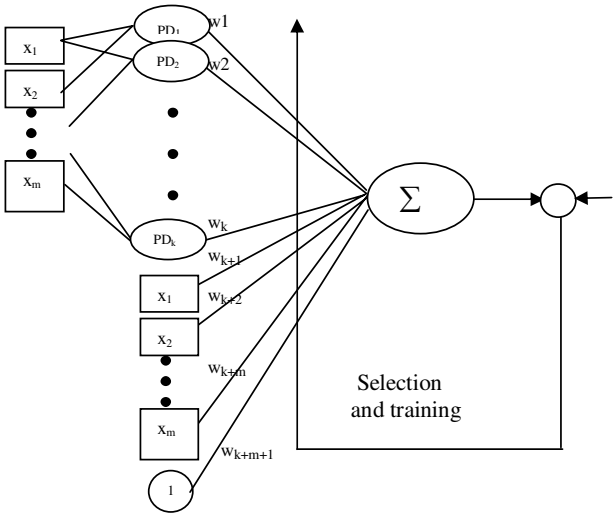

Fig. .2: RPNSN model

In the model, m represents the number of features in the dataset and k represents number of PDs generated out of m features. One bias is included to the net at this level. There are m+k+1 number of weights to be optimized. PSO technique is again used to train these weights.

V. Simulations and Results

The performance of different models is evaluated using the benchmark classification databases. Out of these, the most frequently used in the area of neural networks and of neuro-fuzzy systems are IRIS, WINE, PIMA, BUPA Liver Disorders. All these databases are taken from the UCI machine repository [7]. Table I presents the summary of the main features of the datasets used for our experimental studies.

TABLE I
DESCRIPTION OF DATASESETS USED

Dataset	Patterns	Attributes	Classes	Patterns in Class1	Patterns in Class2	Patterns in Class3
IRIS	150	4	3	50	50	50
WINE	178	13	3	59	71	48
PIMA	768	8	2	268	500	-
BUPA	345	6	2	145	200	-

The dataset is divided into two parts. The division of datasets and its class distribution is shown in Table II.

TABLE II
DIVISION OF DATASET AND ITS PATTERN DISTRIBUTION

		Patterns	Patterns in Class1	Patterns in Class2	Patterns in Class3
IRIS	Set1	75	25	25	25
	Set2	75	25	25	25
WINE	Set1	89	29	36	24
	Set2	89	30	35	24
PIMA	Set1	384	134	250	-
	Set2	384	134	250	-
BUPA	Set1	172	72	100	-
	Set2	173	73	100	-

One part is used for building the model and other part is used for testing the model. The protocol for parameters used for our simulation studies is given in Table III.

TABLE III
PARAMETERS CONSIDERED FOR SIMULATION OF RPNSN MODEL USING PSO

Parameters	Values
Population Size	20
Maximum Iterations	100
Inertia Weight	0.729844
Cognitive Parameter	1.49445
Social Parameter	1.49445
Constriction Factor	1.0

The average percentage of correct classification obtained for the test sets are provided in Table IV for the purpose of comparison.

TABLE IV
COMPARISON OF AVERAGE PERCENTAGE OF CORRECT CLASSIFICATION OF TEST SETS WITH PNN AND RPNSN

Data set	PNN	RPNSN
IRIS	98.68	99.3333
WINE	94.872	99.4382
PIMA	65.1042	76.823
BUPA	70.196	73.9061

In Table V the percentage of PDs used in our model is given, which is also very crucial aspect to obtain an optimum model.

TABLE V
PERCENTAGE OF PDS USED BY THE RPNSN TO OBTAIN THE OPTIMUM MODEL

Data set	Total number of possible PDs.	% of PDs used in optimized model	
		Set1	Set2
IRIS	6	33.33	50.00
WINE	78	32.05	19.23
PIMA	28	21.43	32.14
BUPA	15	33.33	26.67

Table VI shows the processing time of both PNN and RPNSN. From the Table it shows that the time required for our proposed model is comparatively very less than PNN.

TABLE VI
COMPARISON OF PROCESSING TIME PERFORMANCE OF PNN WITH RPNSN (IN SECONDS)

Datasets		PNN	RPNSN
IRIS	Set1	129	0.6563
	Set2	125	0.6719
WINE	Set1	225	4.5625
	Set2	224	4.7031
PIMA	Set1	783	8.1406
	Set2	793	8.4531
BUPA	Set1	352	2.2969
	Set2	353	2.4219

The mathematical model obtained by our PNN Model for Iris dataset is presented for an example purpose.

$$PD_2^1 = [-0.96751, -0.25663, 0.65004, -0.13994, 0.044319, 0.10067] * \text{poly}(x_1, x_3),$$

$$PD_3^1 = [1.633, -1.0407, 1.4825, -0.051333, 0.09108, -0.067678] * \text{poly}(x_1, x_4);$$

$$PD_1^1 = [-1.7572, 1.4898, -2.4317, 0.3308, -0.15092, -0.018686] * \text{poly}(x_1, x_2);$$

$$PD_{16}^2 = [2.0965, 1.4134, -0.8428, 0.046201, -0.57289, 0.079276] * \text{poly}(PD_2^1, x_3);$$

$$PD_{28}^2 = [2.2683, 1.7305, -1.2497, -0.22651, 0.15639, 0.15237] * \text{poly}(PD_3^1, x_2);$$

$$PD_{38}^2 = [-1.8284, -0.58075, 0.42213, 0.081458, -0.25631, 0.015491] * \text{poly}(PD_1^1, x_3);$$

$$PD_{496}^3 = [-0.054529, 0.54087, 0.45923, -1.1876, 0.63090, 0.60812] * \text{poly}(PD_{16}^2, PD_{28}^2);$$

$$PD_{557}^3 = [-0.043133, 0.59432, 0.41454, -0.75286, 0.39684, 0.3952] * \text{poly}(PD_{28}^2, PD_{38}^2);$$

$$y = [0.0057053, 1.158, -0.15925, 5.3642, -2.208, -3.156] * \text{poly}(PD_{496}^3, PD_{557}^3);$$

where

1. *Function ploy* (a_1, a_2)
{
return $[1, a_1, a_2, a_1 * a_2, a_1.^2, a_2.^2]^T$;

}

2. PD_j^i is the output of layer i and j^{th} partial description.

3. y is the output of the PNN model and x is the input features.

The architecture of the PNN model generated is given at Fig.3.

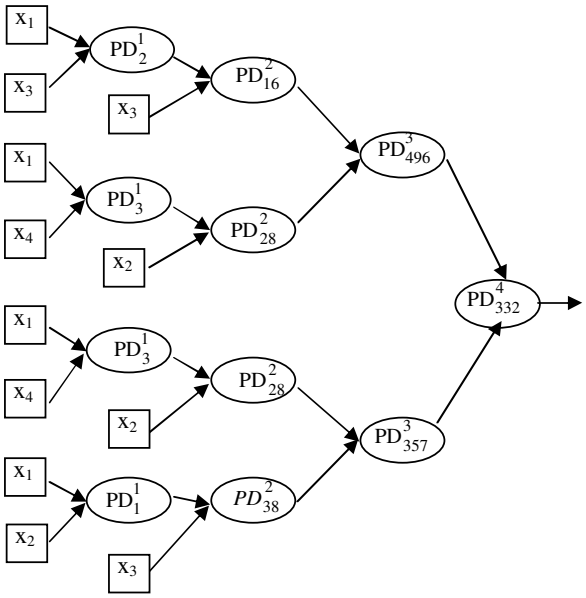

Fig. 3 PNN model for Iris dataset

The mathematical model obtained by our RPNSN Model for Iris dataset presented below.

$$PD_1^1 = [12.265 \quad -27.379, \quad 2.7124, \quad -5.0129, \quad 16.636, \quad 2.6568] \quad * \text{poly}(x_1, x_2),$$

$$PD_5^1 = [3.4633, \quad -0.61437, \quad -4.7075, \quad 3.5712, \quad 0.32404, \quad -0.37042] \quad * \text{poly}(x_2, x_4),$$

$$y = [0.02099x_4 + 0.62292PD_1^1 + 0.33345PD_5^1 + 0.10778];$$

The architecture of the RPNSN model generated is given at Fig.4

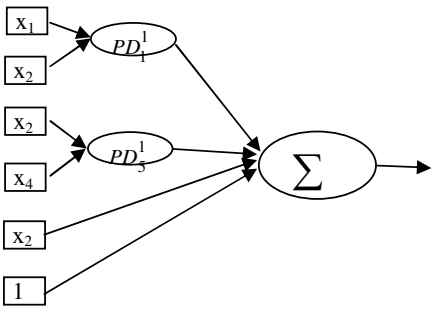

Fig. 4 RPNSN model for iris data set

After generating the RPNSN model, the confusion matrix is obtained for the entire dataset. The confusion matrices for the class 2 datasets are shown in Table VII and the confusion matrix for the class 3 datasets are shown in Table VIII.

TABLE VII
CONFUSION MATRIX FOR TWO CLASS DATASETS

			Predicted	
	Actual		c1	c2
BUPA		c1	85	60
		c2	30	170
PIMA		c1	134	134
		c2	44	456

TABLE VIII
CONFUSION MATRIX FOR THREE CLASS DATASETS

			Predicted		
			c1	c2	c3
IRIS	A	c1	50	0	0
	c	c2	1	49	0
	t	c3	0	0	50
WINE	u	c1	59	0	0
	a	c2	0	70	1
	l	c3	0	0	48

During training of the RPNSN model, different error curves are obtained for different simulations. Fig. 5, 6, 7 and 8 shows the error curves obtained from our model for Iris, Wine, Bupa, Pima datasets respectively. Similarly Fig. 9, 10, 11, and 12 shows the error curves obtained from PNN for Iris, Bupa, Pima and Wine and datasets respectively.

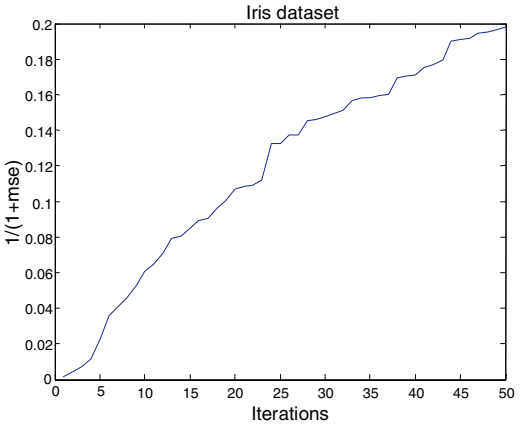

Figure 5 Error curve for Iris dataset

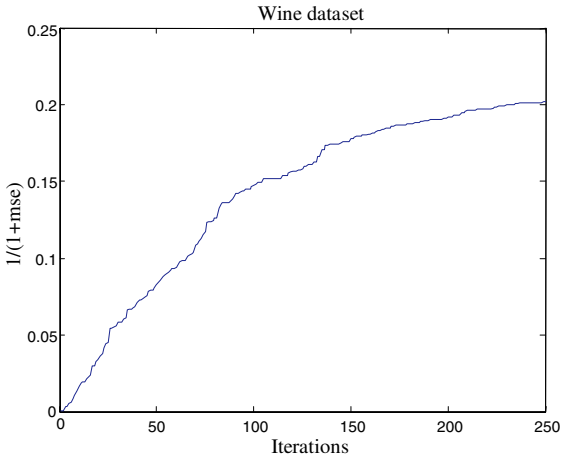

Figure 6 Error curve for wine dataset

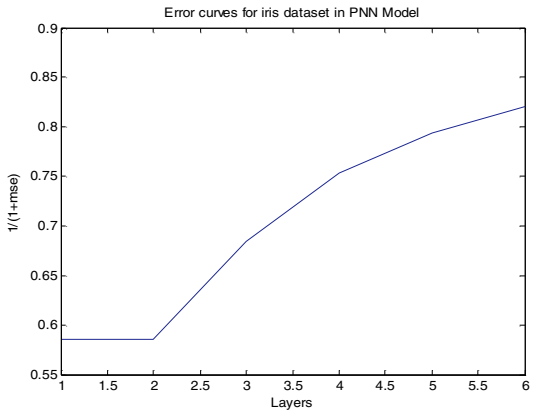

Fig. 9. Error curve for IRIS data

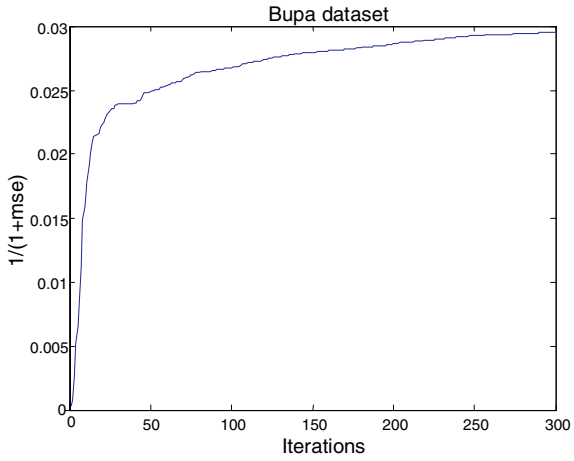

Figure 7 Error curve for bupa dataset

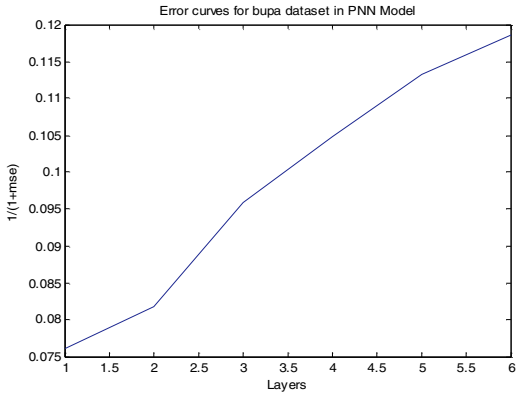

Fig. 10. Error curve for Bupa data

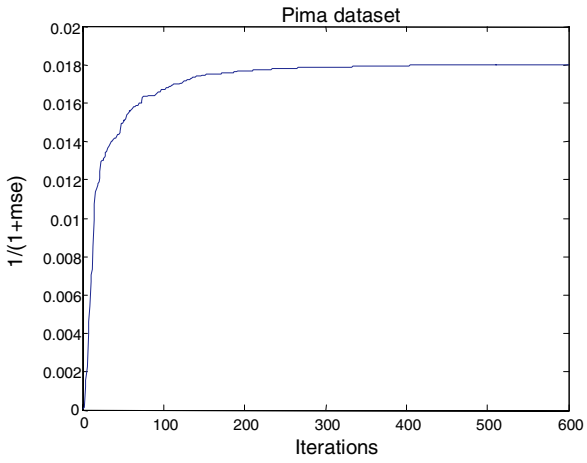

Figure 8 Error curve for pima dataset

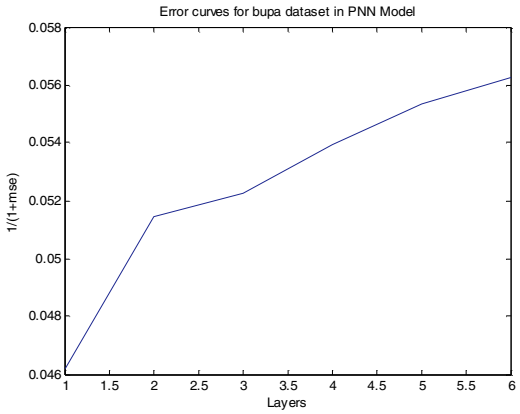

Fig. 11. Error curve for Pima data

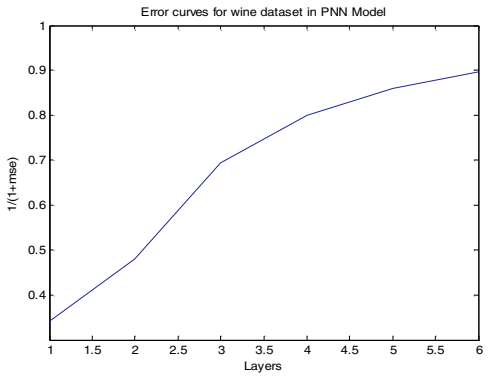

Fig. 12. Error curve for Wine data

VI. CONCLUSION

In this paper, we have proposed RPNSN for the classification task of data mining. The RPNSN model generates PDs for one layer of the basic PNN model. PSO selects the optimal set of PDs and input features, which are fed to the single layer feed forward artificial neural network. The network is also trained using PSO technique. The experimental studies demonstrated that the RPNSN model performs the pattern classification task quite well. In all the cases, the results obtained with the RPNSN model proved to be better than the PNN results. The performance of the RPNSN models is better in terms of processing time, which is also treated as one of the crucial aspect in data mining community.

REFERENCES

[1] T. M. Mitchel, "Machine learning," McGraw Hill, 1997.

[2] Y. H. Pao, "Adaptive pattern recognition neural networks," Addison Wesley, MA, 1989.

[3] A.G. Ivakhnenko, "Polynomial theory of complex systems," IEEE Trans. Syst., Man Cybern-I, pp. 364–378, 1971.

[4] A. G. Ivakhnenko, and H. R. Madala, "Inductive learning algorithm for complex systems modelling," Boca raton:CRC Inc,1994.

[5] S. J. Farlow, "The GMDH algorithm," In:Farlow S.J (eds.). Self-organizating methods in modelling:GMDH type algorithm. New York:Marcel Dekker, pp.1-24, 1984.

[6] A. J. Muller, F. Lemke, and A. G. Ivakhnenko, "GMDH algorithms for complex systems modeling," Math and Computer Modeling of Dynamical Systems , vol. 4, pp.275-315, 1998.

[7] C. L. Blake and C.J. Merz, "UCI Repository of machine learning databases," http://www.ics.uci.edu/ ~mlearn/MLRepository.

[8] J. A. Muller, and F. Lemke, "Self-organizing data mining extracting knowledge from data," Trafford Publishing, Canada British Columbia, 2003.

[9] V. Schetinin, and J. Schult, "The combined technique for detection of artifacts in clinical electroencephalograms of sleeping newborn," IEEE Trans. on Information Technologies in Biomedicine, vol. 8, no.1, pp. 28-35, 2004.

[10] N. L. Nikolaev, and H. Iba, "Automated discovery of polynomials by inductive genetic programming," In: Zutkow J, Ranch J (eds.) Principles of Data Mining and Knowledge Discovery (PKDD'99). Springer, Berlin, pp.456-462, 1999.

[11] F. Heppner and U. Grenander, "A stochastic nonlinear model for coordinate bird flocks," In: Krasner S (eds.) The Ubiquity of Chaos. AAAS Publications, Washington, DC, 1990.

[12] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," Computer Graphics, vol. 21, no. 4, pp.25–34, 1987.

[13] R. C. Eberhart, P. Simpson and R. Dobbins, "Computational intelligence PC tools," Academic Press, 1996.

[14] E. O. Wilson, "Sociobiology: the new synthesis," Belknap Press, Cambridge, MA, 1975.

[15] J. Kennedy, and R. C. Eberhart, "Particle swarm optimization," Proceedings IEEE International Conference on Neural Networks IV, Piscataway, NJ, pp. 1942–1948, 1995.

[16] W. T. Reeves, "Particle systems – a technique for modeling a class of fuzzy objects," ACM Transactions on Graphics vol.2, no.2, pp.91–108, 1983.

[17] P. J. Angeline, "Evolutionary optimization versus particle swarm optimization: philosophy and performance differences," In: Porto VW, Saravanan N,Waagen D and Eiben AE (eds.). Evolutionary Programming VII, pp. 601–610, 1998.

[18] R. C. Eberhart, and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," In: Porto VW, Saravanan N, Waagen D and Eiben AE (eds.). Evolutionary Programming VII, pp. 611–616. Springer, 1998

[19] Y. Shi, and R. C. Eberhart, "Parameter selection in particle swarm optimization," In: Porto VW, Saravanan N, Waagen D and Eiben AE (eds) Evolutionary Programming VII, pp. 611–616. Springer, 1998

[20] Y. Shi, and R.C. Eberhart, "A modified particle swarm optimizer," Proceedings of the IEEE Conference on Evolutionary Computation. AK, Anchorage, 1998

[21] B. B. Misra, B. N. Biswal, P. K. Dash, and G. Panda, "Simplified polynomial neural network for classification task in data mining," Proc. of the IEEE Congress on Evolutionary Computation (CEC), pp. 721–728, 2007.

[22] J. Kennedy, "The behavior of particles," In: Porto VW, Saravanan N, Waagen D and Eiben AE (eds.). Evolutionary Programming VII, pp. 581–590. Springer, 1998.

[23] A. Carlisle, and G. Dozier, "An off-the-shelf PSO," Proceedings of the Particle Swarm Optimization Workshop, pp. 1–6, 2001.

[24] B. B. Misra, S. C. Satapathy, B. N. Biswal, P. K. Dash, and G. Panda, "Pattern classification using polynomial neural networks," IEEE Int. Conf. on Cybernetics & Intelligent Systems (CIS), 2006.

[25] B. B. Misra, S. C. Satapathy, N. Hanoon, and P. K. Dash, "Particle swarm optimized polynomials for data classification," Proc. of the IEEE Int. Conf. on Intelligent Systems Design and Application, 2006.

[26] R. O. Duda, P. E. Hart and D. G. Stork, "Pattern classification," John Wiley and Sons (Asia) Pte. Ltd., 2001.

[27] S.-K. Oh, W. Pedrycz, and B.-J. Park, "Polynomial neural networks architecture: analysis and design," Computers and Electrical Engineering, vol. 29, pp. 703-725, 2003.

[28] T.-S. Lim, W.-Y. Loh and Y. S. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," Machine Learning, vol. 40, pp. 203-228, 2000.