available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cosrev

Survey

Multi-criterion Pareto based particle swarm optimized polynomial neural network for classification: A review and state-of-the-art

S. Dehuri[a,*], S.-B. Cho[b]

[a] Department of Information and Communication Technology, Fakir Mohan University, Vyasa Vihar, Balasore-756019, Orissa, India
[b] Soft Computing Laboratory, Department of Computer Science, Yonsei University, 262 Seongsanno, Seodaemun-gu, Seoul 120-749, Republic of Korea

ARTICLE INFO

Article history:
Received 5 July 2008
Received in revised form
14 November 2008
Accepted 14 November 2008

ABSTRACT

In this paper, we proposed a multi-objective Pareto based particle swarm optimization (MOPPSO) to minimize the *architectural complexity* and maximize the *classification accuracy* of a polynomial neural network (PNN). To support this, we provide an extensive review of the literature on multi-objective particle swarm optimization and PNN. Classification using PNN can be considered as a multi-objective problem rather than as a single objective one. Measures like *classification accuracy* and *architectural complexity* used for evaluating PNN based classification can be thought of as two different conflicting criterions. Using these two metrics as the criteria of classification problem, the proposed MOPPSO technique attempts to find out a set of non-dominated solutions with less complex PNN architecture and high classification accuracy. An extensive experimental study has been carried out to compare the importance and effectiveness of the proposed method with the chosen state-of-the-art multi-objective particle swarm optimization (MOPSO) algorithm using several benchmark datasets. A comprehensive bibliography is included for further enhancement of this area.

1. Introduction

Classification is one of the most studied tasks in Data Mining and Knowledge Discovery in Databases (DM and KDD) [1–7], pattern recognition [8–12], image processing [13–15] and bio-informatics [16–21]. In solving classification tasks, the classical algorithm such as PNN [22,23] and its variants [24] try to measure the performance by considering only one evaluation criterion, i.e. classification accuracy. However, one more important criterion like architectural complexity embedded in PNN is being completely ignored. The PNN architecture takes more computation time as the partial description (PDs) grows over the training period layer-by-layer and makes the network more complex. Furthermore, the complexity of PNN is based on the parameters such as the number of input variables, the order of the polynomial, the number of layers of the polynomial neural network and the number of PDs in a layer. These all together constitute

* *Corresponding author.* Tel.: +82 2 2123 3877; fax: +82 2 365 2579.
E-mail addresses: satchi.lapa@gmail.com (S. Dehuri), sbcho@cs.yonsei.ac.kr (S.-B. Cho).

1574-0137/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.cosrev.2008.11.002

the PNN architecture more complex. The implicit criterion (i.e. architectural complexity) is one of the main bottlenecks of PNN to use it for harvesting knowledge from large datasets. However, the pattern recognition and image processing community can tolerate its architectural complexity due to their small-scale datasets. Therefore, to make this network as a useful tool for data mining we need to minimize the complexity of the architecture without compromising the classification accuracy. In order for acquaintance in PNN we gave a comprehensive study on PNN architecture and its state-of-the-art in the first part of the article.

Decision-makers here desire solutions that simultaneously optimize multiple objectives such as architectural complexity and classification accuracy and obtain an acceptable tradeoff amongst objectives. Multi-criteria problems often characterize a range of solutions, none of which dominate the others with respect to the multiple objectives. These specify the Pareto-frontier of non-dominated solutions, each offering a different level of tradeoff. Multi-objective genetic algorithms (MOGAs) [25–30] are a popular approach to confronting these types of problem. The use of genetic algorithms (GAs) [31–33] as a tool of preference is due to such problems being typically complex, with both a large number of parameters to be adjusted, and several objectives to be optimized. GAs, which can maintain a population of solutions, are in addition able to explore several parts of the Pareto front simultaneously. Particle swarm optimization (PSO) [34–40] similarly has these characteristics. So, given the promising results reported in the literature [41] comparing PSO to GA techniques in the single-objective domain; a transfer of PSO to the MO domain seems an expected headway.

The architectural complexity, an important criterion of PNN, is reduced based on the following assumption. The proposed PNN model is a three-layer architecture: the input layer contains only the input features, the hidden layer contains PDs and the output layer contains only one neuron. We can select an optimal set from the PD's generated in the hidden layer along with the input features using MOPPSO as a driving force to further decrease the architectural complexity. This optimal set of features is fed to the output layer. Further, the proposed MOPPSO technique implicitly optimize the weight between hidden layer and output layer. Furthermore, to support the design methodology of the proposed technique we review and give several multi-objective particle swarm optimization (MOPSO) methods and their applications in the second part of the paper.

In a nutshell, the paper is organized into three parts: (i) review of PNN (ii) a comprehensive survey of MOPSO and its applications and (iii) proposed method for classification. Fig. 1 shows a hierarchical diagrammatic view of the three main wings of this article.

The remainder of this paper is organized as follows: Section 2 describes the design and analysis of basic architecture and algorithmic view of PNN and its state-of-the-art; In Section 3 we provide some basic concepts of multi-objective optimization and reviewed the current theoretical and practical development of PSO for multi-objective problems. The proposed method is formulated and discussed in Section 4; In Section 5 experimental study and comparative performance of the proposed method is presented; Section 6 draws the conclusions with a few lines of open research.

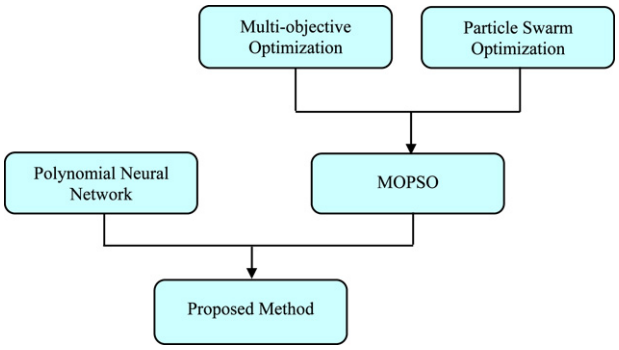

Fig. 1 – A hierarchical view of the components of this article.

2. Polynomial Neural Network

2.1. PNN architecture

The PNN architecture is based on the Group Method of Data Handling (GMDH) [42–44]. GMDH [45–49] was developed by Ivakhnenko in late 1960s for identifying non-linear connections between input and output variables, inspired by the form of Kolmogorov–Gabor polynomial. However, there are several drawbacks associated with the GMDH such as its limited generic structure, overly complex network, etc, and hence prompted a new class of neural networks known as polynomial neural networks (PNNs). In summary, these networks come with a high level of flexibility as each node (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 decided prior to all detailed (parametric) learning, the PNN architecture is not fixed in advance but becomes fully optimized (both structurally and parametrically).

Even though various types of topologies of the PNN are available [50], here we explain the basic architecture of the PNNs for familiarity. The PNN architecture utilizes a class of polynomials such as linear, quadratic, cubic, etc. By choosing the most significant number of variables and an order of the polynomial available, we can obtain the best ones from the extracted PDs according to the selected nodes of each layer. Additional layers are generated until the best performance of the extended model is obtained. Such methodology leads to an optimal PNN structure. Let us assume that the input–output of the data is given in the following form:

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

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

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

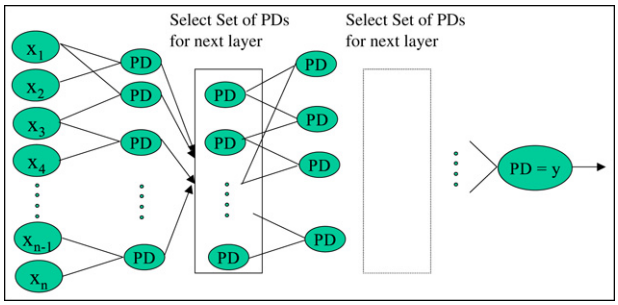

Fig. 2 – Generalized architecture of PNN.

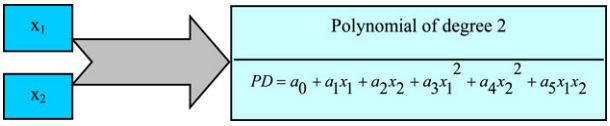

Fig. 3 – Basic building blocks of PNN architecture.

The estimated output of variables can be approximated by Volterra functional series, the discrete form of which is Kolmogorov–Gabor Polynomial [9] and is written as follows.

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

where a_k denotes the coefficients or weights of the Kolmogorov–Gabor polynomial and **x** is an input variable. The architecture of the basic PNN is shown in Fig. 2. The basic building block (i.e., a PD) of the PNN model is shown in Fig. 3.

To compute the value of 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 ${}^{m}C_2 = \frac{m!}{2!(m-2)!}$.

One can determine the parameters of PDs by the method of least square fit by using the training samples. Furthermore, we choose an optimal set of PDs from the first layer based on their predictive performance 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 PDs has been constructed, the node that shows the best performance is selected as the output node and the remaining PDs 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 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 be 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 are 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 the 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 the 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 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 $\tilde{y}_{ji} = a_0 + a_1 x_p + a_2 x_q + a_3 x_p^2 + a_4 x_q^2 + a_5 x_p x_q$, $1 \le p, q \le m$, $j = 1, 2, 3, \ldots, m(m-1)/2$, is computed on the basis of two input variables (i.e., $r = 2$) and 2nd order polynomial.

In order to find out the coefficients, we need to minimize the error criterion E. Differentiate E with respect to all the coefficients, we get the set of linear equations. In matrix form we can write as

$$Y = X.A,$$

equivalently, $X^T.Y = X^T.X.A \Rightarrow A = (X^T.X).X^T.Y$. The details description of these matrices are defined in Appendix.

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 towards the output layer.

Further the following simple algorithms can find out the index of the input features $\langle p, q\rangle, 1 \le p, q \le m, p \ne q$ for each PD_k, $k = 1(1)m(m-1)/2$ by assuming the number of layers for which we compute the PDs be one.

```
Features (p,q)_PD_k
 {
   k = 1;
   FOR i = 1 to m − 1
     FOR j = i + 1 to m
       Then PD_k^1 receives input from the features p = i
        and q = j;
       k = k + 1;
     END FOR
   END FOR
 }
```

Alternatively the index k for each PD can be computed directly from the tuple $\langle p, q\rangle, 1 \le p, q \le m, p \ne q$ by the following formula:

$$k = \left(\sum_{t=1}^{p}(m-t)\right) - (m-q).$$

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 prespecified 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.
   If any of the above two criteria fails, then go to step 4 else execute the step number 8.
8. Post processing steps: Once the final layer is constructed (i.e., the else part of step 7 is executed) then the node exhibit best performance can be chosen as the output node. All the remaining nodes in that layer are discarded. Further all the nodes of the previous layers that do not have influence on the estimated output node are also removed by back tracing.

2.3. *Recent developments of PNN*

In this section, we will introduce few of the selected and more informative polynomial neural networks, which are recently developed and applied to different areas of interest.

Vasechkina and Yarin [51] give a proposal of evolving polynomial neural network by means of genetic algorithm and shown a direction to use in biology, ecology and other neural sciences. However, the work does not fully exploit the nature of polynomial neural architecture. That means the proposed method is evolving feed-forward neural network architecture by a self-organizing process, and does not require learning as a separate process since evaluation of weights is carried out simultaneously with the architecture construction. The NN architecture is built by adding hidden layers to the network, while configuration of connections between neurons is defined by means of GA. The output of the neuron is represented as a polynomial function of the inputs with coefficients evaluated using the least-squares method.

Schetinin [52] showed a direction of how polynomial neural network can successfully extract polynomial classification rules from labeled electroencephalogram (EEG) signals. To represent the classification rules in an analytical form, he used the polynomial neural networks trained by a modified Group Method of Data Handling (GMDH). Park et al.'s. [53] introduced a concept of fuzzy polynomial neural networks (FPNNs) a hybrid modeling architecture combining polynomial neural networks (PNNs) and fuzzy neural networks (FNNs) to extract rules. In this method FNNs contribute to the formation of the premise part of the rule-based structure of the FPNN. The consequent part of the FPNN is designed using PNNs.

Oh and Pedrycz [54] introduced and investigate a class of neural architectures of polynomial neural networks and discuss a comprehensive design methodology. Their experimental part of the study involves two representative time series such as Box–Jenkins gas furnace data and a pH neutralization process. Aksyonova et al.'s. [55] presents a robust polynomial neural network for quantitative-structure activity relationship studies. The method is able to select nonlinear models characterized by high prediction ability; it is insensible to outliers and irrelevant variables. This feature of this work can be a good attracting point for data mining researchers. Oh et al.'s. [56] proposed a new approach to self organizing polynomial neural networks by means of genetic algorithms. The proposed GA-based SOPNN gives rise to a structurally optimized structure and comes with a substantial level of flexibility in comparison to PNNs. The design procedure applied in the construction of each layer of a PNN deals with its structural optimization involving the selection of preferred nodes with specific local characteristics (such as the number of input variables, the order of the polynomial, and a collection of the specific subset of input variables) and addresses specific aspects of parametric optimization. Oh et al.'s. [57] developed a new architecture of hybrid fuzzy polynomial neural networks (HFPNN) that is based on a genetically optimized multi-layer perceptron and develop their comprehensive design methodology involving mechanisms of genetic optimization. The construction of HFPNN exploits fundamental technologies of computational intelligence, namely fuzzy sets, neural networks, and genetic algorithms (GAs).

Kim and Park [58] proposed an architecture that is similar with the architecture proposed in [56] but it is applied to a different domain of interest. The model is implemented to better use the optimal inputs and the order of polynomial in each node of PNN. The appropriate inputs and order are evolved accordingly and are tuned gradually throughout the GA iterations. They have employed a binary encoding and each chromosome is made of three sub-chromosomes, which represent the order, number of inputs, and input candidates for modeling. Liatsis et al.'s. [59], proposed an adaptive polynomial neural networks for time series forecasting. In this work the structure and weight vectors are determined by the use of evolutionary algorithms. Zarandi et al.'s. [60] uses the fuzzy polynomial neural networks for approximation of the comprehensive strength of concrete. To enhance the performance of FPNN, back propagation and list square error algorithms are utilized for the tuning of the system. Six

different FPNN architectures are constructed, trained and tested using the experimental data of 458 different concrete mix-designs collected from three distinct sources. Park et al.'s. [61], studied a new polynomial neural network with layer over passing structure has been developed to replace a relatively time consuming reservoir simulator through robust and systematic search algorithm. The networks are subject to some form of training based on a representative sample of simulations that can be used as a re-usable knowledge base of information for addressing many different management questions.

Misra et al.'s. [24] proposed a reduced and comprehensible polynomial neural network (RCPNN) for classification. In this network the partial descriptions are developed for a single layer and the output of these PDs along with original set of features are fed to the output layer having only one neuron. The weights between hidden layer and output layer are optimized by two different methods such as gradient decent and particle swarm optimization (PSO) [36]. Misra et al.'s. [62] proposed another variants of RCPNN for classification task of data mining known as RPNSN. In this network, the PDs are developed for a single layer. Then PSO is used for selecting the optimal set of PDs and features in a kind of wrapper approach, which are then fed to the output layer.

3. PSO for multi-objective problems

3.1. Multi-objective problems

Multi-objective optimization otherwise known as multi-criteria optimization or, multi-performance or vector optimization is defined as the problem of finding "A vector of decision variables which satisfies and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria, which usually conflict with each other. Hence the term optimizes means finding such solution, which would give the values of all the objective functions acceptable to the user.

Multi-objective optimization methods as the name suggests, deal with finding optimal solutions to problems having multiple objectives. So in this type of problems finding one solution that is optimum with respect to single criterion never satisfies the user. Mathematically this can be stated and visualized as follows:

Find a vector $\vec{x} = \langle x_1, x_2, \ldots, x_d \rangle$, which can optimize the vector functions simultaneously $\vec{f}(\vec{x}) = \langle f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x}) \rangle$.

Fig. 4 shows a mapping of a decision variable between decision space denoted as R^d to objective space F^n.

Multi-objective optimization concept states that when many objectives are simultaneously optimized, there is no single optimal solution; instead there is a set of optimal solution called Pareto optimal set, each one considering a certain trade-off among the objectives.

Strong Pareto optimality: We say that a vector of decision variables $\vec{x} \in R^d$ is strong Pareto optimal if there does not exist another $\vec{x}' \in R^d$ such that (i) $f_i(\vec{x}') \leq f_i(\vec{x})$ for all $i = 1, 2, \ldots, n$ and (ii) $f_i(\vec{x}') < f_i(\vec{x})$ for at least one i.

Weak Pareto optimality: A vector of decision variables $\vec{x} \in R^d$ is said to be weak Pareto optimal solution if there does not exist another $\vec{x}' \in R^d$ such that $f_i(\vec{x}') < f_i(\vec{x})$ for at least one i.

The vector $\vec{x}$ corresponding to the solutions included in the Pareto optimal set are called non-dominated. The plot of the objective functions whose non-dominated vectors are in the Pareto optimal set is called the Pareto front. Fig. 5 shows a simple Pareto front of two conflicting objectives.

3.2. Multi-objective problem solving approaches

Broadly we can categorize the multi-objective problems into three types:

1. The conventional weighted-sum-approach: transforming the original multi-objective problem into a single-objective problem by using a weighted formula.
2. The lexicographical approach, where the objectives are ranked in order of priority.
3. The Pareto approach, which consists of finding as many non-dominated solution as possible and returning a set of non-dominated solution to user.

In a conventional weighted-sum-approach a multi-objective problem is transformed into a single-objective problem. Basically a numerical weight is assigned to each objective (evaluation criterion) and then the values of the weighted criteria combined into a single value by either adding or multiplying all the weighted criteria.

Thus, the fitness function of a given candidate solution can be measured by the following two types of formula:

$$f(\vec{x}) = \sum_{i=1}^{n} w_i . f_i(\vec{x}) \tag{2}$$

$$\text{or} \quad f(\vec{x}) = \prod_{i=1}^{n} f_i(\vec{x})^{w_i}, \tag{3}$$

where w_i, $i = 1, \ldots, n$, denotes the weight assigned to criteria $f_i(\vec{x})$ and n is the number of evaluation criteria.

This method is popular and more useable because of its simplicity. However, there are several drawbacks associated with this method. First the setting of the weights in these formulas is ad-hoc. Second, the problem with weights is that, once a formula with precise values of weights has been defined and given to a data mining algorithm it will be effectively trying to find the best model for that particular setting of weights, missing the opportunity to find other models that might be actually more interesting to the user, representing a better trade-off between different quality criteria. Third, weighted formulas involving a linear combination of different quality criteria can't give solutions in a non-convex region of the solution space.

In the lexicographic approach, different priorities are assigned to different objectives, and then the objectives are optimized in order of their priority. So when two or more candidate solutions are compared with each other to choose the best one, then their performance measure are compared for the highest-priority objective. If one candidate solution is significantly better than the other with respect to that objective, the former is chosen. Otherwise the performance measure of the two candidate solutions is

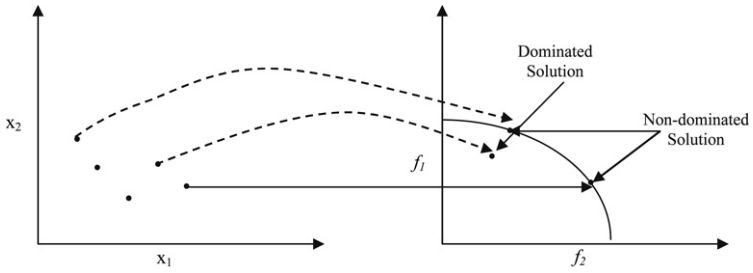

Fig. 4 – Mapping a decision variable from R^d into F^n.

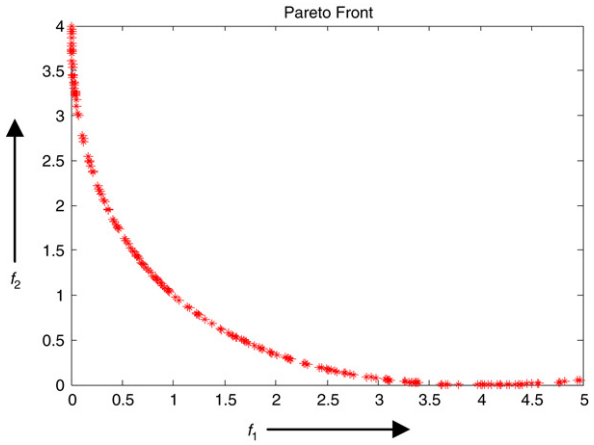

Fig. 5 – Pareto front of simple multi-objective problem.

compared with respect to the second objective. Again, if one candidate solution is significantly better than the other with respect to that objective, the former is chosen, otherwise the performance measure of the two candidate solutions is compared with respect to the third criterion. This process is repeated until one finds a clear winner or until one has used all the criteria. In the latter case, if there was no clear winner, one can simply select the solution optimizing the highest-priority objective.

The lexicographic approach has an advantage over conventional weighted approach, as it treats each of the criteria separately, recognizing that each criterion measures a different aspect of quality of a candidate solution. The disadvantage of this approach is how to specify a tolerance threshold for each criterion.

In Pareto approach, instead of transforming a multi-objective problem into a single-objective problem and then solving it by using a single-objective search method, a multi-objective algorithm is used to solve the original multi-objective problem. The concept is that the solution to a multi-objective optimization problem is normally not a single value but instead a set of values also called the "Pareto set". The proposed method for solving the classification task belongs to this category.

The disadvantage of Pareto approach is that in this approach it is very difficult to choose one best non-dominated solution from a set of non-dominated solutions, as in practice the user will use a single solution. As an advantage Pareto approach is more generic than the Minimum Description Length principle, since the latter is used only to cope with accuracy and simplicity, whereas the Pareto approach can cope with any kind of non-commensurable model quality.

3.3. *Multi-objective PSO*

3.3.1. *Basics of PSO*

Particle swarm optimization (PSO) is a population-based stochastic approach for optimization. It is modeled on the social behaviors observed in animals or insects e.g. bird flocking, fish schooling, and animal herding [63]. James Kennedy and Russell Eberhart originally proposed it in 1995 [64]. Since its inception, PSO has gained increasing popularity among researchers and practitioners as a robust and efficient technique for solving difficult optimization problems. In PSO, individual particles of a swarm represent potential solutions, which move through the problem search space seeking an optimal, or good enough solution. The particles broadcast their current positions to neighboring particles. The position of each particle is adjusted according to its velocity (i.e. rate of change) and the difference between its current position, respectively the best position found by its neighbors, and the best position it has found so far. As the model is iterated, the swarm focuses more and more on an area of the search space containing high-quality solutions.

PSO has close ties to artificial life models. Early works by Reynolds on a flocking model Boids [65], and Heppners studies on rules governing large numbers of birds flocking synchronously [66], indicated that the emergent group dynamics such as the bird flocking behavior are based on local interactions. These studies were the foundation for the subsequent development of PSO for the application to optimization. PSO is in some way similar to cellular automata (CA), which are often used for generating self replicating patterns based on very simple rules, e.g. John Conways Game of Life. CAs has three main attributes: (1) individual cells are updated in parallel; (2) the value of each new cell depends only on the old values of the cell and its neighbors; and (3) all cells are updated using the same rules [67]. Particles in a swarm are analogous to CA cells, whose states are updated in many dimensions simultaneously.

The social behavior of animals, and in some cases of humans, is governed by similar rules. However, human social

behavior is more complex than a flock's movement. Besides physical motion, humans adjust their beliefs, 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. In this context, it is the rule rather than exception that individuals/particles in plants and animal systems live within groups. Why is this? The reason is that many things that a group of individuals can do that an isolated individual cannot. This was the core idea behind the development of PSO.

3.3.2. PSO algorithm

In PSO, the velocity of each particle is modified iteratively by its personal best position (i.e. the best position found by the particle so far), and the best position found by particles in its neighborhood. As a result, each particle searches around a region defined by its personal best position and the best position from its neighborhood. Suppose that the search space is D-dimensional, the ith particle of the swarm can be represented by a D-dimensional vector, $x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$. The *velocity* (position change) of this particle, can be represented by another D-dimensional vector $v_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. The best previously visited position of the ith particle is denoted as $p_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$. Defining $p_g = \left(p_{g1}, p_{g2}, \ldots, p_{gD}\right)$ the best position found by particles in its neighborhood. The swarm is manipulated according to the following two equations:

$$v_{id}^{n+1} = w \times v_{id}^{n} + c_1 \times r_1^n \times \left(p_{id}^n - x_{id}^n\right) + c_2 \times r_2^n \times \left(p_{gd}^n - x_{id}^n\right) \quad (4)$$

$$x_{id}^{n+1} = x_{id}^n + v_{id}^{n+1} \quad (5)$$

where $d = 1, 2, \ldots, D$; $i = 1, 2, \ldots, N$, and N is the size of the swarm; c_1 and c_2 are called *cognitive* and *social* constant, w is called inertia; r_1, r_2 are random numbers, uniformly generated from [0, 1]; and $n = 1, 2, \ldots$, determines the iteration number. Eq. (4) shows that the velocity of a particle is determined by three parts "the momentum", "the cognitive", and "the social" part. The momentum term represents the product of inertia and previous velocity which is used to carry the particle in the direction it has traveled so far; the cognitive part, represents the tendency of the particle to return to the best position it has visited so far; the social part, represents the tendency of the particle to be attracted towards the position of the best position found by the entire swarm.

Position $p_g = \left(p_{g1}, p_{g2}, \ldots, p_{gD}\right)$ in the social part is the best position found by particles in the neighborhood of the ith particle. Different neighborhood topologies can be used to control information propagation between particles. Examples of neighborhood topologies include ring, star, and Von Neumann. Constricted information propagation as a result of using small neighborhood topologies such as Von Neumann has been shown to perform better on complex problems, whereas larger neighborhoods generally perform better on simpler problems [68]. Generally, a PSO implementation that chooses $p_g = \left(p_{g1}, p_{g2}, \ldots, p_{gD}\right)$ from within a restricted local neighborhood is referred to as l_{best} PSO, whereas choosing $p_g = \left(p_{g1}, p_{g2}, \ldots, p_{gD}\right)$ from the entire swarm results in a g_{best} PSO. The following algorithm summarizes a basic PSO, assume that the problem is a maximization one.

```
Algorithm_PSO( )
{
Randomly generate an initial swarm
Repeat
  {
    FOR each particle i
      IF f(x_i) > f(p_i) then
        p_i ← x_i
      End IF
    End FOR
    p_g = max(p_neighbors)
    Update velocity using Eq. (4)
    Update position using Eq. (5)
  } Until termination criterion is met.
}
```

Earlier studies showed that the velocity as defined in Eq. (4) has a tendency to explode to a large value, resulting in particles exceeding the boundaries of the search space. This is more likely to happen especially when a particle is far from p_g and p_i. To overcome this problem, a velocity clamping method can be adopted where the maximum allowed velocity value is set to $V_{\max}$ in each dimension of v_i. This method does not necessarily prevent particles neither from leaving the search space nor from converging. However, it does limit the particle step size, thereby preventing further divergence of particles.

3.3.3. PSO for multiple objectives

Until recently PSO had only been applied to single objective optimization problems, however, in a large number of design applications there are a number of competing quantitative measures that define the quality of a solution. For instance, designing aircrafts requires simultaneous optimization of fuel efficiency, payload, and weight. Similarly for bridge construction a good design characterized by low total mass and high stiffness. These objectives cannot be typically met by a single solution, so, by adjusting the various design parameters, the firm may seek to discover what possible combinations of these objectives are available, given a set of constraints. The relative simplicity of PSO and its population-based approach have made it a natural candidate to be extended for multi-objective optimization. Many different strategies for solving multi-objective problems using PSO has been published since 2002 [69–72]. However, although most of these studies were generated in tandem, each of these studies implements MOPSO in a different fashion. Given the wealth of multi-objective evolutionary algorithms (MOEAs) [73,74] in the literature this may not seem particularly surprising, however the PSO heuristics puts a number of constraints on MOPSO that MOEAs are not subject to. In PSO itself the swarm population is fixed in size, and its member cannot be replaced, only adjusted their p_{best} and g_{best}, which are themselves easy to define. However, in order to facilitate an MO approach to PSO a set of non-dominated solutions

(the best individuals found so far using the search process) must replace the single global best individual in the standard uni-objective PSO case. In addition, there may be no single previous best individual for each member of the swarm. Choosing both which g_{best} and p_{best} to direct a swarm member's flight therefore is not trivial in MOPSO.

In general, when solving a multi-objective problem using evolutionary or non-evolutionary techniques, the following three main goals need to be achieved: (i) maximize the number of elements of the Pareto optimal set found; (ii) minimize the distance of the Pareto front produced by the algorithm with respect to the true (global) Pareto front (assuming we know its location); (iii) maximize the spreads of solutions found, so that we can have a distributions of vectors as smooth and uniform as possible.

Based on the population nature of PSO, it is desirable to produce several (different) non-dominated solutions with a single run. So, as with any other evolutionary algorithm, the three main issues to be considered when using PSO to multi-objective optimization are: (i) how to select g_{best} particles in order to give preference to non-dominated solutions over those that are dominated? (ii) how to retain the non-dominated solutions found during the search process in order to report solutions that are non-dominated with respect to all the past populations and not only with respect to the current one? Also it is desirable that these solutions are well spread along the Pareto front; (iii) how to maintain diversity in the swarm in order to avoid convergence to a single solution?

As we could see in the previous section, when solving single-objective optimization problems, the g_{best} that each particle uses to update its position is completely determined once a neighborhood topology is established. However in the case of multi-objective optimizations problems, each particle might have a set of different g_{best}s from which just one can be selected in order to update its position. Such set of g_{best}s is usually stored in a different place from the swarm that we will call external archive denoted as EX_ARCHIVE. This is a repository in which the non-dominated solutions found so far are stored. The solutions contained in the external archive are used as global bests when the positions of the particles of the swarm have to be updated. Furthermore, the contents of the external archive are also usually reported as the final output of the algorithm. The following algorithm describes how a general MOPSO works.

Algorithm _MOPSO ( )
1. INITIALIZATION of the Swarm
2. EVALUATE the fitness of each particle of the swarm.
3. EX_ARCHIVE = SELECT the non-dominated solutions from the Swarm.
4. $t = 0$.
5. REPEAT
6. FOR each particle
7. SELECT the g_{best}
8. UPDATE the Position
9. MUTATION /* Optional */
10. EVALUATE the Particle
11. UPDATE the p_{best}
12. END FOR
13. UPDATE the EX_ARCHIVE with g_{best}s.
14. $t = t + 1$
15. UNTIL (t <= MAXIMUM_ITERATIONS)
16. Report Results in the EX_ARCHIVE.

First the swarm is initialized. Then a set of g_{best}s is also initialized with the non-dominated particles from the swarm. As we mentioned before, the set of g_{best}s is usually stored in an external archive, which we call EX_ARCHIVE. Later on, some sort of quality measure is calculated for all the g_{best}s in order to select (usually) one g_{best} for each particle of the swarm. At each generation, for each particle, a leader is selected and the flight is performed. Most of the existing MOPSOs apply some sort of mutation operator after performing the flight. Then the particle is evaluated and its corresponding p_{best} is updated. A new particle replaces its p_{best} particle usually when this particle is dominated or if both are incomparable (i.e. they are both non-dominated with respect to each other). After all the particles have been updated, the set of g_{best}s is updated, too. Finally, the quality measure of the set of g_{best}s is recalculated. This process is repeated for a certain number of iterations.

Let us discuss a few of the MOPSO algorithms developed by the different researchers over the years. We can identify the algorithms in five categories such as: (i) weighted sum approach; (ii) Lexicographic approach; (iii) sub-population approach; (iv) Pareto-based approaches; (v) combined approaches.

3.3.3.1. Weighted-sum approach. Under this category we consider approaches that combine (or "aggregate") all the objectives of the problem into a single one. In other words, the multi-objective problem is transformed into a single-objective one. Parsopoulos and Vrahatis [72] adopted three types of aggregating functions: (1) a conventional linear aggregating function (where weights are fixed during the run), (2) a dynamic aggregating function (where weights are gradually modified during the run) and (3) the bang bang weighted aggregation approach (were weights are abruptly modified during the run). Baumgartner et al. [75] approach, based on the *fully connected* topology, uses linear aggregating functions. Suresh et al. [76] presents a multi-agent particle swarm optimization to design an optimal composite box-beam helicopter rotor blade.

3.3.3.2. Lexicographic approach. In this method, the user is asked to rank the objectives in order of importance. The optimum solution is then obtained by minimizing the objective functions separately, starting with the most important one and proceeds according to the assigned order of importance of the objectives [77]. Lexicographic ordering tends to be useful only when few objective functions are used (two or three), and it may be sensitive to the ordering of the objectives [78]. The algorithm by Hu and Eberhart [71] optimizes only one objective at a time using a scheme similar to lexicographic ordering. This approach adopts the *ring* (*local best*) topology with no external archive.

3.3.3.3. Sub-population approaches. These approaches involve the use of several subpopulations as single-objective

optimizers. Then, the subpopulations somehow exchange information or recombine among themselves aiming to produce trade-offs among the different solutions previously generated for the objectives that were separately optimized. Parsopoulos et al. [79] suggested a parallel version of the Vector Evaluated Particle Swarm (VEPSO) method for multi-objective problems. VEPSO is a multi-swarm variant of PSO, which is inspired on the Vector Evaluated Genetic Algorithm (VEGA) [80, 81]. Chow and Tsui [82] used PSO as an autonomous agent response-learning algorithm. For that sake, the authors propose to decompose the award function of the autonomous agent into a set of local award functions and, in this way, to model the response extraction process as a multi-objective optimization problem. Koduru et al.'s. [83] proposed a PSO-Nelder Mead Simplex hybrid multi-objective optimization algorithm based on a numerical metric called ε-fuzzy dominance. Within each iteration of this approach, in addition to the position and velocity update of each particle using PSO, the k-means algorithm is applied to divide the population into smaller sized clusters. The Nelder–Mead Simplex algorithm is used separately within each cluster for added local search. Janson et al.'s. [84] used a multi-objective particle swarm optimization algorithm to simultaneously optimize the intramolecular energies occurring between the atoms of the flexible ligand and the macro-molecule. A clustering method is applied to form several sub-swarms with the PSO algorithm in order to thoroughly sample the search space. Xhang and Xue [85] proposed a dynamic sub-swarms multi-objective particle swarm optimization algorithm. Based on solution distribution of multi-objective optimization problems, it separates particles into multi sub-swarms, each of which adopts an improved clustering archiving technique, and operates PSO in a comparably independent way. Clustering eventually enhances the distribution quality of solutions. Omkar et al. [86] present a generic method/model for multi-objective design optimization of laminated composite components, based on vector evaluated particle swarm optimization (VEPSO) algorithm. In this work a modified version of VEPSO algorithm for discrete variables has been developed and implemented successfully for the, multi-objective design optimization of composites. Mostaghim et al. [87] proposed and empirically compare two parallel versions of multi-objective particle swarm optimization, which differs in the way they divide the swarm into sub-swarms that can be processed independently on different processors.

3.3.3.4. Pareto-based approaches. These approaches use leader selection techniques based on Pareto dominance. The basic idea of all the approaches considered here is to select as leaders to the particles that are non-dominated with respect to the swarm. Note however, that several variations of the leader selection scheme are possible since most authors adopt additional information to select leaders (e.g., information provided by a density estimator) in order to avoid a random selection of a leader from the current set of non-dominated solutions. Moore and Chapman [88] presented an algorithm in an unpublished document and it is based on Pareto dominance. The authors emphasize the importance of performing both an individual and a group search (a cognitive component and a social component). Ray and Liew [89] used Pareto dominance and combines concepts of evolutionary techniques with the particle swarm. Fieldsend and Singh [70] used an unconstrained elite external archive (in which a special data structure called "dominated tree" is adopted) to store the nondominated individuals found along the search process. The archive interacts with the primary population in order to define leaders. The selection of the g_{best} for a particle in the swarm is based on the structure defined by the dominated tree. First, a composite point of the tree is located based on dominance relations, and then the closest member (in objective function space) of the composite point is chosen as the leader. On the other hand, a set of personal best particles found (non-dominated) is also maintained for each swarm member, and the selection is performed uniformly. This approach also uses a "turbulence" operator that is basically a mutation operator that acts on the velocity value used by the PSO algorithm. Coello et al.'s. [69,90] proposal is based on the idea of having an external archive in which every particle will deposit its flight experiences after each flight cycle. The updates to the external archive are performed considering a geographically based system defined in terms of the objective function values of each particle. Toscano and Coello [91] used the concept of Pareto dominance to determine the flight direction of a particle. The authors adopt clustering techniques to divide the population of particles into several swarms. This aims to provide a better distribution of solutions in decision variable space. Srinivasan and Hou [92] proposed an approach, called Particle Swarm Inspired Evolutionary Algorithm (PS-EA), is a hybrid between PSO and an evolutionary algorithm. The main aim is to use EA operators (mutation, for example) to emulate the workings of PSO mechanisms, based on a *fully connected* topology. Mostaghim and Teich [93] proposed a *sigma* method in which the leader for each particle is selected in order to improve the convergence and diversity of a MOPSO approach.

In a further work, Mostaghim and Teich [94] studied the influence of ε-dominance [95] on MOPSO methods. ε-dominance is compared with existing clustering techniques for fixing the external archive size and the solutions are compared in terms of computational time, convergence and diversity. Mostaghim and Teich [96] proposed a new method called *covering* MOPSO (cvMOPSO). This method works in two phases. In phase 1, a MOPSO algorithm is run with an external archive with restricted size and the goal is to obtain a good approximation of the Pareto-front. In phase 2, the non-dominated solutions obtained from phase 1 are considered as the input external archive of the cvMOPSO. Bartz et al. [97] approach starts from the idea of introducing elitism (through the use of an external archive) into PSO. Different methods for selecting and deleting particles (leaders) from the archive are analyzed to generate a satisfactory approximation of the Pareto front.

Li's [98] approach is based on a *fully connected* topology and incorporates the main mechanisms of the NSGA-II [99] to the PSO algorithm. Reyes and Coello's [100] approach is based on Pareto dominance and the use of a nearest neighbor density estimator for the selection of leaders (by means of a binary tournament). This proposal uses two external archives: one for storing the leaders currently used for performing

the flight and another for storing the final solutions. The density estimator factor is used to filter out the list of leaders whenever the maximum limit imposed on such list is exceeded. Alvarez-Benitez et al. [101] propose methods based exclusively on Pareto dominance for selecting leaders from an unconstrained non-dominated (external) archive. Ho et al. [102] proposed a novel formula for updating velocity and position of particles, based on three main modifications to the known flight formula for the *fully connected* topology. Villalobos-Arias et al. [103] proposed a new mechanism to promote diversity in multi-objective optimization problems. Although the approach is independent of the search engine adopted, they incorporate it into the MOPSO proposed in [90]. The new approach is based on the use of stripes that are applied on the objective function space.

The main idea of Salazar–Lechuga and Rowe's [104] approach is to use PSO to guide the search with the help of niche counts (applied on objective function space) [105] to spread the particles along the Pareto front. Raquel and Naval [106] incorporates the concept of nearest neighbor density estimator for selecting the global best particle and also for deleting particles from the external archive of non-dominated solutions. Zhao and Cao [107] approach is very similar to the proposal of Coello and Lechuga [69]. However, the authors indicate that they maintain two external archives. One of them is actually a list that keeps the p_{best} particle for each member of the swarm. Another external archive stores the non-dominated solutions found along the evolutionary process. This truncated archive is similar to the adaptive grid of PAES [108].

Janson and Merkle [109] proposed a hybrid particle swarm optimization algorithm for multi-objective optimization, called ClustMPSO. ClustMPSO combines the PSO algorithm with clustering techniques to divide all particles into several subswarms.

Tsou et al.'s. [110] proposed an improved particle swarm Pareto optimizer in which the local search and clustering mechanism are incorporated. The local search mechanism prevents premature convergence, hence enhances the convergence of optimizer to true Pareto-optimal front. The clustering mechanism reduces the non-dominated solutions to a handful number such that the diversity can be maintained and speed up the search. Goldbarg et al. [111] presented a particle swarm optimization for the bi-objective degree-constrained minimum spanning tree problem. The operators for the particle's velocity are based upon local search and path relinking procedures.

Baltar and Fontane [112] used a multi-objective particle swarm optimization algorithm to find non-dominated (Pareto) solutions when minimizing deviations from outflow water quality targets of temperature, dissolved oxygen (DO), total dissolved solids, and potential of hydrogen (pH). Xu and Rahmat-Samii [113] studied the MOPSO proposed in [100] by applying it to two complex multi-objective electromagnetic problems: a 16-element array antenna synthesized for a tradeoff between the beam efficiency and the half-power beam width, and a single shaped reflector antenna optimized for higher gains of multiple feeds. Gill et al.'s. [114] proposed a new multi-objective particle swarm optimization by introducing the Pareto rank concept and is used for parameter estimation in hydrology. Liu et al.'s. [115] proposed a variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems. The proposed method combines the concept of variable neighborhood search (VNS) and particle swarm optimization (PSO). Niu and Shen [116] proposed an adaptive multi-objective particle swarm optimization and used to search the optimal color image fusion parameters, which can achieve the optimal fusion indices.

Koppen and Veenhuis [117] proposed a new multi-objective particle swarm optimization algorithm by using fuzzy-Pareto-dominance (FPD) relation. FPD can be seen as a paradigm or meta-heuristic to formally expand single objective optimization algorithms to multi-objective optimization algorithms. Tripathi et al.'s. [118] describe a novel time variant multi-objective particle swarm optimization (TV-MOPSO). TV-MOPSO is made adaptive in nature by allowing its vital parameters (viz., inertia weight and acceleration coefficients) to change with iterations. A new diversity parameter has been used to ensure sufficient diversity amongst the solutions of the non-dominated fronts, while retaining at the same time the convergence to the Pareto-optimal front. Reddy and Nagesh Kumar [119] proposed a MOPSO with a newly introduced variable size external repository and an efficient elitist-mutation (EM) operator. Rahimi-Vahed and Mirghorbani [120] adopted a multi-objective particle swarm optimization for solving flow shop scheduling problem. Exploiting a new concept of the Ideal point and a new approach to specify the superior particle's position vectors in the swarm is designed and used for finding locally Pareto-optimal frontier of the problem. Abido [121] introduced a two level of non-dominated solutions approach to multi-objective particle swarm optimization to handle the limitations associated with MOPSO.

3.3.3.5. Combined approaches. Mahfouf et al.'s. [122] proposed an Adaptive Weighted PSO (AWPSO) algorithm, in which the velocity is modified by including an acceleration term that increases as the number of iterations increases. This aims to enhance the global search ability at the end of the run and to help the algorithm to jump out of local optima.

Xiao-hua et al. [123] proposed an Intelligent Particle Swarm Optimization (IPSO) algorithm for multi-objective problems based on an Agent-Environment-Rules (AER) model to provide an appropriate selection pressure to propel the swarm population towards the Pareto optimal front. Nakamura et al.'s. [124] proposed a new multi-objective particle swarm optimization by incorporating design sensitivities concerning objective and constraint functions to apply to structural and mechanical design optimization problems.

4. Proposed method

Neural networks are computational models capable of learning through adjustments of topology/architecture and internal weight parameters according to a training algorithm in response to some training samples. Yao [125] described three common approaches to neural network training and these are: (1) for a neural network with a fixed architecture

nd a near-optimal set of connection weights; (2) nd a near optimal neural network architecture; and (3) simultaneously nd both a near optimal set of connection weights and neural network topology/architecture.

Recall that multi-objective optimization deals with simultaneous optimization of several possibly con icting objectives and generates the Pareto set. Each solution in the Pareto set represents a trade-off among the various parameters that optimize the given objectives. In supervised learning, model selection involves nding a good trade off between at least two objectives: predictive accuracy and architectural complexity. The usual approach is to formulate the problem in a single objective manner by taking a weighted sum of the objectives. Abbass [126] presented several reasons as to why this method is inef cient. Thus the multiple objective optimization (MOO) approach [127–129] is suitable since the architecture with connection weights and predictive accuracy can be determined concurrently and a Pareto set can be obtained in a single simulation from which user has the more exibility to choose a nal solution.

4.1. Problem formulation

In data mining the scalability is one of the important issues: the scalability of the architecture can be determined from the architectural complexity. In this article we are considering the architectural complexity of the polynomial neural network and it is based on the following parameter.

1. Input variables
2. Order of the polynomial
3. Number of layers of the polynomial
4. Number of PDs in a layer.

Let us assume that the number of input variables r remains constant for all layers and m be the dimension of the pattern vector. The order of the polynomial and number of layers is P and L respectively. Based on these assumptions we can calculate the total possible number of PDs in each of the layer of PNN.

1st Layer: $m_1 = \frac{mW}{rW - m - r/W}$.

2nd Layer: $m_2 = \frac{m_1 W}{rW - m_1 - r/W}$.

: : :

Lth Layer: $m_L = \frac{m_L W}{rW - m_L - r/W}$.

where L is the number of hidden layers.

Therefore, the total possible number of PDs (except input and output layer) in the architecture is

$$\begin{aligned} M &= m_1 + m_2 + \cdots + m_L \\ &= \sum_{i=1}^{L} m_i \\ &= \frac{1}{rW}\left(\frac{mW}{m - r/W} + \frac{m_1 W}{m_1 - r/W} + \cdots + \frac{m_L W}{m_L - r/W}\right). \end{aligned} \tag{6}$$

For simplicity, let us say m $\approx$ m_0 and then replacing m with m_0 in the Eq. (6), we get

$$M = \frac{1}{rW}\left(\sum_{j=0}^{L} \frac{m_j W}{m_j - r/W}\right). \tag{7}$$

As a result the architectural complexity can be de ned as follows:

$$\begin{aligned} f_{AC} &= f_{NV} + f_{NP} + f_{OP} + f_{NL} \\ &= r + M + P + L \\ &= r + P + L + \sum_{i=1}^{L} m_i \\ &= r + P + L + \left(\frac{1}{rW}\left\{\sum_{j=0}^{L} \frac{m_j W}{m_j - r/W}\right\}\right) \end{aligned} \tag{8}$$

where r, P, L, and m_0 are independent variables and $m_1, m_2, \ldots, m_L$ are dependent variables.

Hence, $f_{AC} = f_{AC}(r, P, L, m_0) = f_{AC1} + f_{AC2}$, where $f_{AC1} = r + P + L$ and $f_{AC2} = \frac{1}{rW}\sum_{j=0}^{L} \frac{m_j W}{m_j - r/W}$.

As the values of r, P and L are small so no need to optimize the function f_{AC1}. Therefore the only function we have to optimize is f_{AC2}.

Further, in this work we are generating only one hidden layer; but we are considering the input variables twice (i.e. in the input layer and hidden layer).

Therefore the Eq. (8) boils down to

$$f_{AC} = r + P + 1 + \frac{1}{rW}\frac{m_0 W}{m_0 - r/W}.$$

This implies that the value of $f_{AC1} = r + P + 1$ and $f_{AC2} = \frac{1}{rW}\frac{m_0 W}{m_0 - r/W}$.

It should be noted that as we are considering the input features in the hidden layer along with the PDs so the objective functions to measure the architectural complexity is

$$\begin{aligned} f_{AC} &= f_{AC2} + (m_0 + 1) \\ &= \frac{m_0 W}{rW}\frac{m_0 - 1/W}{m_0 - r/W} + rW + 1. \end{aligned} \tag{9}$$

The measure for predictive accuracy is de ned based on the concept of confusion matrix. Let the problem be a C class problem. Then the confusion matrix for the C class problem is de ned as follows.

Actual	Predicted		
	C_1	C_2	C_C
C_1	a_{11}	a_{12}	a_{1c}
C_2	a_{21}	a_{22}	a_{2c}
C_C	a_{c1}	a_{c2}	a_{cc}

The entries in the confusion matrix have the following meaning in the context of our study.

a_{11} is the number of correct predictions that an instance is C_1, a_{21} is the number of incorrect predictions that an instance is C_2, and so on.

Several standard terms have been de ned:

The Predictive accuracy is the proportion of total number of predictions that were correct. It is determined using Eq. (10)

$$f_A = \frac{\sum_{i=1}^{c} a_{ii}}{\sum_{i=1}^{c}\sum_{j=1}^{c} a_{ij}}. \tag{10}$$