Fuzzy Evolutionary Probabilistic Neural Networks

V.L. Georgiou*, Ph.D. Alevizos, and M.N. Vrahatis

Computational Intelligence Laboratory (CI Lab), Department of Mathematics, University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras, GR-26110 Patras, Greece
{vlg,philipos,vrahatis}@math.upatras.gr

Abstract. One of the most frequently used models for classification tasks is the Probabilistic Neural Network. Several improvements of the Probabilistic Neural Network have been proposed such as the Evolutionary Probabilistic Neural Network that employs the Particle Swarm Optimization stochastic algorithm for the proper selection of its spread (smoothing) parameters and the prior probabilities. To further improve its performance, a fuzzy class membership function has been incorporated for the weighting of its pattern layer neurons. For each neuron of the pattern layer, a fuzzy class membership weight is computed and it is multiplied to its output in order to magnify or decrease the neuron's signal when applicable. Moreover, a novel scheme for multi–class problems is proposed since the fuzzy membership function can be incorporated only in binary classification tasks. The proposed model is entitled Fuzzy Evolutionary Probabilistic Neural Network and is applied to several real-world benchmark problem with promising results.

1 Introduction

A rapid development of Computational Intelligence methods has taken place recently. A simple but promising model which combines statistical methods and efficient evolutionary algorithms is the recently proposed *Evolutionary Probabilistic Neural Network* (EPNN) [1,2]. Specifically, EPNN is based on the Probabilistic Neural Network (PNN) introduced by Specht [3] that has been widely used in several areas of science with promising results [4,5,6,7]. PNN is based on discriminant analysis [8] and incorporates the Bayes decision rule for the final classification of an unknown feature vector. In order to estimate the Probability Density Function (PDF) of each class, the Parzen window estimator or in other words the kernel density estimator is used [9]. The recently proposed EPNN employs the Particle Swarm Optimization (PSO) algorithm [10,11] for the selection of the spread parameters of PNN's kernels. Several other variants of PNN have been proposed in the literature. A Fuzzy PNN is proposed in [12], where a modification of the typical misclassification proportion is minimized in the training procedure.

* Corresponding author.

L. Prevost, S. Marinai, and F. Schwenker (Eds.): ANNPR 2008, LNAI 5064, pp. 113–124, 2008.

Several other remarkable efforts have taken place so that fuzzy logic [13,14] can be incorporated into well known and widely used classification models. Such an effort has been made in [15], where a Fuzzy Membership Function (FMF) has been introduced and incorporated into the Perceptron algorithm. Moreover, a Fuzzy Kernel Perceptron has been proposed in [16] in order to form a fuzzy decision boundary that separates two classes. The FMF that was employed in [16], is the one proposed by Keller and Hunt [15].

In this contribution an extension of the EPNN is proposed which incorporates the aforementioned Fuzzy Membership Function (FMF). This function describes the degree of certainty that a given datum belongs to each one of the predefined classes. The FMF provides a way of weighting all the training vectors so that an even better classification accuracy can be achieved.

2 Background Material

For completeness purposes, let us briefly present the necessary background material. As it has already been mentioned, PNN is used mainly for classification tasks. The training procedure of a PNN is quite simple and requires only a single pass of the patterns of the training data which results to a short training time. The architecture of a PNN always consists of four layers: the *input layer*, the *pattern layer*, the *summation layer* and the *output layer* [1,3].

Suppose that an input feature vector $\mathbf{X} \in \mathbb{R}^p$ has to be classified into one of K predefined classes. The vector $\mathbf{X}$ is applied to the p neurons of PNN's input layer and is then passed to the pattern layer. The neurons of the pattern layer are connected with all the input layers' neurons and are organized into K groups. Each group of neurons in the pattern layer consists of N_k neurons, where N_k is the number of training vectors that belong to the class k, $k = 1, 2, \ldots, K$. The ith neuron in the kth group of the pattern layer computes its output using a kernel function that is typically a Gaussian kernel function of the form:

$$f_{ik}(X) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}\left(X - X_{ik}\right)^T \Sigma_k^{-1}\left(X - X_{ik}\right)\right), \qquad (1)$$

where $\mathbf{X}_{ik} \in \mathbb{R}^p$ is the center of the kernel and Σ_k is the matrix of spread (smoothing) parameters of the kernel. The vector $\mathbf{X}_{ik}$ corresponds to the ith feature vector of the kth group of the training data set.

The summation layer comprises K neurons and each one estimates the conditional probability of its class given an unknown vector $\mathbf{X}$:

$$G_k(X) = \sum_{i=1}^{N_k} \pi_k f_{ik}(X), \quad k \in \{1, 2, \ldots, K\}, \qquad (2)$$

where π_k is the prior probability of class k, $\sum_{k=1}^{K} \pi_k = 1$. Thus, a vector $\mathbf{X}$ is classified to the class that has the maximum output of the summation neurons.

Instead of utilizing the whole training data set of size N and create a pattern layer that consists of N neurons, a smaller training set is created so that the

new PNN will have less memory requirements and will be much faster. The well-known K-medoids clustering algorithm [17] is applied to the training data of each class and K is set equal to 5% of the size of each class. Following this strategy, the proportion of instances of a class to the whole training data set remains the same. The adjacent training vectors are grouped as a cluster that is represented by the corresponding medoid. Then, the obtained medoids are used as centers to the corresponding PNN's kernel functions of the pattern layer's neurons. Thus, the pattern layer's size of the proposed PNN is about twenty times smaller than the corresponding PNN which utilizes all the available training data resulting to a much faster model.

For the estimation of the spread matrix Σ_k as well as the prior probabilities π_k, PSO algorithm is used. PSO is a stochastic population–based optimization algorithm [10] and it is based on the idea that a population of particles are released into a search space and travel with adaptable velocity in order to find promising regions into it [11,18]. Moreover, they retain a memory of the best position they have ever visited and at each step they intercommunicate to inform each other about their position and the value of the objective function at that particular point. The velocity of each particle is updated according to the particle's best value and the swarm best value.

Let $g(X)$ be the objective function that has to be minimized. Given a d–dimensional search space $\mathcal{S} \subset \mathbb{R}^d$ and a swarm consisting of NP particles, let $Z_i \in \mathcal{S}$ be the position of the ith particle and V_i be the velocity of this particle. Moreover, let BP_i be the best previous position encountered by the ith particle in $\mathcal{S}$. Assume gl to be the index of the particle that attained the best previous position among all particles, and t to be the iteration counter. Then, the swarm is manipulated by the equations

$$V_i(t+1) = \chi \Big[V_i(t) + c_1\, r_1 \big(BP_i(t) - Z_i(t) \big) + \; c_2\, r_2 \big(BP_{gl}(t) - Z_i(t) \big) \Big], \quad (3)$$

$$Z_i(t+1) = Z_i(t) + V_i(t+1), \quad (4)$$

where $i = 1, 2, \ldots, NP$; χ is a parameter called *constriction coefficient*; c_1 and c_2 are two positive constants called *cognitive* and *social* parameter, respectively; and r_1, r_2, are random vectors that are uniformly distributed within $[0,1]^d$ [19]. All vector operations in Eqs. (3) and (4) are computed component-wise and the best positions are then updated according to the equation

$$BP_i(t+1) = \begin{cases} Z_i(t+1), & \text{if } \; g\left(Z_i(t+1)\right) < g\left(BP_i(t)\right), \\ BP_i(t), & \text{otherwise.} \end{cases}$$

The particles are always bounded in the search space $\mathcal{S}$ and the constriction coefficient is derived analytically through the formula

$$\chi = \frac{2\kappa}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}, \quad (5)$$

for $\varphi > 4$, where $\varphi = c_1 + c_2$, and $\kappa = 1$, based on the stability analysis of Clerc and Kennedy [19,20].

A different matrix of spread parameters $\Sigma_k = \text{diag}(\sigma_{1k}^2, \ldots, \sigma_{pk}^2), k = 1, 2, \ldots, K$ is assumed for each class and a swarm of Σ_k created by PSO. The objective function that PSO should minimize is the misclassification proportion on the whole training data set.

3 The Proposed Approach

One of the desirable properties that a supervised classification model should possess is the ability to adjust the impact of each training sample vector to the final decision of the model. In other words, vectors of high uncertainty about their class membership should have less influence on the final decision of the model, while vectors of low uncertainty should affect more the model's decision. One way of obtaining this desirable property is to incorporate a Fuzzy Membership Function (FMF) into the model. Among the large variety of classification models we chose the EPNN due to its simplicity, effectiveness and efficiency [1,2] and we have incorporated the FMF proposed in [15] for weighting the pattern neurons of the EPNN. By this way, fuzzy class membership values are assigned to each pattern neuron and for this reason the proposed model is named *Fuzzy Evolutionary Probabilistic Neural Network* (FEPNN). The efficiency of the EPNNs and their variants is clearly presented in [2] where the EPNNs are compared with the best ever classification models on several problems.

Next, let us further analyze the proposed model. As it has already been mentioned in Section 2, $\mathbf{X}_{ik},\ i = 1, 2, \ldots, N_k, \quad k = 1, 2, \ldots, K$ is the i–th training sample vector that belongs to class k. Since we are dealing with a two-class classification problem, we consider $K = 2$. Suppose further that $u(\mathbf{X}) \in [0, 1]$ is a fuzzy membership function, then we define:

$$u(\mathbf{X}_{ik}) \equiv u_{ik} = 0.5 + \frac{\exp\left((-1)^k\ [d_1(\mathbf{X}_{ik}) - d_2(\mathbf{X}_{ik})]\ \lambda/d\right) - \exp(-\lambda)}{2\left(\exp(\lambda) - \exp(-\lambda)\right)},$$

where for $k = 1, 2$, $\mathbf{M}_k$ is the mean vector of class k, $d_k(\mathbf{X}) = \|\mathbf{X} - \mathbf{M}_k\|$ is the distance between vector $\mathbf{X}$ and mean vector of class k, $d = \|\mathbf{M}_1 - \mathbf{M}_2\|$ is the distance between the two mean vectors and λ is a constant that controls the rate at which fuzzy membership values decrease towards 0.5 [15]. The fuzzy membership values were designed so that if the vector is equal to the mean of the class that it belongs, then it should be 1.0. Also, if the vector is equal to the mean of the other class, it should be 0.5, meaning that this pattern neuron should not consider the most to the final decision. Moreover, if the vector is equidistant from the two means, then it should be near 0.5, since it cannot really help us to the final classification. In other words, as the vector gets closer to its mean and goes further away the other mean, its value should approach 1.0 exponentially. Moreover, the pseudocode of the proposed approach is presented in Table 1.

As it was previously mentioned, the proposed approach can be applied only to binary classification problems. In order to make it applicable to a wider spectrum of tasks, we propose a way of applying it to multi-class classification problems by using the following multi-class decomposition scheme. Assuming that $K > 2$, let

Create the clustered training set $\mathcal{T}_{cl\,tr}$ of size $N_{cl\,tr}$ from the training data set $\mathcal{T}_{tr}$.

Select initial random values for Σ_k and π_k, $k = 1, 2$.
Construct PNN using $\mathcal{T}_{cl\,tr}$, Σ_k and π_k.
Compute $\mathbf{M}_1$, $\mathbf{M}_2$ and d.
For $i = 1, N_{cl\,k}$ and $k = 1, 2$ do:
  Compute fuzzy membership values u_{ik} using Eq. (3).
EndFor

Compute Σ_k and π_k by PSO
For $l = 1$,NP do:
  Initialize a swarm
    $Z_l(0) = [\sigma_{1\,1\,l}, \sigma_{1\,2\,l}, \ldots, \sigma_{1\,p\,l}, \sigma_{2\,1\,l}, \sigma_{2\,2\,l}, \ldots, \sigma_{2\,p\,l}, \pi_{1\,l}, \pi_{2\,l}]$.
  Initialize best positions $BP_l(0)$.
EndFor
For t=1,MaxGeners do:
  For $l = 1$,NP do:
    Update velocities $V_l(t+1)$ using Eq. (3).
    Update particles $Z_l(t+1) = Z_l(t) + V_l(t+1)$.
    Constrain each particle $Z_l(t+1) \in (0, \gamma]^{2\,p} \times [0, 1]^2$.
    Set $MP_l = 0$. (Misclassification Proportion)
    For $m = 1, N_{tr}$ do:
      Compute $Out(m) = \arg\max_k \left(G_k(\mathbf{X}_m) = \pi_k \sum_{i=1}^{N_{cl\,k}} u_{ik}\, f_{ik}(\mathbf{X}_m) \right)$
      If $(Out(m) \neq Target(m))$ Then $MP_l = MP_l + 1$.
    EndFor
    Set $g(Z_l(t+1)) = MP_l / N_{tr}$
    Update best position $BP_l(t+1)$.
  EndFor
EndFor

Write the optimal Σ_k and Π and the classification accuracy of the PNN on $\mathcal{T}_{tr}$ and $\mathcal{T}_{te}$.

Fig. 1. Pseudocode of the proposed approach

OM be the overall mean vector of the whole data set and $\mathbf{OM}_k$ be the overall mean vector of the data set excluding the vectors of class k, $k = 1, 2, \ldots, K$. We calculate the Euclidean distances $D_k = \|\mathbf{OM} - \mathbf{M}_k\|$ and $D'_k = \|\mathbf{OM}_k - \mathbf{M}_k\|$ for all the classes and we sort the K classes according to their total distance $OD_k = D_k + D'_k$.

So, a sequence of $K - 1$ FEPNNs will be created for the final classification. Let s_k, $k = 1, 2, \ldots, K$ be the indices of the sorted classes. For the first FEPNN, we will utilize a training set consisting of the vectors of class s_1 as class 1 and the rest of the training set as class 2. Since this is a binary classification training set, we can use the proposed FEPNN in order to classify the "unknown" vectors of the test set that belong to class s_1. By this manner, we can record the number of correctly classified vectors of s_1 as C_{s_1}. This procedure is continued for the rest of the s_k, $k = 2, 3, \ldots, K - 1$ and at every step we do not take into account

the vectors of the classes s_i, $i < k$ and we compute the classification accuracies C_{s_k}. At $K-1$ step, we have only the last two classes left, so only one FEPNN is needed from which we compute $C_{s_{K-1}}$ and C_{s_K}. So the final classification accuracy is the sum of C_{s_k}, $k = 1, 2, \ldots, K$.

Several other multi-class decomposition schemes can be used such as the one–vs–others (1–vs–r) or the one–vs–one (1–vs–1) scheme. In the (1–vs–r) scheme, the problem is decomposed into a set of K two-class problems where for each class $k = 1, 2, \ldots, K$ a classifier is constructed that distinguishes between class k and the composite class consisting of all other classes. By this way, the training set always consists of all the exemplars of all classes while in our proposed scheme, in every step one class is excluded and only $K-1$ classifiers are constructed that results in a faster scheme. On the other hand, using the (1–vs–1) scheme also known as pairwise coupling where a classifier is constructed for each distinct pair of classes using only the training samples for those classes, $K(K-1)/2$ classifiers are constructed. This demands $K/2$ times more classifiers to be constructed compared to our proposed scheme although we should note that in our case the training data sets will be larger in the first steps.

4 Experimental Results

The proposed model has been applied to four binary and two multi–class benchmark problems from several fields of science from Proben1 database [21] that come from the UCI repository [22] in order to evaluate its efficiency and performance.

(**1**) The first data set is the *Wisconsin Breast Cancer Database* (WBCD) and the target of this problem is to predict whether a breast tumour is benign or malignant[23]. We have 699 instances and for each one of them we have 9 continuous attributes based on cell descriptions gathered by microscopic examination such as the uniformity of cell size and shape; bland chromatin; single epithelial cell size; and mitoses.

(**2**) In the second data set (*Card*), we want to predict the approval or non-approval of a credit card to a customer. There are 51 attributes which are unexplained for confidential reasons and 690 customers.

(**3**) The third data set is the *Pima Indians Diabetes* data set and the input features are the diastolic blood pressure; triceps skin fold thickness; plasma glucose concentration in a glucose tolerance test; and diabetes pedigree function. The 8 inputs are all continuous without missing values and there are 768 instances. The aim is to classify whether someone is infected by diabetes or not, therefore, there are two classes.

(**4**) In the last binary classification data set, namely *Heart Disease*, its aim is to predict whether at least one of the four major vessels of the heart is reduced in diameter by more than 50%. The 35 attributes of the 920 patients are age, sex, smoking habits, subjective patient pain descriptions and results of various medical examinations such as blood pressure and cardiogram.

(**5**) The fifth data set, *Glass*, consists of 214 instances and its aim is to classify a piece of glass into 6 different types, namely float processed or non float processed building windows, vehicle windows, containers, tableware and heat lamps. The classification is based on 9 inputs, which are the percentages of content on 8 different elements plus the refractive index and this task is motivated by forensic needs in criminal investigation.

(**6**) The last data set is the *Horse* data set and its task is to predict the fate of a horse that has a colic. The prediction whether the horse would survive, would die or would be euthanized is based on 58 inputs of a veterinary examination of the horse and there are 364 instances.

Moreover, the proposed model was applied to the aforementioned benchmark data sets using 10 times 10-fold cross-validation where the folds were randomly selected and the obtained results are presented in Tables 1 and 2. In order to

Table 1. Test set classification accuracy percentage of two-class data sets

Data set	Model	Mean	Median	SD	Min	Max
WBCD	PNN	95.79	95.85	0.25	95.27	96.14
	GGEE.PNN	96.39	96.42	0.20	95.99	96.71
	Hom.EPNN	95.82	95.85	0.28	95.28	96.28
	Het.EPNN	95.32	95.21	0.57	94.42	96.14
	Bag.EPNN	96.85	96.78	0.46	96.14	97.85
	Bag.P.EPNN	97.17	97.14	0.16	96.86	97.43
	FEPNN	**97.61**	**97.56**	0.19	97.42	97.85
Card	PNN	82.10	81.96	0.76	80.87	83.48
	GGEE.PNN	84.31	84.28	0.63	83.48	85.51
	Hom.EPNN	85.35	85.22	0.38	84.93	86.09
	Het.EPNN	**87.67**	**87.76**	0.51	86.96	88.55
	Bag.EPNN	86.64	86.67	0.51	85.80	87.39
	Bag.P.EPNN	86.83	86.81	0.34	86.38	87.39
	FEPNN	87.42	87.39	**0.28**	87.10	87.97
Diabetes	PNN	65.08	65.08	**0.05**	64.99	65.15
	GGEE.PNN	69.43	69.24	0.68	68.53	70.38
	Hom.EPNN	67.67	67.58	0.88	66.03	68.80
	Het.EPNN	69.37	69.46	0.80	67.73	70.54
	Bag.EPNN	71.00	71.16	1.02	68.90	72.09
	Bag.P.EPNN	71.22	71.39	1.00	69.75	72.54
	FEPNN	**75.09**	**75.39**	0.88	73.59	76.22
Heart	PNN	79.23	79.13	0.48	78.59	80.00
	GGEE.PNN	80.68	80.65	0.52	79.89	81.41
	Hom.EPNN	81.50	81.52	**0.27**	80.87	81.74
	Het.EPNN	82.60	82.45	0.40	82.07	83.26
	Bag.EPNN	82.28	82.34	0.62	81.20	83.15
	Bag.P.EPNN	82.35	82.50	1.05	80.43	84.13
	FEPNN	**83.01**	**82.94**	0.32	82.72	83.80

Table 2. Test set classification accuracy percentage of multi-class data sets

Data set	Model	Mean	Median	SD	Min	Max
Glass	PNN	33.25	32.61	3.40	27.96	39.01
	GGEE.PNN	50.07	50.08	**1.44**	47.74	51.94
	Hom.EPNN	68.52	68.15	1.55	66.80	70.78
	Het.EPNN	75.36	75.30	1.77	73.31	77.60
	Bag.EPNN	54.91	55.09	3.98	49.16	63.47
	Bag.P.EPNN	52.74	51.54	4.13	48.84	63.14
	Mult.EPNN	75.79	75.73	2.95	72.19	80.93
	Mult.FEPNN	**77.28**	**77.60**	2.74	71.09	81.73
Horse	PNN	64.63	64.74	**0.72**	63.05	65.42
	GGEE.PNN	61.97	62.39	1.23	59.83	63.75
	Hom.EPNN	66.54	66.74	0.79	65.33	67.55
	Het.EPNN	68.48	68.36	0.97	67.08	69.75
	Bag.EPNN	66.47	66.40	1.40	64.56	69.19
	Bag.P.EPNN	66.16	66.33	1.56	63.33	67.97
	Mult.EPNN	72.23	72.14	1.89	69.89	74.72
	Mult.FEPNN	**72.78**	**72.75**	1.78	70.19	75.19

eliminate the influence of PSO initialization phase, we conducted 5 runs on each cross-validated data set and selected the results (σ's and π's) that were obtained by the run on which the classification accuracy was the median of the classification accuracies on each training set. In particular, the mean, median, standard deviation, minimum and maximum classification accuracy on the test sets is presented in the aforementioned tables. Moreover, the CPU training times are also reported in Tables 3 and 4. In order to evaluate the performance of our model, we have applied these six benchmark problems to Homoscedastic and Heteroscedastic Evolutionary Probabilistic Neural Networks [1] as well as to original PNNs and Bagging EPNNs [2]. For the original PNN's implementation, an exhaustive search for the selection of the spread parameter σ has been conducted in the interval $[10^{-3},\ 5]$ and the σ that resulted to the best classification accuracy on the training set has been used for the calculation of PNN's classification accuracy on the test set. The number of functional evaluations for PNN's exhaustive search is the same with the one of EPNNs and FEPNNs. Moreover, a variation of the PNN that is proposed by Gorunescu *et al.* [24] has also been used by the name GGEE.PNN. In multi–class problems, the proposed approach that constructs a sequence of EPNNs or FEPNNs has been applied with and without the incorporation of the fuzzy membership function and is named Mult.FEPNN and Mult.EPNN respectively.

Searching for the most promising spread matrix Σ_k in EPNNs and FEPNNs, a swarm of 5 particles has been evolved for 50 generations for EPNN's homoscedastic case and a swarm of 10 particles for 100 generations for the other cases. The space that PSO was allowed to search in, was the aforementioned interval $[10^{-3},\ 5]$ for Hom.EPNN, $[10^{-3},\ 5]^K$ for Het.EPNN, $[10^{-3},\ 5]^{Kp}$ for Bagging EPNN and $[10^{-3},\ 5]^{2p}$ for FEPNN. On the Bagging EPNN case, an

Table 3. CPU time for the training of the models (seconds)

Data set	Model	Mean	Median	SD	Min	Max
WBCD	PNN	42.09	42.42	0.66	40.66	42.69
	GGEE.PNN	1.52	1.61	0.17	1.22	1.65
	Hom.EPNN	89.12	88.82	1.07	88.12	91.73
	Het.EPNN	171.78	171.75	1.07	170.21	174.04
	Bag.EPNN	82.78	78.07	8.86	76.22	99.75
	Bag.P.EPNN	90.01	89.86	0.92	88.97	92.12
	FEPNN	8.59	8.56	0.14	8.39	8.82
Card	PNN	182.01	186.37	7.88	169.82	187.93
	GGEE.PNN	5.46	5.45	0.06	5.38	5.53
	Hom.EPNN	266.10	274.39	74.56	168.72	342.27
	Het.EPNN	521.60	510.24	142.74	327.08	671.83
	Bag.EPNN	309.85	309.36	1.88	307.58	314.33
	Bag.P.EPN	309.73	309.84	2.62	305.26	314.95
	FEPNN	28.94	28.80	0.47	28.37	29.94
Diabetes	PNN	49.58	49.64	0.38	49.06	50.09
	GGEE.PNN	1.87	1.87	0.03	1.83	1.90
	Hom.EPNN	101.17	101.13	0.48	100.40	102.01
	Het.EPNN	195.27	195.66	0.92	193.82	196.62
	Bag.EPNN	106.42	106.53	0.92	104.25	107.73
	Bag.P.EPNN	106.24	106.26	0.81	105.31	108.06
	FEPNN	10.03	10.08	0.14	9.79	10.19
Heart	PNN	207.99	223.48	45.27	125.62	241.32
	GGEE.PNN	6.47	6.94	0.92	4.95	7.18
	Hom.EPNN	223.28	224.35	4.28	215.15	228.97
	Het.EPNN	438.10	440.29	6.82	422.45	449.24
	Bag.EPNN	394.49	392.36	5.93	387.13	404.55
	Bag.P.EPNN	393.22	391.47	4.95	388.02	401.03
	FEPNN	38.00	38.03	0.86	36.81	39.18

ensemble of 11 EPNNs was constructed. The value of the parameter f in the FMF was set to 0.5 after a trial-and-error procedure. In order to decide whether parametric or non parametric statistical tests should be conducted for the statistical comparison of the models' performance, a Kolmogorov-Smirnov test has been conducted on each sample of runs [25]. In all the samples, the normality assumption was met so a corrected resampled t–test was employed for the comparisons [26,27]. The level of significance in all the statistical tests was set to 0.05 and if a model's mean performance is statistically significantly superior than the second best performance, then it is depicted in a box. On the cancer data set, the best mean performance was achieved by the FEPNN and there was a statistically significant difference between its performance and the Bag.P.EPNN's performance which achieved the second best performance. Moreover, FEPNN obtained the lowest standard deviation of the classification accuracies. The best mean performance on the Card data set was obtained by the Het. EPNN but it was quite similar to the one that FEPNN obtained. However, FEPNN's standard

Table 4. CPU time for the training of the models (seconds)

Data set	Model	Mean	Median	SD	Min	Max
Glass	PNN	3.82	3.80	0.02	3.79	3.85
	GGEE.PNN	3.66	3.64	0.08	3.57	3.79
	Hom.EPNN	9.16	9.26	0.65	7.99	9.95
	Het.EPNN	17.21	17.47	0.76	16.04	18.27
	Bag.EPNN	29.03	29.03	0.14	28.80	29.19
	Bag.P.EPNN	28.30	28.31	0.11	28.10	28.48
	Mult.EPNN	6.02	6.08	0.31	5.40	6.41
	Mult.FEPNN	6.17	6.25	0.31	5.68	6.67
Horse	PNN	29.29	29.53	0.88	26.92	30.00
	GGEE.PNN	5.46	5.56	0.14	5.26	5.58
	Hom.EPNN	76.10	77.98	7.97	66.17	87.37
	Het.EPNN	169.92	169.92	23.39	147.73	192.11
	Bag.EPNN	126.76	126.84	2.02	124.68	129.96
	Bag.P.EPNN	123.03	121.61	2.24	121.35	126.92
	Mult.EPNN	17.61	17.69	0.74	16.50	18.65
	Mult.FEPNN	17.65	17.74	0.67	16.53	18.48

deviation was almost half of the Het.EPNN's. Besides that, the number of pattern layer's neurons of Het.EPNN was about 690 while in the FEPNN there were 34 neurons, which has as a result a much faster model both in training and response time as it is confirmed in Table 3.

On the diabetes data set, the statistically significant superiority of FEPNN is clear compared with the rest of the models. FEPNN's standard deviation is moreover similar to the rest of models' standard deviation except of the one obtained by the PNN which is much smaller but since there is such a great difference between the mean classification accuracies, it is not worth noting. On the heart data set, FEPNN obtained the best mean accuracy and there is a statistically significant difference with Het.EPNN's mean accuracy.

On the two multi–class problems, the proposed approach achieved the best performance and especially in Horse there was a statistically significant superiority than Het.EPNN. Summarizing the above, in five out of six cases the FEPNN had a superior performance and in four of them, the superiority was statistically significant.

Moreover, the proposed approach needs much less CPU training time than Bagging EPNNs and Het.EPNNs in all the benchmark problems as we can observe from Tables 3 and 4.

5 Concluding Remarks

In this contribution, a novel classification model has been proposed, namely the Fuzzy Evolutionary Probabilistic Neural Network that incorporates a fuzzy membership function for binary classification. A novel way of handling multi–class problems using binary classification models is also proposed.

It has been shown that the FEPNN can achieve similar or superior performance compared to other PNN variations both in binary and multi–class problems. Nevertheless, it is much faster in training and response times since it utilizes only a small fraction of the training data and achieves similar or superior accuracy. It is clear that the incorporation of the fuzzy membership function into Evolutionary Probabilistic Neural Network, helped it to obtain even more promising results.

The proposed approach is a general purpose method since it achieves promising results in classification problems on several areas of science either binary classification or multi-class classification problems.

Acknowledgment

We thank the European Social Fund (ESF), the Operational Program for Educational and Vocational Training II (EPEAEK II) and particularly the IRAKLEITOS Program for funding the above work.

References

1. Georgiou, V.L., Pavlidis, N.G., Parsopoulos, K.E., Alevizos, Ph.D., Vrahatis, M.N.: New self–adaptive probabilistic neural networks in bioinformatic and medical tasks. International Journal on Artificial Intelligence Tools 15(3), 371–396 (2006)
2. Georgiou, V.L., Alevizos, Ph.D., Vrahatis, M.N.: Novel approaches to probabilistic neural networks through bagging and evolutionary estimating of prior probabilities. Neural Processing Letters 27, 153–162 (2008)
3. Specht, D.F.: Probabilistic neural networks. Neural Networks 1(3), 109–118 (1990)
4. Ganchev, T., Tasoulis, D.K., Vrahatis, M.N., Fakotakis, N.: Locally recurrent probabilistic neural networks with application to speaker verification. GESTS International Transaction on Speech Science and Engineering 1(2), 1–13 (2004)
5. Ganchev, T., Tasoulis, D.K., Vrahatis, M.N., Fakotakis, N.: Generalized locally recurrent probabilistic neural networks with application to text-independent speaker verification. Neurocomputing 70(7–9), 1424–1438 (2007)
6. Guo, J., Lin, Y., Sun, Z.: A novel method for protein subcellular localization based on boosting and probabilistic neural network. In: Proceedings of the 2nd Asia-Pacific Bioinformatics Conference (APBC 2004), Dunedin, New Zealand, pp. 20–27 (2004)
7. Huang, C.J.: A performance analysis of cancer classification using feature extraction and probabilistic neural networks. In: Proceedings of the 7th Conference on Artificial Intelligence and Applications, Wufon, Taiwan, pp. 374–378 (2002)
8. Hand, J.D.: Kernel Discriminant Analysis. Research Studies Press, Chichester (1982)
9. Parzen, E.: On the estimation of a probability density function and mode. Annals of Mathematical Statistics 3, 1065–1076 (1962)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings IEEE International Conference on Neural Networks, Piscataway, NJ, vol. IV, pp. 1942–1948. IEEE Service Center, Los Alamitos (1995)

11. Parsopoulos, K.E., Vrahatis, M.N.: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing 1(2–3), 235–306 (2002)
12. Delgosha, F., Menhaj, M.B.: Fuzzy probabilistic neural networks: A practical approach to the implementation of bayesian classifier. In: Reusch, B. (ed.) Fuzzy Days 2001. LNCS, vol. 2206, pp. 76–85. Springer, Heidelberg (2001)
13. Zadeh, L.A.: Fuzzy sets. Information and Control 8(3), 338–353 (1965)
14. Zadeh, L.A.: Fuzzy logic. IEEE Computer 21(4), 83–93 (1988)
15. Keller, J.M., Hunt, D.J.: Incorporating fuzzy membership functions into the perceptron algorithm. IEEE Trans. Pattern Anal. Machine Intell. 7(6), 693–699 (1985)
16. Chen, J., Chen, C.: Fuzzy kernel perceptron. IEEE Transactions on Neural Networks 13(6), 1364–1373 (2002)
17. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley and Sons, New York (1990)
18. Parsopoulos, K.E., Vrahatis, M.N.: On the computation of all global minimizers through particle swarm optimization. IEEE Transactions on Evolutionary Computation 8(3), 211–224 (2004)
19. Clerc, M., Kennedy, J.: The particle swarm–explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6(1), 58–73 (2002)
20. Trelea, I.C.: The particle swarm optimization algorithm: Convergence analysis and parameter selection. Information Processing Letters 85, 317–325 (2003)
21. Prechelt, L.: Proben1: A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe (1994)
22. Newman, D.J., Hettich, S., Blake, C.L., Merz, C.J.: UCI repository of machine learning databases (1998)
23. Mangasarian, O.L., Wolberg, W.H.: Cancer diagnosis via linear programming. SIAM News 23, 1–18 (1990)
24. Gorunescu, F., Gorunescu, M., Revett, K., Ene, M.: A hybrid incremental/monte carlo searching technique for the smoothing parameter of probabilistic neural networks. In: Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2007, Cluj-Napoca, Romania, pp. 107–113 (2007)
25. Kanji, G.K.: 100 Statistical Tests. Sage Publications, Thousand Oaks (1999)
26. Bouckaert, R.R., Frank, E.: Evaluating the replicability of significance tests for comparing learning algorithms. In: Dai, H., Srikant, R., Zhang, C. (eds.) PAKDD 2004. LNCS (LNAI), vol. 3056, pp. 3–12. Springer, Heidelberg (2004)
27. Nadeau, C., Bengio, Y.: Inference for the generalization error. Machine Learning 52(3), 239–281 (2003)