Probabilistic Neural Network - parameters adjustment in classification task

Piotr A. Kowalski, *Senior Member IEEE* and Maciej Kusy *Member IEEE*
and Szymon Kubasiak and Szymon Łukasik *Member IEEE*

***Abstract*—This work presents a comparative analysis of probabilistic neural network training methods applied to achieve best performance in various classification tasks. Two result from classical mathematical methods based on the theory of kernel density estimators: the plug-in method and cross-validation procedure. The other two methods are more advanced: a metaheuristic algorithm of particle swarm optimization, and a procedure based on reinforcement learning. Ten data sets, regarded in eleven classification problems, taken from the UCI repository are used for the numerical analysis. A comparative analysis of probabilistic neural network learning methods leads to interesting conclusions. Although it does not allow for unambiguous selection of the best learning method, it provides a possibility of choosing a method that is adequate for the given conditions. The description of this is included in the work.**

***Index Terms*—probabilistic neural network, learning procedures, plug-in algorithm, cross-validation procedure, particle swarm optimization, reinforcement learning , prediction ability**

I. Introduction

Computational intelligence, being a subset of artificial intelligence, is one of the most dynamically developing fields of computer science. This is due to the need of analyzing larger and larger data sets, what in turn requires the application of more efficient computer equipment and the use of more advanced methods of analysis and data mining. The creation of artificial neural networks is one of the basic procedures in computational intelligence. These are structures inspired by biological neural networks that make up the brain and the nervous system of live animals. Structures of these types learn to perform tasks based on given examples, deemed a 'learning set'. The artificial neural network consists of interconnected units called 'artificial neurons'. As in the real world, the neuron receives a signal that can be modified in some way and then transmitted to other neurons connected to it.

A special case of the artificial neural network is the probabilistic neural network (PNN) that was proposed by Donald F. Specht [1]–[3]. This type of network is very much inspired by the idea behind kernel density estimators (KDE) of probability. It assumes that for each element in the training set, there is a probability density function of the set of attributes, the estimation of which allows determining the desired value.

Neural networks can be used in two issues: classification [4] and regression [5]. The classification task consists in the functional transformation of the object's attribute space into a set of labels for the classes under consideration, while the regression task allows for prediction of an unknown function value based on the character of elements from the training set. In each of the above cases, learning is a key issue in the context of neural networks, without which the network can generate partially or completely wrong or unrepresentative results. Moreover, various learning methods can provide different parameters in the neural network being selected which in turn may contribute to diverse results during network operation for the same set of input data. This difference can be significant, both in the context of solution correctness and computational efficiency of the neural algorithm. Therefore, choosing the right learning method is a very important issue.

There are many different methods and algorithms for parameter selection for the probabilistic neural network. Traditional methods result from the mathematical foundations of the theory of kernel estimators, however, newer methods are gaining in popularity. Among these are metaheuristics, which describe the methods of transition between solutions in order to find the optimal solution. Further methods are built upon reinforcement learning (RL), based on searching the solution space as built upon the system's response to the proposed change of parameter values.

Many research works do not mention the issue of applying PNN training algorithms. Other scientific articles focus on using only a single training method not considering the problem of the most suitable approach. This article is, therefore, a work aimed at comparing different methods of training a PNN for deriving solutions for the classification problem. The learning methods of plug-in algorithm, cross-validation procedure, particle swarm optimization metaheuristics (PSO), and reinforcement learning will be considered.

P.A. Kowalski is with Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Cracow, Poland, e-mail: pkowal@agh.edu.pl and Systems Research Institute, Polish Academy of Sciences, ul. Newelska 6, 01-447 Warsaw, Poland, e-mail: pakowal@ibspan.waw.pl.

M. Kusy is with the Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, al. Powstancow Warszawy 12, 35-959 Rzeszow, Poland, e-mail: mkusy@prz.edu.pl.

Szymon Kubasik is with Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Cracow, Poland

Szymon Łukasik is with Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Cracow, Poland, e-mail: slukasik@agh.edu.pl and Systems Research Institute, Polish Academy of Sciences, ul. Newelska 6, 01-447 Warsaw, Poland, e-mail: slukasik@ibspan.waw.pl.

This work was partly supported by Grant for Statutory Activity from: (i) Faculty of Physics and Applied Computer Science of the AGH University of Science and Technology in Cracow; (ii) Systems Research Institute, Polish Academy of Sciences; (iii) Department of Electronics Fundamentals, Rzeszow University of Technology, within the subsidy for maintaining research potential (UPB).

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

Despite the passage of time since the introduction of this type of neural networks, we have observed their very dynamic development. A PNN as data classifier finds common use in following tasks: image classification and recognition [6]–[8], earthquake magnitude prediction [9], multiple partial discharge sources classification [10], interval information processing [11], medical diagnosis and prediction [12]–[14] and phoneme recognition [15].

Moreover, in recent years many works have been published that are devoted to changes in the topology structure of the PNN network. In [4], [16], the authors propose the use of local sensitivity analysis to reduce the number of neurons in the input and pattern layers; in [17] a global sensitivity analysis is presented. A similar consideration can be observed in [18] and [19], with the difference that the main tool used for reduction is the clustering algorithm. A completely different task in exploratory data analysis is the processing of data in various types, but it should be clearly stated that this is not a trivial task. In the paper [20], authors propose to enrich the current set of vector features with interval type information, while in [21], a completely different task is considered - classification in a time-varying environment.

This article is composed as follows. Section II discusses topology and mathematical principals of probabilistic neural network. In section III, we present algorithms for learning the PNN process: plug-in algorithm, cross-validation procedure, PSO method and finally RL procedure. Section IV, provides results and discussion of numerical analysis. Finally, in Section V, we conclude our work.

II. Probabilistic Neural Network

In its basic form, the probabilistic neural network is a one-way, multi-layer network consisting of four layers. The first is the input layer, represented by the input data attributes; the next is the pattern layer that consists of as many neurons as there is data in the training set, which are divided into individual classes. The third layer is the summation layer, where there is one neuron for each of the considered classes; and at the end of the network there is the output layer, where a single neuron determines the class to which the investigated tested data belongs. Figure 1 depicts the graphical structure of a PNN as applied to solve a classification problem.

PNN is based on the concept of the probability density estimator. Consider, therefore, an n-dimensional random variable derived from a distribution with a probability density f. Its estimator is determined on the basis of an m-element random sample:

$$\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m. \tag{1}$$

The variables x_i are interpreted as the values obtained experimentally in the course of independent experiments in examining the value of the variable X. The kernel probability density estimator, in its basic form, is defined as:

$$\hat{f}(\mathbf{x}) = \frac{1}{mh^n} \sum_{j=1}^{m} K\left(\frac{\mathbf{x} - \mathbf{x}_j}{h}\right), \tag{2}$$

where m denotes random sample size, n is random variable dimension, h notes positive real number defined as smoothing parameter. The K function is called the kernel function. More about PNN and KDE can be found in [4], [22].

III. Probabilistic neural network training algorithms

The choice of the kernel function K form is one of the basic and most important issues that affect the quality of the estimator and hence PNN. Choosing the wrong form can significantly reduce the quality of the kernel estimator and, consequently, the neural network.

The most effective function of the kernel, in the sense of minimizing the Mean Integrated Squared Error (MISE), is the Epanecznik kernel. It should be noted that some methods for determining the value of the smoothing parameter require differentiation of the kernel function. This makes selection impossible. The functions fulfilling this condition are Gaussian, logistic and sigmoid functions. The last two are, however, the least effective of those presented [22]. Hence it was decided to use the Gaussian kernel function in the implemented neural network.

In the case when a multidimensional data set is considered, the methodology of PNN in our approach is generalized to the product kernel notation:

$$\begin{aligned} K(\mathbf{x}) = K(x_1, x_2, \ldots, x_N)^T = \\ \mathcal{K}(x_1) \cdot \mathcal{K}(x_2) \cdot \ldots \cdot \mathcal{K}(x_N), \end{aligned} \tag{3}$$

where $\mathcal{K}(x_i)$ denotes one dimensional kernel function.

Training a neural network is about choosing the right number of parameters that will allow the network to generalize. This is the property of generating the correct solution for the test data set, i.e. examples which did not participate in the learning process. Training a PNN involves selecting the appropriate number of smoothing parameters for individual kernel estimators in the pattern layer. The number of variables determined depends on the network model, the number of patterns and classes.

Figure 2 shows how important it is to select the smoothing parameter in the PNN learning process. This drawing reveals the operation of the KDE for three sample values of the smoothing parameter. It can easily be seen that only one curve represents the approximate probability density distribution. This is the curve for the parameter $h = 0.08$ (the orange line).

A. Training procedures

There are several methods for determining the values of smoothing parameters. In this work, four dominant methods in literature are compared. Two result from mathematical foundations which minimize the MISE, i.e. the plug-in algorithm [23] and cross-validation procedure [24]. Another method is the metaheuristic swarm algorithm Particle Swarm Optimization [25]. The last one is based on reinforcement learning [14]. Each method has its advantages, disadvantages and limitations. These algorithms are used under different

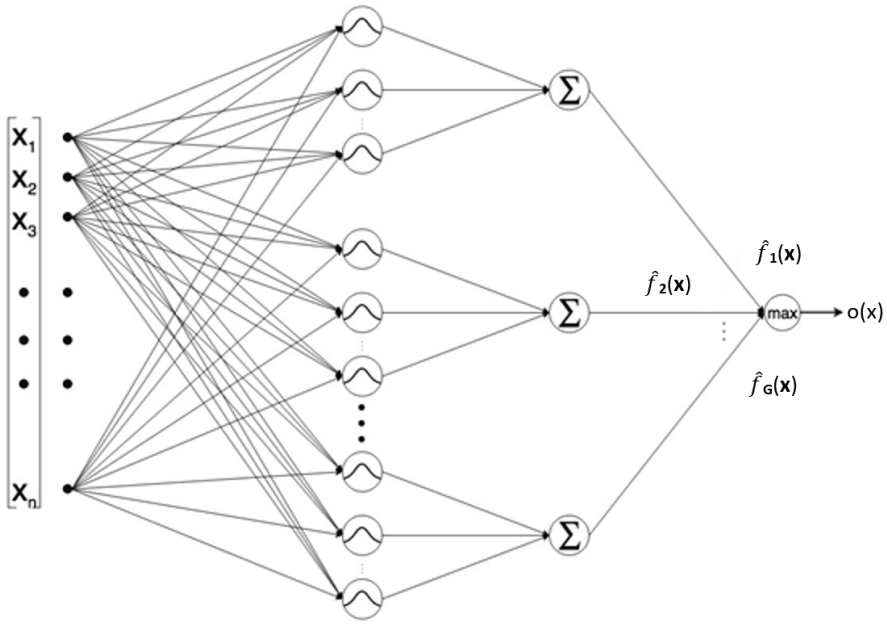

Fig. 1. Structure of a probabilistic neural network in the classification problem.

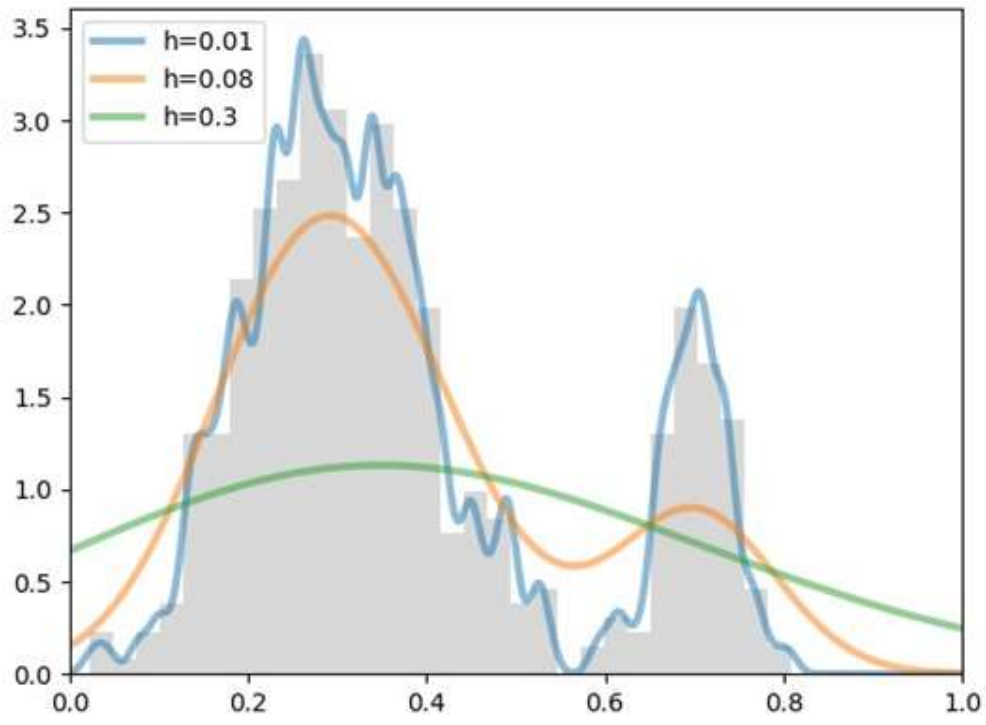

Fig. 2. Comparison of different values of the smoothing parameter for the probability density estimator.

assumptions, but they all have the same purpose. The selection of the appropriate parameter value is done by performing the task of minimizing the error value (cost) or maximizing the profit.

B. Types of probabilistic neural networks

There are four types of probabilistic neural network. They differ in the form of the artificial neuron smoothing parameter. The first of these, referred to as PNNS, has a scalar constant smoothing parameter for all neurons. This type is described by the way of equation (2).

The second type of probabilistic network is called PNNC. Here, the smoothing parameter is a vector with G dimensions $h_C = [h_{(1)}, h_{(2)}, ..., h_{(G)}]^T$, where G is the number of possible classes. This type of network is described by following equation:

$$\hat{f}(\mathbf{x}) = \frac{1}{m(h_{(g)})^n} \sum_{j=1}^{m} K\left(\frac{\mathbf{x} - \mathbf{x}_j}{h_{(g)}}\right). \tag{4}$$

The next type is denoted as PNNV. In this variant, the smoothing parameter is defined as a vector with n dimensions corresponding to a constant value for each attribute $h_V = [h_1, h_2, ..., h_n]$. So, in this case the fundamental equation (2) is extended in the following way:

$$\hat{f}(\mathbf{x}) = \frac{1}{m\prod_{i=1}^{n} h_i} \sum_{j=1}^{m} K\left(\frac{\mathbf{x} - \mathbf{x}_j}{h_V}\right). \tag{5}$$

The last and the most complex, and thus the most accurate type of probabilistic network, is PNNVC, which is a combination of the previous two. Herein, the matrix of smoothing parameters h with dimensions $G \times n$ is defined, and contains parameters corresponding to the attributes of the tested object separately for each class.

The selection of a network model is one of the basic tasks when designing a network. However, one of the implemented methods requires one dimensional data, which is a significant limitation that cannot be bypassed. In addition, determining a large number of variables requires immense computational resources, resulting in a significant extension of their selection time. This is not a desirable phenomenon, therefore, in this study, it was decided to choose the PNNV model as it meets the assumptions of all learning methods, and at the same time is not overly complicated.

C. Plug-in algorithm

The plug-in method is one of the basic methods for determining the smoothing parameter of the KDE. It results from the fact of minimizing the value of MISE. This method can only be used for one-dimensional cases. In this approach, the product form (3) of network model is chosen while the parameter for each dimension is determined separately.

The value minimizing the integrated mean square error is described by the formula:

$$h_0 = \left(\frac{W(K)}{U(K)^2 Z(f) m}\right)^{1/5}. \tag{6}$$

The values of $U(K)$ and $W(K)$ then characterize the kernel function K.

$$U(K) = \int_{-\infty}^{\infty} x^T x K(x) dx, \tag{7}$$

$$W(K) = \int_{-\infty}^{\infty} K(x)^2 dx, \tag{8}$$

while the value of $Z(f)$ relates to the estimated density of the function f:

$$Z(f) = \int_{-\infty}^{\infty} f''(x)^2 dx. \tag{9}$$

Deriving the value of the functional $Z(f)$ is the most difficult task due to required second derivative of the unknown probability density function. In the basic method, its kernel estimator is first constructed, and then it is necessary to determine the value of the smoothing parameter of the newly created estimator. This is done in an analogous way by constructing an estimator of the fourth derivative of the probability density function, and subsequently, a new estimator f^{2k} is determined. The approximate method is used to find the value of its smoothing parameter.

D. Cross-validation procedure

The cross-validation method, like the plug-in method, is based on the task of minimizing the MISE error value. Unlike the previous method, it does not place restrictions on the dimensionality or differentiation of kernel functions. The algorithm of cross-validation for finding the smoothing parameter of the kernel estimator is as follows. For the established form of the kernel, the following function is given:

$$g(h) = \frac{1}{h^n}\left[\frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\Gamma\left(\frac{\mathbf{x}_j - \mathbf{x}_i}{h}\right) + \frac{2}{m}K(0)\right], \tag{10}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ represent value of i-th and j-th object. The function $\Gamma(x)$ is determined by the equation:

$$\Gamma(x) = K^{*2}(x) - 2K(x), \tag{11}$$

where $K^{*2}(x)$ is the convolution square of the K function, defined as:

$$K^{*2}(x) = (4\pi)^{-n/2} exp(-\frac{1}{4}x^T x). \tag{12}$$

The appropriate value for the smoothing parameter is a value that minimizes the value of the function (10). Various methods of minimization can be used, while in this paper the method of the golden ratio is applied. This consists of gradually narrowing the range (a, b) with a value equal to:

$$q = \frac{\sqrt{5} - 1}{2}(b - a). \tag{13}$$

The value determined by the formula (13) is subsequently added to the upper or lower boundary from the previous

iteration depending on the values in the intermediate points, which are potential new boundaries. As with the substitution method, the cross-validation method is used to determine the smoothing parameter for each dimension separately.

E. Particle Swarm Optimization

Many algorithms and problem solving methods in computer science are inspired by natural processes. The same is true for the PSO algorithm. This is called the 'swarm algorithm', i.e. a metaheuristic algorithm inspired by swarm behavior [26]–[28]. The algorithm was proposed by James Kennedy and Russel C. Eberhart in [25].

Five basic principles can be distinguished, thanks to which modelling of artificial life can be called 'swarm intelligence': proximity - the population should store data in simple structures so that the calculations are not expensive; quality - the quality of solutions results from factors exclusively from the environment; diverse answers – the population should not focus on a limited set of behaviors; stability - the population should not change their behavior despite changes in the environment; adaptability - the population should be able to change their behavior when it proves to be beneficial to it.

The swarm optimization algorithm is an iterative method of optimizing solution value. The procedure seeks the optimum in an infinite space of solutions denoted as R^n of real numbers space. Function parameters can be understood as points in the real, multidimensional space of solutions, and their change as the agent's movement in this space. This is the basic idea of the PSO algorithm. The algorithm operates on a population of solutions instead of processing one solution. The number of particles is assumed, which, moving through the space of solutions iteratively, find better and better solutions to the optimization problem.

In the basic version, the swarm optimization algorithm consists of three steps: determining the quality for each of the particles, generating local and global updates of best solutions, and finally updating the speed and position of each of the particles. Each particle has the following attributes: (i) location in space solutions, (ii) a speed vector, (iii) the quality of the solution for the current position or the cost of this solution, (iv) best location and quality for this point, (v) the location of the point that gives the best solution to date of the entire swarm.

The particle speed is determined on the basis of few components. The most important is the inertia component, i.e. the velocity vector from the previous iteration that is consistent with the direction of the best current position found by the entire swarm and consistent with the direction of the best position found by this particle. In the proposed solution, each member of the swarm represents one vector h_V. During the PSO procedure, a calculation of the new vector h_V is generated. The quality of solution is achieved by the way of creating a neural network with new h classification being applied to a testing set.

F. Reinforcement learning algorithm

The purpose of the RL algorithm is to learn a policy which allows an agent to find an optimal solution for a given task. The learning process amounts to the interactions between the agent and the environment and may be realized using different approaches. The most popular approach relies on the update of the action value function $Q_t(s_t, a_t)$ in the iterative manner [29]:

$$Q_t(s_t, a_t) = Q_{t-1}(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q_{t-1}(s_{t+1}, a) - Q_{t-1}(s_t, a_t)\right), \quad (14)$$

where t is a discrete time step, $s_t \in S$ denotes the agent's state, $a_t \in A$ stands for an action chosen according to the policy of the agent, while r_t is the reinforcement signal. The algorithm that uses this type of action value function update is known as $Q(0)$–learning. Moreover, an enhanced version of this algorithm exists: $Q(\lambda)$–learning, where the computation of the action value function is enriched by the history of state–action pair activations. In both cases, the function Q is represented in table form.

The most important task in RL is to design the form of the environment that results in the choice of set of states S, set of actions A and the signal r_t. Additionally, the procedure of action value function update must be adjusted to a considered problem. In order to conduct the PNN training process, the environment must be conformed to the data classification problem with the use of PNN. In [14], it was assumed that the environment is composed of: (i) the data set in the form of pairs $\langle \mathbf{x}_j, t_j \rangle$ for $j = 1, \ldots, m$, where m is the data cardinality, (ii) the PNN classifier and (iii) the classification quality indicator.

In [14], the set of states S was defined by means of the classification error:

$$\mathcal{E} = \frac{1}{m}\sum_{j=1}^{m} \delta\left[o(\mathbf{x}_j) \neq t_j\right], \quad (15)$$

where $o(\mathbf{x}_j)$ is the PNN's output determined for $\mathbf{x}_j$, while $\delta[\cdot] = 1$, if $o(\mathbf{x}_j) \neq t_j$ and 0 otherwise. Thus, $S = \{0, 1/m, 2/m, \ldots, (m-1)/m, 1\}$. This solution has a natural interpretation since the state defined in such a way is the function of PNN output, which depends on the smoothing parameter.

The actions were defined as the values from a symmetric set $A = \{-a^{(1)}, -a^{(2)}, \ldots, -a^{(P)}, a^{(P)}, \ldots, a^{(2)}, a^{(1)}\}$. The elements $a^{(p)} \in A$ were directly used to modify the parameter h:

$$h_t = h_{t-1} + a_t. \quad (16)$$

This allows the smoothing parameter to increase or decrease its values in each PNN training step. In the article, a six element action set was proposed wherein $A = \{-1, -0.1, -0.01, 0.01, 0.1, 1\}$.

The basis of the action value function update is a temporal difference error, for which the reinforcement signal r is an integral part. The task for the signal r is to pass information

to the learning agent on the benefit from taking the action a_t in the state s_t. Since in this work, the core of the modification of the smoothing parameter is the minimization of the PNN's classification error (15), it is natural to propose the reinforcement signal in the way in which it rewards an agent when the training error decreases and punishes an agent when the error increases. The reinforcement signal is defined as:

$$r_t = \mathcal{E}_{t-1}(h_{t-1}) - \mathcal{E}_t(h_t), \tag{17}$$

where $\mathcal{E}_{t-1}(h_{t-1})$ and $\mathcal{E}_t(h_t)$ are the errors in the previous and current learning step. This not only makes this idea become satisfied, but, additionally takes into the account the dynamics of the changes of the classification error. Such a form of the reinforcement signal combined with the action value function update strengthens the confidence that the choice of an action is beneficial or not.

In order to perform the process of smoothing parameter update according to (16), $Q(0)$–learning, $Q(\lambda)$–learning and stateless Q–learning algorithms is proposed in [14]. The entire PNN training procedure can be summarized as follows. For each time step t:

1) choose action a_t by means of an actual policy derived from the action value function Q;
2) update the smoothing coefficient on the basis of formula (16);
3) compute the reinforcement signal r_t;
4) update the action value function Q.

G. Procedure for modification of smoothing parameter

In the basic algorithm of the PNN, the smoothing parameter value is constant for all objects in the training set. In the investigated network model (PNNV) it is a vector. Due to non optimal determination of parameter values or the presence of outlier elements in the sample on the basis of which the network is constructed, the determined values may cause erroneous or insufficiently accurate estimation of the probability density distribution.

The smoothing parameter modification algorithm is designed to individualize its impact on individual kernels. It consists in reducing the value in the areas of compaction of the objects of the set, which in turn enables the specific features of the estimated distribution to be better shown. However, in rare regions, the value of h is increased, which causes an additional smoothing of the distribution and reduces the sensitivity to outliers, which is a very favourable phenomenon. An additional advantage is the increase of the estimator's resistance to improper determination of the smoothing parameter. This feature is extremely beneficial due to the fact that the issue of selection of the smoothing parameter is a significant problem during the construction of the kernel estimator.

The algorithm used to modify the smoothing parameter looks as follows. Multiplicative parameters modifying $s_j > 0$ for $(j = 1, 2, \ldots, m)$ are first selected, such that:

$$s_j = \left(\frac{\hat{f}_*(\mathbf{x}_j)}{\tilde{s}} \right)^{-c}. \tag{18}$$

Here, the $\tilde{s}$ denotes the geometrical mean of the estimator quantities $\hat{f}_*(\mathbf{x}_j)$ for $j = 1, 2, \ldots, m$, while the c parameter is a constant denoting the intensity of the modification. Analysis of the mean square criterion indicates a suitable value of $c = 0.5$, which is why this value is used in the implemented version of the algorithm [23]. Finally, the kernel estimator is defined, in which the value of the smoothing parameter is modified:

$$\hat{f}(\mathbf{x}) = \frac{1}{m\det(\mathbf{h})} \sum_{j=1}^{m} \frac{1}{s_j^n} K\left(\frac{(\mathbf{x} - \mathbf{x}_j)^T \mathbf{h}^{-1}}{s_j} \right). \tag{19}$$

The estimator presented in equation (19) is an extension of its basic version (2), where the smoothing parameter is a fixed scalar value. In this work, the model (19) was applied.

IV. Results of numerical analysis

In the current study, the simulations are conducted to find solutions of the classification problems for the University of California, Irvine machine learning repository (UCI-MLR) data sets [30]. In order to compare the operation of a PNN trained with four different methods, ten data sets are used. Table I shows the characteristics of the sets: the dimensionality, the cardinality, the type and the domain of use.

In all classification problems, the results are assessed by comparing (i) the classification accuracy defined as the ratio of the number of correctly classified cases to all available examples and (ii) computational time (in seconds) necessary to provide the final solution.

The structure of a PNN is strongly dependent on the partition of the data into the learning and testing sets; therefore, for each of the sets, every method is trained and tested on exactly the same separation of the data into the above-mentioned parts. However, the relationship between the structure and the data division makes the comparison of these methods difficult due to the fact that a different division may result in selecting diverged values of smoothing parameters and therefore completely different accuracy outcomes than the ones presented here. Thus, the partition into training and testing sets takes place in such a way that 30% of the elements selected at random from the whole set constitute the test set. The rest of the data forms the training part which is fed to the learning algorithm.

Table II shows the accuracy of a PNN in all classification tasks computed on the test set. Based on such an indicator, it is easy to point out the method with the highest performance. The individual results shown in this table are the averages over 100 outcomes of the entire verification procedure.

As shown in numerical results, the classification quality, expressed in terms of testing accuracy, is similar for all implemented methods. However, if we consider the ranking based on the highest accuracy outcome (including the draw), the plug-in method achieves the best results in 6 out of 11 cases. The cross-validation and PSO methods perform worse since they provide the highest quality four times. The RL based PNN training method turns out to be least efficient with solely two best results.

TABLE I
CHARACTERISTIC OF THE DATA SETS.

Data set	Dimensionality	No. of data sample	Data type	Domain
Banknote Authentication	5	1372	real	computer science
Breast Cancer	10	699	integer	medicine
Glass	10	214	real	material
Indian Liver Patient Dataset	10	582	real and integer	medicine
Iris	4	150	real	biology
Seeds	7	210	real	biology
Statlog (Image Segmentation)	19	2310	real	computer science
Wine	13	178	real and integer	food
Wireless Indoor Localization	7	2000	real	computer science
Vertebral Column (a) 2 classes	6	310	real	medicine
Vertebral Column (b) 3 classes	6	310	real	medicine

TABLE II
COMPARISON OF ACCURACY SOLUTIONS IN THE CLASSIFICATION PROBLEM.

Data set	Plug-in	Cross-validation	PSO	Reinforcement
Banknote Authentication	1.00	1.00	0.99	0.99
Breast Cancer	0.59	0.62	0.94	0.95
Glass	0.45	0.73	0.64	0.53
Indian Liver Patient Dataset	0.74	0.74	0.69	0.71
Iris	1.00	1.00	1.00	0.97
Seeds	0.91	0.88	0.83	0.88
Statlog (Image Segmentation)	0.31	0.30	0.95	0.93
Wine	0.49	0.54	0.80	0.74
Wireless Indoor Localization	0.61	0.24	0.96	0.96
Vertebral Column (a) 2 classes	0.82	0.58	0.58	0.74
Vertebral Column (b) 3 classes	0.76	0.26	0.26	0.25

TABLE III
COMPARISON OF COMPUTATIONAL TIME EFFICIENCY OF SOLUTIONS IN THE CLASSIFICATION PROBLEM. THE EVALUATION TIMES ARE PRESENTED FOR THE ENTIRE TESTING PROCESS, I.E.: FOR 100 ITERATIONS.

Data set	Plug-in	Cross-validation	PSO	Reinforcement
Banknote Authentication	11.15s	40.75s	1001s	12393s
Breast Cancer	3.69s	19.68s	52043s	7361s
Glass	0.38s	2.39s	5557s	767s
Indian Liver Patient Dataset	1.91s	10.08s	14974s	790s
Iris	0.15s	0.62s	1311s	162s
Seeds	0.18s	0.87s	142724s	82.87s
Statlog (Image Segmentation)	32.8s	252s	420074s	21084s
Wine	0.35s	1.33s	5174s	728s
Wireless Indoor Localization	14.46s	62.99s	106880s	5710s
Vertebral Column (a) 2 classes	0.45s	1.65s	2536s	154s
Vertebral Column (b) 3 classes	0.46s	1.57s	2665s	119s

Comparing the PSO and RL based PNN training methods we can observe that the first one achieves a better performance when dealing with a limited number of examples. In the collections of a large number of records, there is no significant difference in the accuracy between the two. On the other hand, PNN trained with basic methods, i.e. the plug-in and the cross-validation, generates very similar results. However, it appears that the plug-in method allows obtaining a better quality when creating a model for the sets which are more difficult to classify: Seeds, Wireless Indoor Localization and Vertebral Column. A similar phenomenon is observed for PSO and RL methods. The swarm optimization procedure is a better choice for more difficult databases (such as Statlog, Glass or Wine).

In hard-to-classify collections (Wine and Glass), the choice of most reliable solution requires introducing some compromise, especially between PSO and RL methods. PSO works well when the set is not numerous, but characterized by a considerable dimensionality while RL method turns out to be a better solution in the opposite case.

Now, if one considers the computational time (Table III) as the efficiency indicator, the plug-in method is undoubtedly the best approach. However, we need to realize that a significant error is achieved by a PNN when this solution is adopted for network's training. It is also worth noting that PSO training method requires a longer time to complete the classification task than the RL based approach in 9 out 11 cases. This is especially important when both methods yield similar accuracy values. For example, PSO method provides the final solution for Seeds and Statlog data sets in 142 724 s and 420 074 s.

This is respectively: 1.7×10^3 and ≈ 20 times longer time than in the case when RL method is applied.

V. Conclusions

In this paper, four different PNN training methods were presented and compared. All of them relied on computing and optimizing the smoothing coefficient which is the network's kernel function parameter. Four various solutions were considered: the plugin method, the cross-validation, the particle swarm optimization and the reinforcement learning. For the purpose of the comparative analysis, ten UCI-MLR data sets were utilized. Despite the satisfactory results presented in terms of testing accuracy, it was not possible to select single best method. It was only possible to indicate a good choice under some criterion, e.g. data cardinality or dimensionality.

References

[1] D. F. Specht, "Probabilistic neural networks for classification, mapping, or associative memory," in *IEEE international conference on neural networks*, vol. 1, no. 24, 1988, pp. 525–532.

[2] ——, "Probabilistic neural networks," *Neural networks*, vol. 3, no. 1, pp. 109–118, 1990.

[3] ——, "Probabilistic neural networks and the polynomial adaline as complementary techniques for classification," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 111–121, 1990.

[4] P. A. Kowalski and M. Kusy, "Sensitivity analysis for probabilistic neural network structure reduction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 5, pp. 1919–1932, May 2018. doi: 10.1109/TNNLS.2017.2688482

[5] D. F. Specht *et al.*, "A general regression neural network," *IEEE transactions on neural networks*, vol. 2, no. 6, pp. 568–576, 1991.

[6] Y. Chtioui, S. Panigrahi, and R. Marsh, "Conjugate gradient and approximate newton methods for an optimal probabilistic neural network for food color classification," *Optical Engineering*, vol. 37, no. 11, pp. 3015–3023, 1998.

[7] S. Ramakrishnan and S. Selvan, "Image texture classification using wavelet based curve fitting and probabilistic neural network," *International Journal of Imaging Systems and Technology*, vol. 17, no. 4, pp. 266–275, 2007.

[8] X.-B. Wen, H. Zhang, X.-Q. Xu, and J.-J. Quan, "A new watermarking approach based on probabilistic neural network in wavelet domain," *Soft Computing*, vol. 13, no. 4, pp. 355–360, 2009.

[9] H. Adeli and A. Panakkat, "A probabilistic neural network for earthquake magnitude prediction," *Neural networks*, vol. 22, no. 7, pp. 1018–1024, 2009.

[10] S. Venkatesh and S. Gopal, "Orthogonal least square center selection technique–a robust scheme for multiple source partial discharge pattern recognition using radial basis probabilistic neural network," *Expert Systems with Applications*, vol. 38, no. 7, pp. 8978–8989, 2011.

[11] P. A. Kowalski and P. Kulczycki, "Data sample reduction for classification of interval information using neural network sensitivity analysis," in *Artificial Intelligence: Methodology, Systems, and Applications*, ser. Lecture Notes in Computer Science, D. Dicheva and D. Dochev, Eds. Springer Berlin Heidelberg, 2010, vol. 6304, pp. 271–272. ISBN 978-3-642-15430-0

[12] R. Folland, E. Hines, R. Dutta, P. Boilot, and D. Morgan, "Comparison of neural network predictors in the classification of tracheal–bronchial breath sounds by respiratory auscultation," *Artificial intelligence in medicine*, vol. 31, no. 3, pp. 211–220, 2004.

[13] D. Mantzaris, G. Anastassopoulos, and A. Adamopoulos, "Genetic algorithm pruning of probabilistic neural networks in medical disease estimation," *Neural Networks*, vol. 24, no. 8, pp. 831–835, 2011.

[14] M. Kusy and R. Zajdel, "Application of reinforcement learning algorithms for the adaptive computation of the smoothing parameter for probabilistic neural network," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 26, no. 9, pp. 2163–2175, Sept 2015.

[15] K. Elenius and H. G. Tråvén, "Multi-layer perceptrons and probabilistic neural networks for phoneme recognition." in *EUROSPEECH*, 1993.

[16] M. Kusy and P. A. Kowalski, "Weighted probabilistic neural network," *Information Sciences*, vol. 430–431, pp. 65–76, 2018.

[17] P. A. Kowalski and M. Kusy, "Determining the significance of features with the use of sobol' method in probabilistic neural network classification tasks," in *Federated Conference on Computer Science and Information Systems 2017 (FedCSIS 2017)*, ser. Annals of Computer Science and Information Systems, M. Ganzha, L. Maciaszek, and M. Paprzycki, Eds., vol. 11. Prague (Czech Republic): IEEE, September 2017. doi: 10.15439/2017F225 pp. 39–48. [Online]. Available: http://dx.doi.org/10.15439/2017F225

[18] Y. Chtioui, D. Bertrand, and D. Barba, "Reduction of the size of the learning data in a probabilistic neural network by hierarchical clustering. application to the discrimination of seeds by artificial vision," *Chemometrics and Intelligent Laboratory Systems*, vol. 35, no. 2, pp. 175–186, 1996.

[19] M. Kusy, "Fuzzy c-means-based architecture reduction of a probabilistic neural network," *Neural Networks*, vol. 108, pp. 20 – 32, 2018. doi: https://doi.org/10.1016/j.neunet.2018.07.012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608018302119

[20] P. A. Kowalski and P. Kulczycki, "Interval probabilistic neural network," *Neural Computing and Applications*, vol. 28, no. 4, pp. 817—-834, 2017. doi: 10.1007/s00521-015-2109-3. [Online]. Available: http://dx.doi.org/10.1007/s00521-015-2109-3

[21] L. Rutkowski, "Adaptive probabilistic neural networks for pattern classification in time-varying environment," *Neural Networks, IEEE Transactions on*, vol. 15, no. 4, pp. 811–827, July 2004.

[22] P. Kulczycki, "Kernel estimators in industrial applications," in *Soft Computing Applications in Industry*. Springer, 2008, pp. 69–91.

[23] M. P. Wand and M. C. Jones, *Kernel smoothing*. Chapman and Hall/CRC, 1994.

[24] B. W. Silverman, "Monographs on statistics and applied probability," *Density estimation for statistics and data analysis*, vol. 26, 1986.

[25] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[26] R. C. Eberhart, Y. Shi, and J. Kennedy, *Swarm intelligence*. Elsevier, 2001.

[27] X.-S. Yang, *Nature-inspired optimization algorithms*. Elsevier, 2014.

[28] P. A. Kowalski, S. L. ukasik, and P. Kulczycki, "Methods of collective intelligence in exploratory data analysis: A research survey," in *Proceedings of the International Conference on Computer Networks and Communication Technology (CNCT 2016)*, ser. Advances in Computer Science Research, P. A. Kowalski, S. L. ukasik, and P. Kulczycki, Eds., vol. 54. Xiamen (China): Atlantis Press, December 2016. doi: 10.2991/cnct-16.2017.1 pp. 1–7.

[29] J. Christopher, "Learning from delayed rewards," *PhD thesis, Cambridge University*, 1989.

[30] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml