A condensed polynomial neural network for classification using swarm intelligence

S. Dehuri[a,*], B.B. Misra[b], A. Ghosh[c], S.-B. Cho[d]

[a] Department of Information and Communication Technology, Fakir Mohan University, Vyasa Vihar, Balasore 756019, Orissa, India
[b] Department of Computer Science and Engineering, College of Engineering, Bhubaneswar 751024, Orissa, India
[c] Machine Intelligence Unit and Center for Soft Computing Research, Indian Statistical Institute Kolkata, 203 B.T. Road, Kolkata 700108, India
[d] Soft Computing Laboratory, Department of Computer Science, Yonsei University, 134 Shinchon-dong, Sudaemoon-ku, Seoul 120-749, Republic of Korea

ARTICLE INFO

Article history:
Received 29 April 2008
Received in revised form 11 August 2010
Accepted 12 December 2010
Available online 17 December 2010

Keywords:
Polynomial neural network
Classification
Particle swarm optimization
Partial descriptor
Data mining

ABSTRACT

A novel condensed polynomial neural network using particle swarm optimization (PSO) technique is proposed for the task of classification in this paper. In solving classification task classical algorithms such as polynomial neural network (PNN) and its variants need more computational time as the partial descriptions (PDs) grow over the training period layer-by-layer and make the network very complex. Unlike PNN the proposed network needs to generate the partial description for a single layer. The discrete PSO (DPSO) is used to select a relevant set of PDs as well as features with a hope to get better accuracy, which are in turn fed to the output neuron. The weights associated with the links from hidden to output neuron is optimized by PSO for continuous domain (CPSO). Performance of this model is compared with the results obtained from PNN. Simulation result shows that the performance of this model both in processing time and accuracy, is encouraging for harnessing its power in domain with large and complex data particularly in data mining area.

1. Introduction

In conjunction with the exponential growth of the information and communication technologies, we have witnessed a proliferation of databases with varying sizes of different degrees of complexities. Nevertheless, as the number of instances and its dimension grows, it is not that easy to analyze and retrieve high-level knowledge from the same databases. There are not as many off-the-shelf solutions for data analysis as there are for database creation and management; furthermore, they are pretty harder to suit to our needs. Data mining comprehend the actions of (semi) automatically looking for, identifying, validating, and using for decision making in data that might be categorized into classification, clustering, and association rule mining. In this paper we are giving emphasis on the problem of classification. Classification [1–4] is also treated as a challenging problem in pattern recognition and forecasting.

The goal of classification is to assign a class label from a predefined set of classes to an unknown sample based on the model of preference. Therefore, classification is based on some discovered model, which forms an important piece of knowledge about the application domain. Neural network based classifiers like multi-layer perceptron with back propagation learning are available for classification. The reasons why it is being not yet exploited in data mining area are due to its longer training time, difficult to decide the number of neurons and number of layers and finally its black box nature, where knowledge learned is concealed with large number of connections.

To alleviate the shortcomings of NNs, polynomial neural network (PNN) based on Group Method of Data Handling (GMDH) approach suggested by Ivakhnenko [5–7] can be used for classification purposes. Polynomial neural networks are multi-layer partial descriptions (PDs) which produce high order multivariate polynomial mappings. The approach is based on evolutionary strategy where PNN generates populations or layers of PDs and then trains and selects those PDs, which provide the best classification. During learning the PNN model grows the new population of PDs and the number of layers until a predefined criterion is met. Thereby the complexity of the architecture increases and it is very difficult to comprehend by human user. Such models can be comprehensively described by a set of short-term polynomials. Coefficients of PNN can be estimated by least square fit.

The network architecture grows depending on the number of input features, PNN model selected, number of layers required, and the number of PD's preserved in each layer. In turn the architecture becomes very complex, requires huge memory and computation time. Hence to cope with this problem of PNN we propose a con-

* Corresponding author. Tel.: +91 94372 89873; fax: +82 2 365 2579.
E-mail addresses: satchi.lapa@gmail.com (S. Dehuri), misra_bijan@yahoo.co.in (B.B. Misra), ash@isical.ac.in (A. Ghosh), sbcho@yonsei.ac.kr (S.-B. Cho).

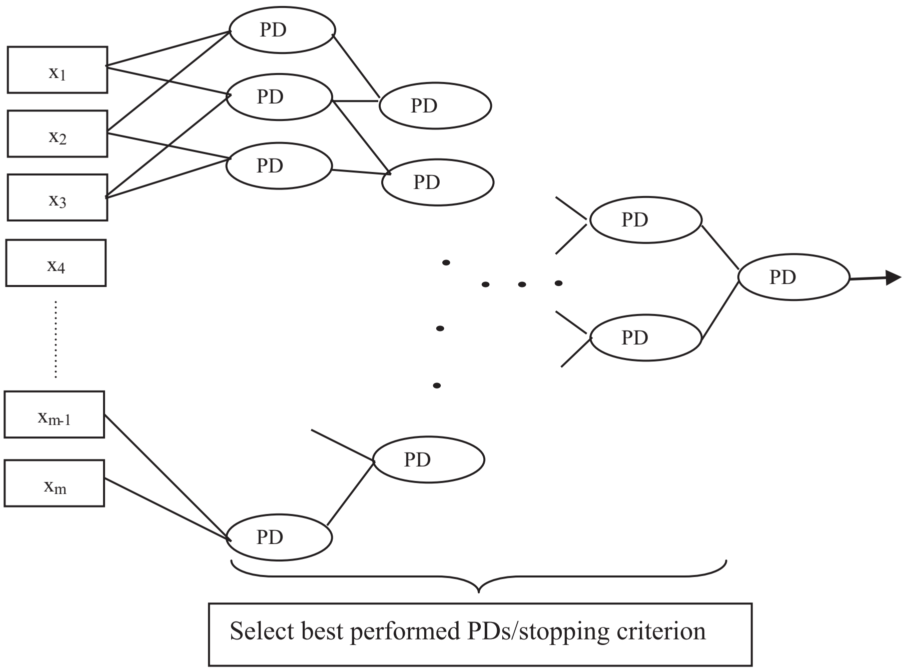

Fig. 1. Basic architecture of PNN model.

densed PNN model, which is a three layer architecture: input layer contains only the input features, hidden layer contains PDs and output layer contains only one neuron. We select an optimal set of PD's generated in the hidden layer along with the input features using the discrete PSO (DPSO) technique [8–10]. This optimal set is fed to the output layer. In conjunction the weights between hidden layer and output layer are optimized by PSO for continuous domain (CPSO). Why both version of PSO is chosen? The reason is that compared to other evolutionary algorithms (e.g. GAs), DPSO and CPSO is easy to implement and require very few parameters to adjust [11].

The rest of the paper is organized as follows. Section 2 describes the basics of PNN. In Section 3, PSO is discussed. The proposed model is formulated and discussed in Section 4. In Section 5, simulation result of the model is presented. Section 6 summarizes this paper with a prospect of future research directions.

2. Polynomial neural network

2.1. PNN architecture

The PNN architecture is based on the Group Method of Data Handling (GMDH) [12]. GMDH was developed by Ivakhnenko in late 1960s to identify non-linear relationship between input and output variables. However, there are several drawbacks associated with the GMDH such as its limited generic structure and overly complex network, and hence prompted a new class of polynomial neural networks (PNNs). In summary, these networks come with a high level of flexibility as each PD can have a different number of input variables as well as exploit a different order of polynomial (say linear, quadratic, cubic, etc.). Unlike neural networks whose topologies are commonly fixed prior to all detailed (parametric) learning, the PNN architecture is not fixed in advance but becomes fully optimized (both structurally and parametrically).

There are various types of PNN topology are developed so far, however it is worth noting to cover the basic one for getting more concrete idea in later part of the paper. The PNN architecture utilizes a class of polynomials such as linear, quadratic and cubic. By choosing the most significant number of variables and an order of the polynomial among these available forms, we can obtain the best ones from the extracted PDs according to selected nodes of each layer. Additional layers are generated until the best performance of the extended model has been reached. Such methodology leads to an optimal PNN structure. Let us assume that the input–output of the data is given in the following form:

$(X_i, y_i) = (x_{i1}, x_{i2}, ..., x_{im}, y_i)$, where $i = 1, 2, 3, \ldots, n$, n is the number of samples and m is the number of features. In matrix form we represent as follows:

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

The input–output relationship of the above data by PNN model can be described in the following manner: $y = f(x_1, x_2, \ldots, x_m)$.

The estimated output of variables can be approximated by Volterra functional series, the discrete form of which is Kolmogorov–Gabor polynomial [4] and is written as follows.

$$y = a_0 + \sum_{1 \le i \le m} a_i x_i + \sum_{1 \le i,j \le m} a_{ij} x_i x_j + \sum_{1 \le i,j,k \le m} a_{ijk} x_i x_j x_k + \cdots \qquad (1)$$

where a_k denotes the coefficients or weights of the Kolmogorov–Gabor polynomial and **x** vector is the input variables. The architecture of the basic PNN is shown in Fig. 1.

Each PDs the basic building block of the PNN model is shown in Fig. 2.

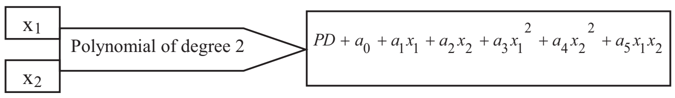

Fig. 2. Basic building blocks of PNN architecture.

To compute the estimated output y, we construct a PD for each possible pair of independent variables. For example, if the number of independent variables is m, then the total number of possible PDs is ${}^{m}C_2$. Here one can determine the parameters of PDs by the least square fit method by using given training samples. Furthermore we choose the optimal set of PDs from the first layer and construct a new set of PDs for the next layer of PNN and repeat this operation until stopping criterion is met. Once the final layer PD has been constructed, the node that shows the best performance is selected as the output node and all remaining are discarded. Furthermore by back tracking the nodes of the previous layers that do not have influence on the output node PD are deleted.

2.2. High-level algorithm of PNN

The high level algorithm of PNN is described as the following sequence of steps:

1. Determine the system's input variables and if needed carry out the normalization of input data.
2. Partition the given samples into training and testing samples: the input–output dataset is divided into two parts: training and test part. Training part is denoted as TR and test part is denoted as TS. Let the total number of samples is n. Then obviously we can write $n = TR + TS$. Using training part we construct the PNN model (including an estimation of coefficients of the PDs of every layer of PNN) and test data is used to evaluate the estimated PNN.
3. *Select a structure of the PNN*: The structure of the PNN is selected based on the number of input variables and the order of PDs in each layer. The PNN structures can be categorized into two types, namely a basic PNN and a modified PNN. In the case of basic PNN the number of input variables of PDs is the same in every layer, whereas in modified PNN the number of input variables of PDs varies from layer to layer.
4. *Generate PDs*: In particular, we select the input variables of a node from m input variables $x_1, x_2, \ldots, x_m$. The total number of PDs located at the current layer differs according to the number of selected input variables from the nodes of the preceding layer. This results in $c = m!/r!(m-r)!$ nodes, where r is the number of chosen input variables. The choice of the input variables and the order of a PD is very important to select the best model with respect to the characteristics of the data, model design strategy, non-linearity and the predictive capability.
5. *Estimate the coefficient of the PD*: The vector of coefficients $\vec{a} = (a_0, a_1, a_2, a_3, a_4, a_5)$ is derived by minimizing the mean squared error between y_i and $\tilde{y}_{ji}$,

$$E = \frac{1}{TR}\sum_{i=1}^{TR}(y_i - \tilde{y}_{ji})^2,$$

where $\tilde{y}_{ji} = a_0 + a_1x_p + a_2x_q + a_3x_p{}^2 + a_4x_q{}^2 + a_5x_px_q$, $1 \le p, q \le m, j = 1, 2, 3, \ldots, m(m-1)/2$.

In order to find out the coefficients, we need to minimize the error criterion E. Differentiating E with respect to all the coefficient we get the set of linear equations. In matrix form we can write as $Y = X \cdot A$,

Equivalently, $X^T \cdot Y = X^T \cdot X \cdot A$

$\Rightarrow A = (X^T \cdot X) \cdot X^T \cdot Y$.

This procedure is implemented repeatedly for all nodes of the layer and also for all layers of PNN starting from the input layer and moving to the output layer.

Further, the following simple algorithm can find out the index of the input features for each PD.

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

6. *Select PDs with best predictive capability*: Each PD is estimated and evaluated using both the training and testing data sets. Using the evaluated values, choose PDs which give the best predictive performance for the output variable. Normally we use a pre-specified cutoff value of the performance for all PDs. In order to be retained at the next generation the PD has to exhibit its performance above the cutoff value.
7. *Check the stopping criterion*: Two termination methods can be exploited.
   7.1 The following stopping condition indicates that an optimal PNN model has been accomplished at the previous layer, and the modeling can be terminated. This condition reads as $E_c \ge E_p$,where E_c is a minimal identification error of the current layer, and E_p denotes a minimal identification error that occurred at the previous layer.
   7.2 The PNN algorithm terminates when the number of iterations predetermined by the designer is reached. When setting up a stopping (termination) criterion, one should be prudent in achieving a balance between model accuracy and an overall computational complexity associated with the development of the model.
8. *Determine new input variables for the next layer*: If any of the above two criterion fails then the model has to be expanded.

In this work, the PDs along with the features are selected by DPSO. Section 3 will describe how DPSO is working in general.

3. Particle swarm optimization

Particle swarm optimization (PSO) introduced by Kennedy and Eberhart [10] in 1995, is an emerging population based optimization method. PSO has been applied successfully to optimize continuous non-linear functions [10], neural network [13], non-linear constraint optimization problems [14], etc. Most of the applications have been concentrated on solving continuous optimization problems, but the studies of PSO on discrete optimization problems are relatively few. In this paper both continuous and discrete version of PSO is used for optimization of the proposed model. Therefore it is worth noting to discuss both continuous and discrete versions of PSO.

3.1. PSO for continuous domain (CPSO)

In PSO a set of randomly generated solutions (initial swarm) propagates in the design space towards the optimal solution over a number of iterations (moves) based on large amount of information about the design space that is assimilated and shared by all members of the swarm. A complete chronicle of the development of the PSO algorithm form merely a motion simulator to a heuristic optimization approach is described in [10].

The standard PSO algorithm broadly consists of three computational steps:

(i) generation of particles' position and velocities;
(ii) updating the velocity of each particle; and
(iii) updating the position of each particle.

Here, a particle refers to a potential solution to a problem. A particle $\vec{x}_k$ in d-dimensional design space is represented as $\vec{x}_k = \langle x_{k1}, x_{k2}, x_{k3}, \ldots, x_{kd} \rangle$, where $k = 1, 2, 3, \ldots, N$, N is the number of particles in a swarm. Each particle has its own velocity and maintains a memory of its previous best position $\vec{p}_k = \langle p_{k1}, p_{k2}, \ldots, p_{kd} \rangle$. Let the vector $\vec{p}_g = \langle p_{g1}, p_{g2}, \ldots, p_{gd} \rangle$ refer to the position found by the gth member of its neighborhood (i.e., entire swarm) that has had the best performance so far. The particle changes its position from iteration to iteration based on velocity updates. In each iteration $\vec{p}_g$ and $\vec{p}_k$ of the current swarm is combined with some weighting coefficients to adjust the velocity of the particles in the swarm. The velocity and position of a particle can be updated using Eqs. (2) and (3) respectively.

$$\vec{v}_k(t+1) = w \otimes \vec{v}_k(t) + \vec{c}_1 \otimes \vec{r}_1(t) \otimes (\vec{p}_k(t) - \vec{x}_k(t)) + \vec{c}_2 \otimes \vec{r}_2(t) \otimes (\vec{p}_g(t) - \vec{x}_k(t)). \tag{2}$$

$$\vec{x}_k(t+1) = \vec{x}_k(t) + \vec{v}_k(t+1). \tag{3}$$

The symbol $\otimes$ denotes point-by-point vector multiplication. The inertia/momentum factor $w (0 < w \leq 1)$, the self confidence factor c_1 and swarm confidence factor c_2 are non-negative real constants. Randomness (useful for good state space exploration) is introduced via the vectors of random numbers $\vec{r}_1$ and $\vec{r}_2$. They are usually selected as uniform random numbers in the range [0,1]. Over the years many researchers have fine tuned these parameters and found out very standard optimized values [15]. The three steps of velocity update, position update, and fitness computations are repeated until a desired convergence criterion is met. There are also other alternative criteria to stop, however convergence and stability of the standard PSO has been proposed by many researchers [16].

3.2. PSO for discrete domain (DPSO)

In the above discussion, PSO is restricted in real number space. However, many optimization problems are set in a space featuring discrete or qualitative distinctions between variables. To meet the need, Kennedy and Eberhart [8] developed a discrete version of PSO. Discrete PSO essentially differs from the original (or continuous) PSO in two characteristics. First, the particle is composed of the binary variable. Second, the velocity must be transformed into the change of probability, which is the chance of the binary variable taking the value one.

By Eq. (3), each particle moves according to its new velocity. Recall that particles are represented by binary variables. For the velocity value of each bit in a particle, Kennedy and Eberhart [8] claim that higher value is more likely to choose 1, while lower value favors the 0 choice. Furthermore, they constrain the velocity value to the interval (0, 1) by using the following sigmoid function:

$$s(v_{kd}(t)) = \frac{1}{1 + \exp(-v_{kd}(t))}, \tag{4}$$

where $s(v_{kd}(t))$ denotes the probability of bit $x_{kd}(t)$ taking 1. In this work we extend the DPSO proposed in [9] and used synergistic way with the proposed method.

4. Proposed method

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 we found that PDs giving better result combined with PDs giving inferior result may improve the performance in subsequent layers. 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 with large scale [17].

Each PD tries to approximate the input output relationship of the dataset. In the proposed model, we have developed PDs for a single layer (i.e. hidden layer). The generated PDs along with the original features are undergone for evaluation. The optimal set of PDs as well as features obtained after evaluation is given as the input to the output neuron. For searching such an optimal set of PDs along with features is accomplished in this paper by discrete version of the PSO. Additionally the weight vector between hidden and output neuron is optimized by a continuous version of PSO. Fig. 3 shows the proposed abstract model.

In our model, m represents the number of features in the dataset and k represents the 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. The following procedure with synergistic effect of CPSO and DPSO describes the architecture shown in Fig. 3.

1. **DETERMINE** *the number of input variables and the order of the polynomial forming a partial description (PD) of the data.*
2. **ESTIMATE** *the coefficients for the PDs of the hidden layer.*
3. **SELECTION** *the PDs and features with the best predictive accuracy by DPSO.*
4. **RANDOMLY INITIALIZE** *the weights between hidden layer and output layer.*
5. **ESTIMATE MSE** *by comparing the generated outputs with the desired outputs.*
6. **CHECK WHETHER THE STOPPING CRITERION** *is met or not. If not then go to the step 7 otherwise stop.*
7. **UPDATE THE WEIGHTS** by **CPSO** *and go to the step 5.*

4.1. DPSO in proposed method

The DPSO algorithm deals directly with discrete variables (attributes and PDs), its swarm of candidate solutions contains particles of different sizes. Potential solutions to the optimization problem at hand are represented by a swarm of particles. There are 'N' particles in a swarm. The size of each particle may vary from 1 to n, where $n = k + m$ is the number of variables – PDs and attributes in this work. In this context, the length of a particle refers to the number of different attribute and PDs indices. In this work each particle is logically partitioned into two parts.

For example, given $i, j \in \{1, 2, 3, \ldots, N\}$ in DPSO it may occur that a particle p_i in the swarm has size 7 ($p_i = \{*, *, *, * | +, +, +\}$), '*' represent PDs and '+' represent feature, whereas another particle p_j in the same swarm has size 3 ($p_j = \{* | +, +\}$) and so forth, or any other sizes between 1 to $m+k$.

4.2. Particle representation

The $m+k$ number of features and PDs are represented by a unique positive integer number or index. These numbers or indices vary from 1 to $m+k$. A particle is a subset of non-ordered indices without representation, for example, $p_i = \{1, 2, 6 | 10, 7\}$, $i \in \{1, 2, 3, \ldots, N\}$.

4.3. Swarm initialization

The initial swarm used by the DPSO is always identical to the initial swarm randomly created by binary PSO (BPSO) [8]. They differ only in the way in which solutions are represented. The conver-

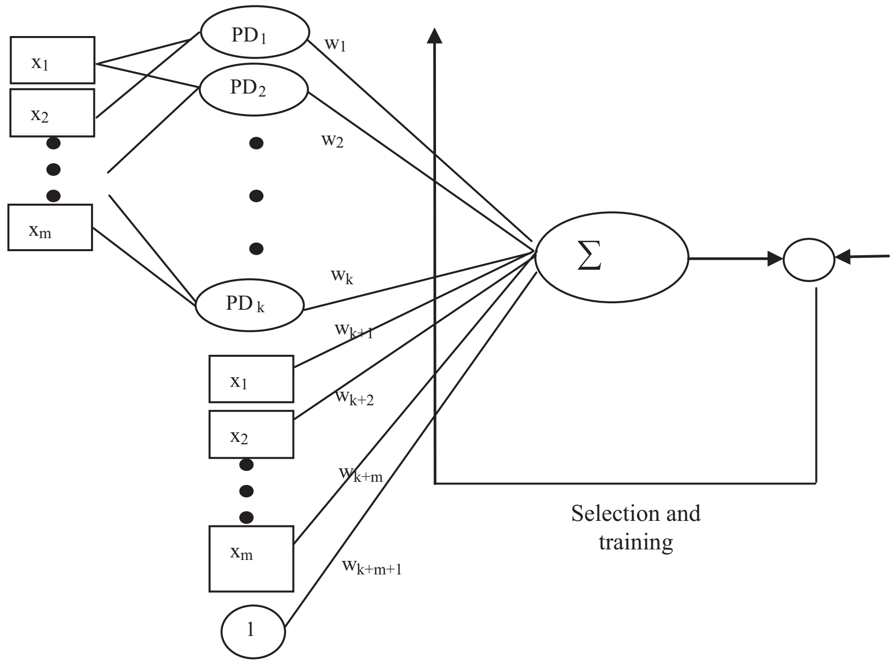

Fig. 3. The proposed model.

sion of every particle in the initial swarm of solutions of the binary PSO to the discrete PSO initial swarm as follows. The index of every PDs and features are copied to the new solution of the DPSO initial swarm.

For example, an initial candidate solution for the binary PSO algorithm equal to $bp_i = \langle 1, 0, 1|0, 1, 1 \rangle$ is converted into $dp_i = \langle 1, 3|5, 6 \rangle$ for the DPSO algorithm. Initializing the particle p_i in this way causes different particle in DPSO, to have different sizes. In the DPSO algorithm, for simplicity once the size of a particle is determined at the initialization, the particle will keep the same size during the entire execution of the algorithm.

4.4. Representation of particle velocity

The DPSO algorithm does not use a vector of velocities as the standard PSO algorithm does. It works with proportional likelihoods instead. Arguably the notion of proportional likelihood used in the DPSO algorithm and the notion of velocity used in the standard PSO are somewhat similar. Each particle in DPSO is associated with a 2-by-$m + k$ array of proportional likelihoods (denoted as V), where 2 is the number of rows in this array and $m + k$ is the number of columns-note that the number of columns in V is equal to the number of features and PDs of the problem.

This is an example of a generic proportional likelihood array,

$$V = \begin{pmatrix} \text{proportional} - \text{likelihood} - \text{row} \\ \text{attribute} - \text{PDs} - \text{index} - \text{row} \end{pmatrix}.$$

Each of the n-elements in the first row of V represents the proportional likelihood that an attribute and PDs be selected. The second row of V shows the indices of the attributes associated with the respective proportional likelihoods.

There is a one-to-one correspondence between the columns of this array and the features and PDs of the problem domain. At the beginning all elements in the first row of V are set to 1. After the initial swarm of particles is generated, this array is always updated before a new configuration for the particle associated to its generated. The updating of the likelihoods V is based on the current, best and global best position of the particle and three constant updating factors namely p_α, p_β and p_δ. These updating factors determine the strength of the contribution of current, best and global best towards the optimal accuracy. Note that p_α, p_β and p_δ are parameters chosen by the user. The contribution of these parameters to the updating of V is as follows. All indices present in current position of the particle have their correspondent proportional likelihood increased by p_α. In addition to that, all indices present in best position so far attained by the particle have their correspondent proportional likelihood increased by p_β. The same for global best position for which the proportional likelihoods are increased by p_δ. In this work the values of p_α, p_β and p_δ is determined empirically but this can be fine tuned depending on the complexity of the problem. The new updated array V replaces the old one and will be used to generate a new configuration to the particle associated to it as follows.

4.5. Updating particle position

The proportional likelihood array V is then used to update the position of the particle – the particle associated to V. To complete the process a series of operations is performed on the array. To start with, every element of the first row of the array is multiplied by a uniform random number between 0 and 1. A new random number is drawn for every single multiplication performed.

A new particle position is then defined by ranking the columns in V by the values in its first row. That is the elements of the first row of the array are ranked in a decreasing order of value, and the indices of the features and PDs – in the second row of the array V – follow their respective proportional likelihoods. The next operation now is to select the indices that will compose the new particle position.

The CPSO follows what is described in Section3. Once the algorithmic framework has been explained, the next section describes the simulation of the proposed model with data set obtained from University of California, Irvine (UCI) repository [18].

5. Simulations and results

The performance of the model 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

Table 1
Description of datasets used.

Dataset	Patterns	Attributes	Classes	Patterns in class 1	Patterns in class 2	Patterns in class 3
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	–

Table 2
Division of dataset and its pattern distribution.

	Patterns	Patterns in class 1	Patterns in class 2	Patterns in class 3
IRIS				
Set 1	75	25	25	25
Set 2	75	25	25	25
WINE				
Set 1	89	29	36	24
Set 2	89	30	35	24
PIMA				
Set 1	384	134	250	–
Set 2	384	134	250	–
BUPA				
Set 1	172	72	100	–
Set 2	173	73	100	–

Table 3
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
$p_\alpha, p_\beta, p_\delta$	{0.12, 0.23, 0.27}

from the UCI machine repository [18]. Table 1 presents the summary of the main features of the datasets used for experimental studies.

The data set is divided into two parts. The division of datasets and its class distribution is shown in Table 2.

One part is used for building the model and other part is used for testing the model. The protocol used for our simulation studies is given in Table 3.

The average percentage of correct classification obtained for the test sets is provided in Table 4 for the purpose of comparison. The measurement of the correct classification of the proposed model and PNN should be a reliable estimate of how well that model classifies the test samples – unseen during the training phase. The confusion matrix is used for the estimation of classification accuracy.

In Table 5 the percentage of PDs used in our model is given, which is also very crucial aspect to obtain an optimum model.

Table 6 shows the processing time of both PNN and proposed method.

Table 4
Comparison of average percentage of correct classification of test sets with PNN and proposed model.

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

Table 5
Percentage of PDs used by the proposed model to obtain the optimum model.

Data set	Total number of possible PDs	% of PDs used in optimized model	
		Set 1	Set 2
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 6
Comparison of processing time performance of PNN with proposed method (in seconds).

Datasets	PNN	Proposed method
IRIS		
Set 1	129	0.6563
Set 2	125	0.6719
WINE		
Set 1	225	4.5625
Set 2	224	4.7031
PIMA		
Set 1	783	8.1406
Set 2	793	8.4531
BUPA		
Set 1	352	2.2969
Set 2	353	2.4219

The mathematical model obtained by our model for Iris dataset is presented as an example.

$$\mathrm{PD}_2^1 = [-0.96751, -0.25663, 0.65004, -0.13994, 0.044319, 0.10067] * \mathrm{poly}(x_1, x_3),$$

$$\mathrm{PD}_3^1 = [1.633, -1.0407, 1.4825, -0.051333, 0.09108, -0.067678] * \mathrm{poly}(x_1, x_4);$$

$$\mathrm{PD}_1^1 = [-1.7572, 1.4898, -2.4317, 0.3308, -0.15092, -0.018686] * \mathrm{poly}(x_1, x_2);$$

$$\mathrm{PD}_{16}^2 = [2.0965, 1.4134, -0.84284, 0.046201, -0.57289, 0.079276] * \mathrm{poly}(\mathrm{PD}_2^1, x_3);$$

$$\mathrm{PD}_{28}^2 = [2.2683, 1.7305, -1.2497, -0.22651, 0.15639, 0.15237] * \mathrm{poly}(\mathrm{PD}_3^1, x_2);$$

$$\mathrm{PD}_{38}^2 = [-1.8284, -0.58075, 0.42213, 0.081458, -0.25631, 0.015491] * \mathrm{poly}(\mathrm{PD}_1^1, x_3);$$

$$\mathrm{PD}_{496}^3 = [-0.054529, 0.54087, 0.45923, -1.1876, 0.6309, 0.60812] * \mathrm{poly}(\mathrm{PD}_{16}^2, \mathrm{PD}_{28}^2);$$

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

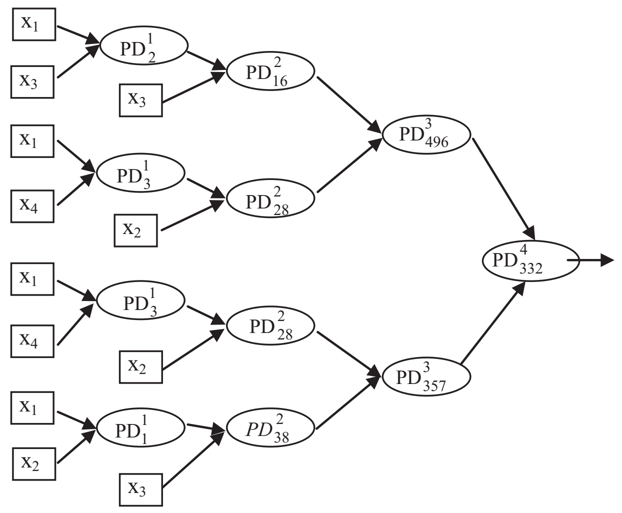

Fig. 4. PNN model for Iris dataset.

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

where function ploy(a$_1$, a$_2$) {return [1, a$_1$, a$_2$, a$_1$*a$_2$, a$_1\hat{2}$, a$_2\hat{2}$]^T;}, PD^i_j is the output of layer i and jth partial description, y is the output of the PNN model and x is the input features.

The architecture of the PNN model generated is given in Fig. 4.

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

$$PD^1_1 = [12.265, -27.379, 2.7124, -5.0129, 16.636, 2.6568] *\text{poly}(x_1, x_2),$$

$$PD^1_5 = [3.4633, -0.61437, -4.7075, 3.5712, 0.32404, -0.37042] *\text{poly}(x_2, x_4),$$

$$y = [0.02099x_4 + 0.62292PD^1_1 + 0.33345PD^1_5 + 0.10778];$$

The architecture of our model generated is given in Fig. 5.

The confusion matrix is an alternative ways to show the experimental results, which is obtained from our model for the entire dataset. The confusion matrices for the class 2 datasets are shown in Table 7 and the confusion matrix for the class 3 datasets are shown in Table 8.

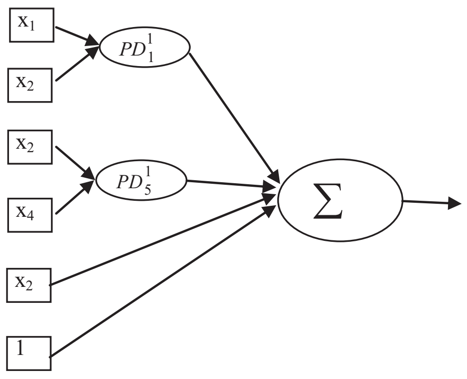

Fig. 5. Proposed model for iris dataset.

Table 7
Confusion matrix for two class datasets.

Actual	Predicted	
	c1	c2
BUPA		
C1	85	60
C2	30	170
PIMA		
C1	134	134
C2	44	456

Table 8
Confusion matrix for three class datasets.

Actual	Predicted		
	c1	c2	c3
IRIS			
c1	50	0	0
c2	1	49	0
c3	0	0	50
WINE			
c1	59	0	0
c2	0	70	1
c3	0	0	48

6. Conclusions

In this paper, we have proposed a condensed polynomial neural network using swarm intelligence for the classification task. Our model generates PDs for a single layer of the basic PNN model. DPSO selects the optimal set of PDs and input features, which are fed to the hidden layer. Further, the model optimizes the weight vectors using CPSO technique. The experimental studies demonstrated that our model performs better classification accuracy. In all the cases, the results obtained with the proposed model proved to be better than the PNN results. The performance of our model is better in terms of processing time, which is also treated as one of the crucial aspects in data mining. The future work includes more fine tuning of the parameters p_α, p_β and p_δ on biological data.

Acknowledgement

Dr. S. Dehuri would like to thank the Department of Science and Technology, Govt. of India vide letter number SR/BY/E-07/2007 dated 03-01-2008 for financial support under the BOYSCAST fellowship 2007–2008 and Prof. S.-B. Cho, Department of Computer Science, Yonsei University, Seoul, Korea for providing the Soft Computing Laboratory facilities.

References

[1] T.M. Mitchel, Machine Learning, McGraw Hill, 1997.
[2] Y.H. Pao, Adaptive Pattern Recognition Neural Networks, Addison Wesley, MA, 1989.
[3] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, John Wiley and Sons (Asia) Pte. Ltd, 2001.
[4] T.-S. Lim, W.-Y. Loh, Y.S. Shih, A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, Machine Learning 40 (2000) 203–228.
[5] A.G Ivakhnenko, Polynomial theory of complex systems, IEEE Transactions on Systems, Man and Cybernetics-I (1971) 364–378.
[6] A.G. Ivakhnenko, H.R. Madala, Inductive Learning Algorithm for Complex Systems Modelling, CRC Inc., Boca Raton, 1994.
[7] S.-K. Oh, W. Pedrycz, B.-J. Park, Polynomial neural networks architecture: analysis and design, Computers and Electrical Engineering 29 (2003) 703–725.
[8] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, in: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics vol. 5, 1997, pp. 4104–4109.
[9] E.S. Correa, A.A. Freitas, C.G. Jhonson, Particle swarm for attribute selection in Bayesian classification: an application to protein function prediction, Journal

of Artificial Evolution and Applications 2008 (2008), doi:10.1155/2008/876746, 12 pages, Article ID 876746.

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

[11] R.C. Eberhart, Y. Shi, Comparison between genetic algorithms and particle swarm optimization, in: V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben (Eds.), Evolutionary Programming VII, Springer, 1998, pp. 611–616.

[12] S.J. Farlow, The GMDH algorithm, in: S.J. Farlow (Ed.), Self-organizating Methods in Modelling: GMDH Type Algorithm, Marcel Dekker, New York, 1984, pp. 1–24.

[13] F. Van den Bergh, A.P. Engelbrecht, Cooperative learning in neural network using particle swarm optimizers, South African Computer Journal 26 (2000) 84–90.

[14] A.I. El-Galland, M.E. El-Hawary, A.A. Sallam, Swarming of intelligent particles for solving the nonlinear constrained optimization problem, Engineering Intelligent Systems for Electrical Engineering and Communications 9 (2001) 155–163.

[15] Y. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, in: V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben (Eds.), Evolutionary Programming VII, Springer, 1998, pp. 611–616.

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

[17] B.B. Misra, S. Dehuri, P.K. Dash, G. Panda, A reduced and comprehensible polynomial neural network for classification, Pattern Recognition Letters 29 (2008) 1705–1712.

[18] C.L. Blake, C.J. Merz, UCI repository of machine learning databases, 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.