ISSN 2007-9737

An Experimental Study of Evolutionary Product-Unit Neural Network Algorithm

Alain Guerrero-Enamorado, Daimerys Ceballos-Gastell

Universidad de las Ciencias Informáticas (UCI), La Habana, Cuba

{alaing,dceballo}@uci.cu

Abstract. This paper aims to obtain empirical information about the behavior of an Evolutionary Product-Unit Neural Network (EPUNN) in different scenarios. To achieve this, an extensive evaluation was conducted on 21 data sets for the classification task. Then, we evaluated EPUNN on eleven noisy data sets, on sixteen imbalanced data sets, and on ten missing values data sets. As a result of this evaluation process, we conclude that there does not exist a significant difference between EPUNN and the four algorithms assessed; the accuracy of EPUNN rapidly worsen in the presence of noise, so we do not recommend its utilization in noisy environments; we found a tendency to robustness in EPUNN while the imbalance ratio grows; finally, we can state that it is able to handle missing data, but in this kind of data, a significant performance deterioration was manifested. For future work, we recommend to assess the impact of irrelevant attributes on EPUNN performance. In addition, an extension of noisy data set evaluation would be opportune.

Keywords. Evolutionary Product-Unit Neural Network (EPUNN), missing values, imbalanced data, noisy data.

1 Introduction

A simple method to classify patterns provides the probability of class membership based on evaluating linear functions on a set of predictive variables. However, quite often, in real-world classification problems, we cannot assume linearity in input variables. Specifically, in this paper we analyze an algorithm that avoids the effects of non-linearity of the input variables. Using an approach based on non-linear functions constructed with the product of the inputs raises to arbitrary powers. The exponents are real values and can be adjusted by machine learning. These functions can discover relations between predictive variables. The Product-Unit based Neural Networks (PUNN) were introduced by Durbin and Rumelhart in [9]. They are an alternative to sigmoidal neural networks and are based on multiplicative nodes instead of additive ones. Their training is more difficult than the training of standard sigmoidal-based networks. The cause is the existence of multiple local optima and plateaus in the error surface. The main reason for this difficulty is that small changes in the exponents can cause large changes in the total error surface. The complexity of the error surface associated with PUNN justifies the use of an evolutionary algorithm to design the topology of the network and to train its corresponding weights. In this case PUNN becomes EPUNN. This novel approach has been subjected to few comparisons in different scopes since its invention by Martínez-Estudillo et al. [16]. For this reason we compared the Evolutionary Product-Unit Neural Network Classifier (EPUNN) with some of the top ten [26] classifiers: NB, SVM, KNN, and C4.5 in four different comparative scenarios: noisy data, imbalanced data, missing values data sets, and classical data sets. In section 2 we describe in detail the main characteristics of the EPUNN classifier and in section 3 we describe the remaining algorithms.

With this work, we have made the following contributions: we validated experimentally (in 21 classical data sets) that EPUNN behaves similarly to four of the top ten algorithms; we showed that its performance is greatly reduced (exponentially), when the level of noise present in the data sets

increases; we found that its performance is better than the performance of the others algorithms on imbalanced data sets; finally, EPUNN decreases its performance when executed on data sets with missing values (see details about the data sets in section 4.1).

The methodology for the evaluation can be consulted in section 4, the tools are presented in section 4.2, and the statistical analysis is given in section 4.3.

2 Evolutionary Product-Unit Neural Networks Classifiers

The method consists of obtaining the neural network architecture and simultaneously estimating the weights of the model coefficients with an algorithm of evolutionary computation. A cross-entropy error function is used in the neural model. In this way a neuro-evolutive model is obtained from the training set and then checked against the patterns of the testing set. Some advantages of Product-Unit Neural Networks (PUNN) are their increased information capacity and the ability to form higher-order combinations of inputs. In the early work of Durbin and Rumelhart [9] it was determined empirically that the information capacity of product units (learning random boolean patterns) is approximately 3N, compared to 2N of a network with additive units for a single threshold logic function, where N denotes the number of inputs to the network. Despite these advantages, product-unit based networks have a major drawback: they have more local optima and more probability of becoming trapped in them [12]. The main reason for this difficulty is that small changes in the exponents can cause large changes in the total error surface and therefore their training is more difficult than the training of standard sigmoidal based networks. Several efforts have been made to carry out learning methods for product units [12, 10]. Studies carried out on PUNN have not tackled the problem of designing the structure and weights simultaneously in this kind of neural network, either using classic or evolutionary methods.

In this type of algorithm it is not possible to work with inputs that have negative values. Because weights are often non-integer values, therefore there would be roots of negative numbers which result in complex numbers. Since neural networks with complex outputs are rarely used in applications, Durbin and Rumelhart [9] suggested discarding the imaginary part and using only the real component for further processing but this manipulation would have disastrous consequences. To avoid this problem, the input domain is restricted. We define the input set by

$$\{x = (x_1, x_2, \ldots, x_n) \in \Re^n : x_i > 0, i = 1, 2, \ldots, n\}$$

In [17, 16] a neural architecture was proposed shown in Figure 1, an input layer with n nodes, a hidden layer with m nodes, and an output layer with l nodes, one for each class. The activation function of the j-th node in the hidden layer, h_j, is given by equation 1 where w_{ji} is the weight of the connection between input node i and hidden node j:

$$h_j = \prod_{i=1}^{n} x_i^{w_{ji}}, \quad j = 1, 2, \ldots, m. \tag{1}$$

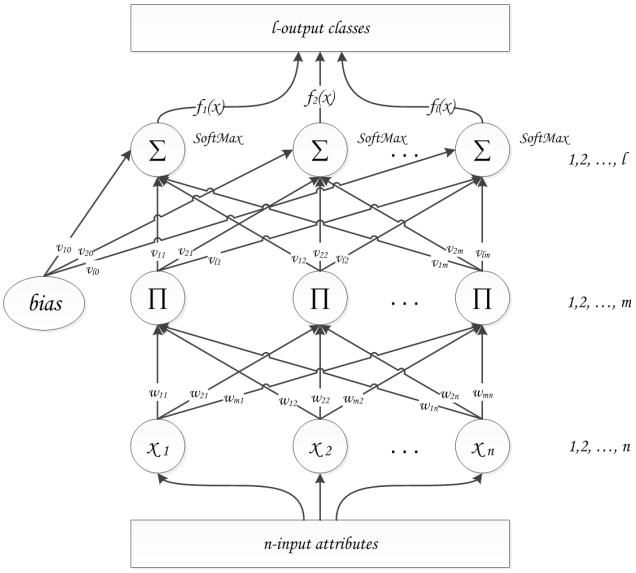

Fig. 1. Product-unit neural network

The activation function of the k-th node in the output layer, g_k, is given by equation 2 where v_{kj} is the weight of the connection between hidden node

j and output node k:

$$g_k = \sum_{j=1}^{m} v_{kj} * h_j + v_{k0} * 1, \quad k = 1, 2, \ldots, l. \quad (2)$$

By default the transfer function of all hidden and output nodes is the identity function. In this way, the estimated function g_k from each output $k = 1, 2, \ldots, l$, is given by the equation 3:

$$g_k(x) = \sum_{j=1}^{m} v_{kj} * \left(\prod_{i=1}^{n} x_i^{w_{ji}} \right) + v_{k0} * 1. \quad (3)$$

2.1 Evolutionary PUNN

The authors of [17, 16] applied an evolutionary neural network algorithm to learn the weights that minimize the cross-entropy error function and design the structure of PUNN. The search begins with an initial population of PUNN, and in each iteration the population is updated using a population-update algorithm. The population is evolved by replication and mutation. The authors exclude the crossover operator due to its potential disadvantages suggested by [27, 18] in evolving artificial networks. The pseudo-code of an Evolutionary Product-Unit Neural Network (EPUNN) is shown in (Algorithm 1).

2.2 Classification with EPUNN

A classification problem starts with feature measurements $x_i, i = 1, 2, \ldots, n$ for any single individual (or object), then the individuals should be classified into one of the l classes based on these feature measurements. A training sample $D = \{(x_t, y_t); t = 1, 2, \ldots, T\}$ is available, where $x_t = x_{1t}, x_{2t}, \ldots, x_{nt}$ is the vector of feature measurements taking values in $\psi \subset \Re^n$, and y_t is the class value of the t-th individual. Based on the training sample we intend to find a decision function $C : \psi \rightarrow \{1, 2, \ldots, l\}$ for classifying individuals. A misclassification occurs when a decision rule ψ assigns an individual (based on the feature measurement vector) to a class k when it actually comes from a class $q \neq k$, where $k, q = 1, 2, \ldots, l$. The authors of EPUNN defined the corrected classification rate (CCR) by $CCR = \frac{1}{T} \sum_{t=1}^{T} I\left(C\left(x_t\right) = y_t\right)$, where $I(\cdot)$ is the zero-one loss function. A good classifier tries to achieve the highest possible CCR in a given problem, for this, they consider the SoftMax activation function [4] in the output layer, obtaining the equation 4:

$$f(x, k) = \frac{\exp(g_k(x))}{\sum_{k=1}^{l} \exp\left(g_k(x)\right)}, \quad k = 1, 2, \ldots, l. \quad (4)$$

Observe that SoftMax transformation produces positive estimates that sum to one and, therefore, the outputs can be interpreted as the conditional probability of class membership. In general, the parameters needed for operation of the algorithm are given in the next subsection 2.3.

Algorithm 1: EPUNN, pseudo-code

Result: Classifier

Generate a random population of size N_p;

repeat

  Calculate the fitness of every individual in the population;

  Rank the individuals with respect to their fitness;

  The best individual is copied into the new population;

  The best 10% of population individuals are replicated and substitute the worst 10% of individuals;

  Apply parametric mutation to the best 10% of individuals;

  Apply structural mutation to the remaining 90% of individuals;

until $(numGeneration \geq maxGeneration || \, fitness\,(10\%bestInd)_{variance} \leq 10^{-4})$;

2.3 EPUNN Parameters

This model must be configured in KEEL [1] [2] with the following parameters:

— Hidden nodes: define the number of neurons in the hidden layer.

[1] Knowledge Extraction based on Evolutionary Learning. http://www.keel.es

— Transfer: define the transfer function in each neuron of the hidden layer (the input uses the identity transfer function).

— Generations: define the maximum number of generations in the evolutionary algorithm.

2.4 EPUNN Properties

This model was designed for operation under the following conditions:

— Continuous Variables: true.

— Nominal Variables: true.

— Discretized Variables: true.

— Integer Variables: true.

— Variables without values for some examples: true[2]

— Variables with imprecise values for some examples: false.

3 Related Work

In this article, we compared the competence of the EPUNN with four widely-known techniques for classification task. More specifically, we compared it with C4.5 [20], NB [15, 13, 8], KNN [11, 6, 24], and SVM [5]. The KEEL platform was used for all of them.

3.1 C4.5

This algorithm induces classification rules in the form of decision trees from a set of given examples. The decision trees are constructed top-down. In each step a test for the actual node is chosen (starting with the root node), which best separates the given examples into classes. C45 is an evolution of ID3 algorithm [19]. The extensions or improvements of ID3 are that it accounts for unavailable or missing values in data, it handles continuous attribute value ranges, chooses an appropriate attribute selection measure (maximizing information gain), and it prunes the result decision trees with minimal description length principle [21].

[2]KEEL reports this property false, however, experiment #4 shows that this algorithm is capable of handling missing values, although the accuracy deteriorates significantly.

3.1.1 C4.5 Parameters

This model must be configured in KEEL with the following parameters:

— Prune: allows activating or deactivating the pruning mechanism of the tree.

— Confidence: defines the minimal confidence that a leaf must have in order to be considered in the tree.

— minItemsets: defines the minimum number of instances per leaf. It is an integer value that determines how many data instances must be contained in a leaf in order for the leaf to be created.

3.1.2 C4.5 Properties

This model was designed for operation under the following conditions:

— Continuous Variables: true.

— Nominal Variables: true.

— Discretized Variables: true.

— Integer Variables: true.

— Variables without values for some examples: true.

— Variables with imprecise values for some examples: false.

3.2 NB

The NB classifier is based on Bayes' theorem, assuming independence between predictor attributes. The conditional probability of every example to be in a class is computed. Then the output class of the example will be assigned as the highest conditional probability calculated. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large data sets. Despite its simplicity, the NB classifier often performs surprisingly well (even when it is not known if there is independence between predictor attributes) and it is widely used because it often outperforms more sophisticated classification methods.

3.2.1 NB Parameters

This model must be configured in KEEL without parameters; however, we utilized uniform frequency discretization [14] prepossessing to convert numerical attributes (real and integers) to nominal ones.

3.2.2 NB Properties

This model was designed for operation under the following conditions:

— Continuous Variables: false.

— Nominal Variables: true.

— Discretized Variables: true.

— Integer Variables: false.

— Variables without values for some examples: false.

— Variables with imprecise values for some examples: false.

3.3 KNN

This is a supervised classification method that permits to estimate the density function of predictive attributes for each class. This estimation is obtained from the information provided by a set of k nearest neighbors. One point in the space is assigned to the class-C if this is the most frequent class between the k nearest examples in the training set. A special case is $k = 1$, in this case the algorithm is known as the Nearest Neighbor Algorithm [6]. The Euclidean distance is commonly used as a distance metric.

3.3.1 KNN Parameters

This model must be configured in KEEL with the following parameters:

— K: the number of neighbors to be tested. If this value is too high (similar to the data size), then it is the majority class classifier. When k=1, then it is the nearest neighbor algorithm.

— Distance Function: KEEL-KNN implements tree distance functions:

  – Euclidean, with normalized attributes.

  – Heterogeneous Value Difference Metric (HVDM) [25]; this distance function uses the Euclidean distance for quantitative attributes and the VDM distance [22] for qualitative attributes. The VDM metric considers the classification similarity for each possible value of a qualitative attribute to calculate the distances between these values.

  – Manhattan, the distance between two points is the sum of the absolute differences of their coordinates.

3.3.2 KNN Properties

This model was designed for operation under the following conditions:

— Continuous Variables: true.

— Nominal Variables: true.

— Discretized Variables: true.

— Integer Variables: true.

— Variables without values for some examples: true.

— Variables with imprecise values for some examples: false.

3.4 SVM

An SVM performs classification by finding the hyperplane that maximizes the margin between two classes. The vectors (instances) that define the hyperplane are the support vectors. The algorithm begins with defining an optimal hyperplane (maximizing the margin). Then the data is mapped to a high dimensional space by means of a Kernel function, where it is easier to classify with linear decision surfaces: the problem is reformulated so that data is mapped implicitly to this space. In the ideal case SVM should produce a hyperplane that completely separates the instances into two non-overlapping classes. However, perfect separation may not be possible, or it may result in a model with so many cases that the model does not classify correctly. Hence SVM finds the hyperplane that maximizes the margin and minimizes the misclassifications. This method is also apt to classify problems with more than two classes with a voting scheme.

3.4.1 SVM Parameters

This model must be configured in KEEL with the following parameters:

— KernelType: which kernel will be used to transform the data.

— C: cost; it is the penalty parameter of the error term.

— Degree: sets degree in kernel function.

— Gamma: sets gamma in kernel function.

— Coef0: sets coef0 in kernel function.

— Shrinking: reduces the size of the optimization problem without considering some bounded variables. The decomposition method then works on a smaller problem which is less time-consuming and requires less memory.

3.4.2 SVM Properties

This model was designed for operation under the following conditions:

— Continuous Variables: true.

— Nominal Variables: false.

— Discretized Variables: true.

— Integer Variables: true.

— Variables without values for some examples: true.

— Variables with imprecise values for some examples: true.

A more detailed description, performance evaluation, and review of current and further research of the four algorithms that we selected for this work can be found condensed in a survey paper [26]. They are recognized by the research community as ones of the most influential algorithms for data mining in the classification task.

4 Empirical Evaluation

In this section, we present the experimental methodology followed to evaluate the algorithms presented. In order to undertake the evaluation process, we performed four experiments. In the first one, we compared the performance of the EPUNN algorithm against the others using the accuracy metric. In this case, the evaluation was performed on 21 real-world data sets, in subsection 4.1.1 we present the main features of them. The second experiment was conducted by executing the algorithm EPUNN compared with the others on 11 data sets that were generated with different noise levels. The accuracy was used again as the evaluation metric in this case. Details of how these data sets were generated can be found in subsection 4.1.2. The third experiment was conducted to study the behavior of all algorithms under analysis on a series of data sets with different levels of imbalance, in subsection 4.1.3 we present the main features of them. In this case, as an evaluation metric, the well-known area under ROC curve (AUC) was used, following the

recommendations of [23]. Finally, we conducted the fourth experiment where we evaluated the behavior of the EPUNN accuracy on several data sets with missing values, the characteristics of these can be viewed in subsection 4.1.4. In what follows, we present the real-world problems chosen for the experimentation, the experimental tools and configurations parameters for each tool, the experimental results, and the statistical analysis applied to compare the obtained results.

4.1 Training Sets

4.1.1 Classical Real-World Training Sets

To perform the first experiment and evaluate the behavior of the EPUNN classifier, 21 real-world data set were chosen from the KEEL repository [1]. KEEL is an open source Java software tool which empowers the user to assess the behavior of evolutionary learning and Soft Computing based techniques; in particular, the KEEL-dataset includes the data set partitions in the KEEL format for using regression, clustering, multi-instance, imbalanced classification, multi-label classification, etc. The experiments were executed on the following data sets: appendicitis, australian, automobile, balance, banana, bands, breast, bupa, ecoli, glass, heart, hepatitis, ionosphere, iris, lymphography, pima, sonar, wdbc, wine, wisconsin, and zoo. A summary of the characteristics of these data sets can be seen in Table 1.

4.1.2 Led Data Sets

In order to obtain several data sets for the second experiment, we utilized the led generators from the UCI[3] repository [3]. With the parameters shown in Table 2, 11 data sets were generated with 512 instances in each one and with noise from 0 to 50.

[3]http://archive.ics.uci.edu/ml/

Table 1. Data sets characteristics

Data set	Instances	Atributes	Class
appendicitis	106	9	2
australian	690	14	2
automobile	159	25	6
balance	625	4	3
banana	5300	2	2
bands	365	19	2
breast	277	9	2
bupa	345	6	2
ecoli	336	7	8
glass	214	9	7
heart	270	13	2
hepatitis	80	19	2
ionosphere	351	33	2
iris	150	4	3
lymphography	148	18	4
pima	768	8	2
sonar	208	60	2
wdbc	569	30	2
wine	178	13	3
wisconsin	683	9	2
zoo	101	17	7

4.1.3 Imbalanced Data Sets

To perform the third experiment and evaluate the behavior of the EPUNN classifier, 16 imbalanced data sets were chosen from the KEEL repository [1]. In this case the experiments were executed with the following data sets: glass1, ecoli-0_vs_1, pima, iris0, glass0, glass-0-1-2-3_vs_4-5-6, ecoli1, ecoli2, glass6, ecoli3, ecoli4, glass-0-1-6_vs_5, glass2, glass4, glass5, and ecoli-0-1-3-7_vs_2-6. The first ten data sets have an imbalance ratio less than 9, and the last six, a ratio greater than 9, see Table 3.

4.1.4 Missing Values Data Sets

To perform the fourth experiment and evaluate the behavior of the EPUNN classifier, 10 missing values data sets were chosen from the KEEL repository [1]. In this case the experiments were executed with the following data sets: australian+mv, ecoli+mv, iris+mv, pima+mv, wine+mv, automobile-mv, bands-mv, breast-mv, hepatitis-mv,

Table 2. Led generator parameters

Data set	# Instances	Seed	% Noise
led7-i512-n00	512	12345678	0
led7-i512-n05	512	12345678	5
led7-i512-n10	512	12345678	10
led7-i512-n15	512	12345678	15
led7-i512-n20	512	12345678	20
led7-i512-n25	512	12345678	25
led7-i512-n30	512	12345678	30
led7-i512-n35	512	12345678	35
led7-i512-n40	512	12345678	40
led7-i512-n45	512	12345678	45
led7-i512-n50	512	12345678	50

and wisconsin-mv, see table 4. The first five data sets are used for standard classification with induced missing values.

4.2 Experimental Tools

The experiments were done with the KEEL framework [1]. For the experimental executions we used an Intel Core i3-2350M, 2300 MHz dual processor, with 6 GB of RAM, and Ubuntu 12.10 with Linux kernel 3.5.0-46 as the operating system. For each algorithm, we used the default KEEL configurations, except for the case of KNN where we set a value of $k = 5$.

4.3 Statistical Analysis

We followed the recommendations pointed out by Demšar [7] to perform the statistical analysis of the results in experiments 1 and 3. As suggested by him, we used non-parametric statistical tests to compare the accuracies of the models built by the different learning systems. To compare multiple learning methods, first, we applied a multi-comparison statistical procedure to test the null hypothesis that all the learning algorithms obtained the same results on average. Specifically, we used the Friedman's test. When the Friedman's test rejected the null hypothesis, we applied post-hoc Bonferroni-Dunn test and Holm's step-up step-down procedure.

Table 3. Imbalanced data sets

Data set	Imbalanced Ratio
glass1	1.82
ecoli-0_vs_1	1.86
pima	1.90
iris0	2.00
glass0	2.06
glass-0-1-2-3_vs_4-5-6	3.19
ecoli1	3.36
ecoli2	5.46
glass6	6.38
ecoli3	8.19
ecoli4	13.84
glass-0-1-6_vs_5	19.44
glass2	10.39
glass4	15.47
glass5	22.81
ecoli-0-1-3-7_vs_2-6	39.15

Table 4. Missing values data sets

Data set	% Missing
australian+mv	70.58
ecoli+mv	48.21
iris+mv	32.67
pima+mv	50.65
wine+mv	70.22
automobile-mv	28.83
bands-mv	32.28
breast-mv	3.15
hepatitis-mv	48.39
wisconsin-mv	2.29

4.4 Experimental Results

4.4.1 Experiment #1

In the first experiment we evaluated the performance of all the models with the accuracy metric (the proportion of correct classifications on previously unseen examples). We used a ten-fold cross validation procedure with 5 different random seeds over each data set. The average values of the results for each collection are shown in Table 5. The last row shows the average rank of each algorithm. As it can be seen, EPUNN-C is the KEEL implementation of EPUNN algorithm.

Table 5. Accuracy of the algorithms on the classic data sets

Data set	NB-C	C45-C	KNN-C	SVM-C	EPUNN-C
appendicitis	84.27 (4)	83.27 (5)	85.91 (3)	87.91 (1)	86.09 (2)
australian	86.67 (1.5)	85.22 (4)	84.78 (5)	85.80 (3)	86.67 (1.5)
automobile	67.95 (2)	80.93 (1)	56.54 (5)	61.97 (3)	58.38 (4)
balance	91.20 (3)	76.80 (5)	86.24 (4)	91.68 (2)	96.16 (1)
banana	71.17 (4)	89.08 (2)	89.11 (1)	55.17 (5)	74.45 (3)
bands	70.50 (1)	64.99 (5)	68.46 (4)	69.46 (2)	69.24 (3)
breast	74.40 (2)	76.92 (1)	72.28 (4)	70.75 (5)	72.93 (3)
bupa	61.16 (5)	67.00 (3)	61.31 (4)	70.14 (2)	72.76 (1)
ecoli	81.56 (1)	79.47 (3)	81.27 (2)	75.92 (5)	76.82 (4)
glass	69.28 (1)	67.44 (2)	66.85 (3)	62.59 (5)	64.40 (4)
heart	82.22 (2)	78.15 (5)	80.74 (4)	84.44 (1)	81.85 (3)
hepatitis	84.83 (2)	84.00 (3)	86.27 (1)	83.56 (4)	80.36 (5)
ionosphere	88.90 (3)	90.90 (2)	85.17 (5)	88.03 (4)	92.30 (1)
iris	94.67 (5)	96.00 (3)	96.00 (3)	96.67 (1)	96.00 (3)
lymphography	85.76 (1)	74.30 (5)	79.44 (4)	83.98 (2)	81.30 (3)
pima	74.88 (3)	74.23 (4)	73.06 (5)	77.10 (1)	76.71 (2)
sonar	77.38 (2)	70.07 (5)	83.10 (1)	77.26 (3)	74.90 (4)
wdbc	94.38 (4)	94.55 (3)	96.83 (1.5)	94.37 (5)	96.83 (1.5)
wine	96.05 (1.5)	94.90 (4)	96.05 (1.5)	94.38 (5)	96.01 (3)
wisconsin	97.67 (1)	95.63 (5)	96.95 (2)	96.51 (3)	96.36 (4)
zoo	94.47 (3)	92.81 (5)	93.64 (4)	96.50 (1)	95.89 (2)
Rank	2.4762	3.5714	3.1905	3.0000	2.7619

According to the results obtained, see Table 5, we statistically analyzed the results to detect significant differences in the accuracy of the obtained models by the different learning methods. The multi-comparison Friedman's test did not reject the null hypotheses that all the systems performed the same on average with $p = 0.1631$. The obtained $p_{value} > 0.05$ shows that there does not exist a significant difference between the five algorithms under study. Using an $\alpha = 0.10$ in equation 5 with k, the number of algorithms, $q_\alpha = 2.241$, and N, the number of data sets, a critical difference of 1.0935 was obtained:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}. \quad (5)$$

Figure 2 shows the results of Bonferroni-Dunn test, comparing all systems by means of accuracy. The major advantage of this test is that it seems to be easier to visualize because it uses the same critical difference for all comparisons [7].

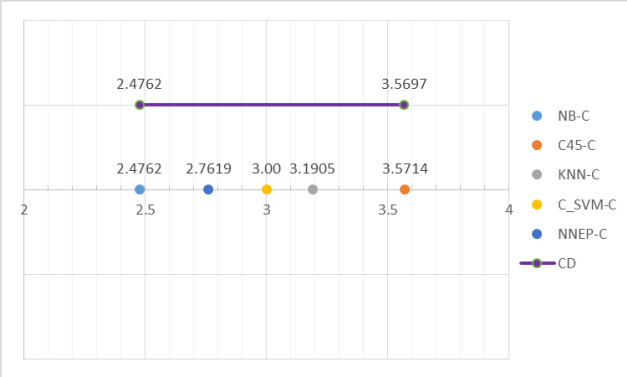

Fig. 2. Comparison of one classifier (NB-C) against the others with the Bonferroni-Dunn test

The EPUNN algorithm behaves similarly to four of the top ten algorithms, because its performance is within the critical difference, see Figure 2.

4.4.2 Experiment #2

In the second experiment we evaluated the performance of all models in the presence of noise with the accuracy metric. We used a ten-fold cross validation procedure with 5 different random seeds over each data set. The average values of the results for each collection are shown in Table 6.

As we can see in Figure 3, the accuracy of the EPUNN algorithm decays rapidly (exponentially) in the presence of noise. In the same manner, the remaining algorithms worsen exponentially. In Figure 3, the red dotted line represents the exponential curve fit to the EPUNN behavior.

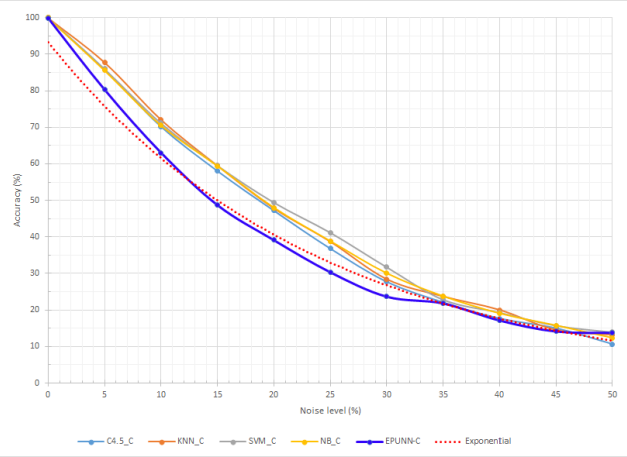

Fig. 3. Accuracy graphic of the algorithms in the presence of incremental noise in led7 data set

4.4.3 Experiment #3

In the third experiment we evaluated the performance of all the models with the AUC metric. We used a ten-fold cross validation procedure with 5 different random seeds over each data set. The average values of the results for each collection are shown in Table 7. The last three rows show the average rank when IR< 9, the average rank when IR> 9, and the total average rank of each algorithm.

According to the results obtained, see Table 7, we statistically analyzed the results to detect significant differences in the AUC of the obtained models by the different learning methods. The multi-comparison Friedman's test rejected the null hypotheses that all the systems performed the same on average with $p = 0.0824$. The obtained $p_{value} > 0.05$ shows that there does not exist a significant difference between the five algorithms under study on the imbalanced data sets. However, we can note a tendency to robustness of the EPUNN classifier with the increment of the imbalanced level. To validate these results, we also applied the Holm's step-up and step-down procedure sequentially to test the hypotheses ordered by their significance. As it can be observed in Table 8, the Holm's test at $\alpha = 0.05$ detected a significant difference between EPUNN and SVM-C, this is an important difference with respect to the experiment #1. Furthermore, in experiment #3, the winner algorithm was EPUNN.

4.4.4 Experiment #4

In the fourth experiment we evaluated the performance of the EPUNN classifier with the test accuracy metric. We used a ten-fold cross validation procedure with 5 different random seeds over each data set. The average values of the results for each collection are shown in Table 9. The last row shows the average rank of EPUNN over each data sets group.

According to the results obtained, see Table 9, we statistically analyzed the results to detect significant differences. We used the accuracy metric in EPUNN over classical data sets (EPUNNvsNotMV) and in EPUNN over missing values data sets (EPUNNvsMV). The multi-comparison Friedman's test rejected the null hypothesis that both cases (with and without missing values) behaved equally with a $p = 0.0114$. The obtained $p_{value} < 0.05$ shows that there exists a significant difference. To check this results, we also applied the Holm's step-up and step-down procedure sequentially to test the hypotheses ordered by their significance. As it can be observed in Table 10, Holm's test at $\alpha = 0.05$ showed significant differences between EPUNNvsNotMV and EPUNNvsMV.

Based on these results, we can conclude that the EPUNN algorithm was able to handle missing data, but in this kind of data, a significant performance deterioration was manifested.

Table 6. Accuracy of algorithms in the presence of incremental noise in led7 data set

Data set	% Noise	C4.5-C	KNN-C	SVM-C	NB-C	EPUNN-C
led7d-i512-n00	0	100.00	100.0000	100.0000	100.0000	99.84
led7d-i512-n05	5	85.94	87.8167	86.0173	85.6082	80.28
led7d-i512-n10	10	70.25	72.1395	71.1606	70.5539	63.05
led7d-i512-n15	15	58.06	59.5253	59.5419	59.4480	48.72
led7d-i512-n20	20	47.18	47.8258	49.4521	48.0045	39.15
led7d-i512-n25	25	36.80	38.7179	41.2014	38.8590	30.37
led7d-i512-n30	30	27.70	28.4231	31.7474	30.0313	23.68
led7d-i512-n35	35	22.04	23.6968	22.8360	23.7353	21.84
led7d-i512-n40	40	17.64	20.0011	19.3032	19.0290	17.17
led7d-i512-n45	45	14.99	14.4943	15.7247	15.7213	14.16
led7d-i512-n50	50	10.62	13.1844	13.9106	12.3047	13.76

Table 7. AUC achieved by the algorithms on the imbalanced data sets

Data set	IR	NB-C	C45-C	KNN-C	SVM-C	EPUNN-C
glass1	1.82	69.76 (4)	71.37 (2)	77.49 (1)	50.1 (5)	70.81 (3)
ecoli-0_vs_1	1.86	96.75 (3.5)	98.35 (1)	96.30 (5)	96.75 (3.5)	97.35 (2)
pima	1.90	73.06 (1)	70.35 (4)	66.43 (5)	71.96 (2)	70.59 (3)
iris0	2.00	100 (2.5)	99.00 (5)	100 (2.5)	100 (2.5)	100 (2.5)
glass0	2.06	80.99 (3)	81.67 (2)	82.69 (1)	69.14 (5)	75.59 (4)
glass-0-1-2-3_vs_4-5-6	3.19	92.70 (2)	91.66 (3)	91.30 (4)	90.32 (5)	93.56 (1)
ecoli1	3.36	86.56 (1)	85.94 (2)	80.32 (5)	81.85 (4)	84.05 (3)
ecoli2	5.46	86.43 (2)	85.92 (4)	90.37 (1)	73.59 (5)	86.17 (3)
glass6	6.38	92.67 (1)	81.41 (5)	87.39 (3)	91.75 (2)	86.85 (4)
ecoli3	8.19	85.85 (1)	72.79 (4)	74.49 (3)	50.00 (5)	78.17 (2)
ecoli4	13.84	88.26 (2)	81.39 (4)	87.03 (3)	57.50 (5)	89.68 (1)
glass-0-1-6_vs_5	19.44	88.03 (2)	88.03 (2)	81.90 (4)	49.71 (5)	88.03 (2)
glass2	10.39	46.45 (5)	68.30 (1)	60.64 (2)	50.00 (3)	49.24 (4)
glass4	15.47	86.47 (2)	80.02 (4)	83.12 (3)	56.95 (5)	87.22 (1)
glass5	22.81	82.60 (4)	88.64 (1)	88.16 (2)	50.00 (5)	83.09 (3)
ecoli-0-1-3-7_vs_2-6	39.15	57.14 (5)	71.25 (4)	84.98 (3)	85.71 (1)	85.17 (2)
$RankIR < 9$		2.10	3.20	3.05	3.90	2.75
$RankIR > 9$		3.33	2.67	2.83	4.00	2.17
$TotalRank$		2.56	3.00	2.97	3.94	2.53

Table 8. Holm / Hochberg Table for $\alpha = 0.05$

i	algorithm	$z = (R_0 - R_i)/SE$	p	Holm/Hochberg/Hommel
4	SVM-C	2.5156	0.0119	0.0125
3	C45-C	0.8385	0.4017	0.0167
2	KNN-C	0.7826	0.4338	0.0250
1	NB-C	0.0559	0.9554	0.0500

5 Conclusions and Future Work

In this paper, we tested the competitiveness of an EPUNN algorithm for classification task, generally using accuracy as the evaluation metric. Furthermore, AUC metric was used in evaluation

Table 9. Accuracy of EPUNN on classical data sets and EPUNN on data sets with missing values

% Missing	Data set	EPUNNvsMV	EPUNNvsNotMV	Data set	Difference
70.58	australian+mv	77.97	86.67	australian	deteriorate
48.21	ecoli+mv	81.85	76.82	ecoli	progress
32.67	iris+mv	94.00	96.00	iris	deteriorate
50.65	pima+mv	75.01	76.71	pima	deteriorate
70.22	wine+mv	88.76	96.01	wine	deteriorate
28.83	automobile-mv	53.10	58.38	automobile	deteriorate
32.28	bands-mv	67.27	69.24	bands	deteriorate
3.15	breast-mv	71.49	72.93	breast	deteriorate
48.39	hepatitis-mv	80.32	80.36	hepatitis	deteriorate
2.29	wisconsin-mv	96.23	96.36	wisconsin	deteriorate
	Rank	1.9000	1.1000		

Table 10. Holm / Hochberg Table for $\alpha = 0.05$

i	algorithm	$z = (R_0 - R_i)/SE$	p	Holm/Hochberg/Hommel
1	EPUNNvsMV	2.5298	0.0114	0.0500

of imbalanced data sets. As a result of the four experiments described in section 4.4, we can make the following conclusions:

— From experiment #1 described in subsection 4.4.1, we conclude that there does not exist a significant difference between EPUNN and the four algorithms assessed. We based this statement on the evaluation done over 21 benchmark data sets. For this reason, we recommend its use for the classification task.

— From experiment #2 described in subsection 4.4.2, we determined experimentally that the accuracy of EPUNN rapidly worsen in the presence of noise. For that reason, we do not recommend its utilization in noisy environments.

— From experiment #3 described in subsection 4.4.3, we can note a tendency to robustness in EPUNN despite of the growth of the imbalance ratio.

— From experiment #4 described in subsection 4.4.4, we can conclude that EPUNN is able to handle missing data; but in this kind of data, a significant performance deterioration was manifested.

In future work, we can also assess the impact of irrelevant attributes on the performance of EPUNN and the influence of noise using other data sets. Also, we can use more different kinds of data sets with missing values, since the nature of values and their distribution affect the performance of the classifiers.

References

1. **Alcala-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L., & Herrera, F.** (**2011**). Keel data-mining software tool: Data set repository and integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, Vol. 17, No. 2-3, pp. 255–287.
2. **Alcalá-fdez, J., Sánchez, L., García, S., del Jesus, M. J., Ventura, S., Garrell, J. M., Otero, J., Romero, C., Bacardit, J., Rivas, V. M., Fernández, J. C., & Herrera, F.** (**2008**). KEEL : a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, Vol. 13, No. 3, pp. 307–318.
3. **Bache, K. & Lichman, M.** (**2013**). UCI machine learning repository.
4. **Bishop, C. M.** (**1995**). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, U.K.

5. **Cortes, C. & Vapnik, V.** (**1995**). Support-Vector Networks. *Machine Learning*, Vol. 20, pp. 273–297.
6. **Cover, T. M. & Hart, P. E.** (**1967**). Nearest Neighbor. *IEEE Transactions on Information Theory*, Vol. IT-13, No. 1, pp. 21–27.
7. **Demšar, J.** (**2006**). Statistical Comparisons of Classifiers over Multiple Data Sets. *Machine Learning Research*, Vol. 7, pp. 1–30.
8. **Domingos, P. & Pazzani, M.** (**1997**). On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, Vol. 1997, No. 29, pp. 103–130.
9. **Durbin, R. & Rumelhart, D.** (**1989**). Products units: a computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, Vol. 1, pp. 133–142.
10. **Engelbrecht, A. P. & Ismail, A.** (**1999**). Training product unit neural networks. *Stability and Control: Theory and Applications*, Vol. 2, pp. 59–74.
11. **Fix, E. & Hodges, J.** (**1951**). An important contribution to nonparametric discriminant analysis and density estimation. *International Statistical Review*, Vol. 3, No. 57, pp. 233–238.
12. **Ismail, A. & Engelbrecht, A. P.** (**2000**). Global optimization algorithms for training product units neural networks. *International Joint Conference on Neural Networks IJCNN'2000, Italy*.
13. **John, G. H. & Pat, L.** (**1995**). Estimating Continuous Distributions in Bayesian Classifiers. San Mateo, California, pp. 338–345.
14. **Liu, H., Hussain, F., Lim, C., & Dash, M.** (**2002**). Discretization: An Enabling Technique. *Data Mining and Knowledge Discovery*, Vol. 2002, No. 6:4, pp. 393–423.
15. **Maron, M. E.** (**1961**). Automatic Indexing : An Experimental Inquiry. *Journal of the ACM (JACM)*, Vol. 8:3, No. January, pp. 404–417.
16. **Martínez-Estudillo, F., Hervás-Martínez, C., Gutiérrez, P., & Martínez-Estudillo, A.** (**2008**). Evolutionary product-unit neural networks classifiers. *Neurocomputing*, Vol. 72, No. 1-3, pp. 548–561.
17. **Martínez-Estudillo, F. J., Hervás-Martínez, C., Gutiérrez-Peña, P. A., & Martínez-Estudillo, A. C.** (**2006**). Evolutionary Product-Unit Neural Networks Classifiers. *International Joint Conference on Neural Networks*.
18. **P. J. Angeline, G. M. S. & Pollack, J. B.** (**1994**). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, pp. 54–65.
19. **Quinlan, J.** (**1986**). Induction of decision trees. *Machine Learning*, Vol. 1, pp. 81–106.
20. **Quinlan, J.** (**1993**). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Mateo, California.
21. **Rissanen, J.** (**1978**). Modeling by shortest data description. *Automatica*, Vol. 14, No. 5, pp. 465–471.
22. **Stanfill, C. & Waltz, D.** (**1986**). Instance-based learning algorithms. *Communications of the ACM*, Vol. 12, pp. 1213–1228.
23. **Sun, Y., Wong, A. K. C., & Kamel, M. S.** (**2009**). Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 23, No. 4, pp. 687–719.
24. **Wang, J., Neskovic, P., & Cooper, L. N.** (**2007**). Improving nearest neighbor rule with a simple adaptive distance measure. *Pattern Recognition Letters*, Vol. 28, pp. 207–213.
25. **Wilson, D. R. & Martinez, T. R.** (**2000**). Reduction tecniques for instance-based learning algorithms. *Machine Learning*, Vol. 38:3, pp. 257–286.
26. **Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z.-H., Steinbach, M., Hand, D. J., & Steinberg, D.** (**2007**). Top 10 algorithms in data mining. *Knowledge Information Systems*, Vol. (2008), No. 14, pp. 1–37.
27. **Yao, X. & Liu, Y.** (**1997**). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, Vol. 8, No. 3, pp. 694–713.

Alain Guerrero-Enamorado received his B.Sc. degree in Automatic Control from the Universidad de Oriente in 2003. He got a Master degree in Applied Informatics from the Universidad de las Ciencias Informáticas in 2009. He is currently an assistant professor at the Department of Programming Techniques of Faculty 1 of Universidad de las Ciencias Informáticas. Guerrero-Enamorado is a doctoral student in the Knowledge Discovery and Intelligent Systems Group, his main research

interests are in the fields of machine learning, data mining, and their applications.

Daimerys Ceballos-Gastell received her B.Sc. degree in Computer Engineering from the Universidad de las Ciencias Informáticas in 2008. She is currently an assistant professor at the Department of Programming Techniques in Faculty 1 of the Universidad de las Ciencias Informáticas. Ceballos-Gastell is a master student in Applied Informatics, her main research interests are in the fields of artificial neural networks, machine learning, data mining, and their applications.

Article received on 10/03/2015; accepted on 18/02/2016.
Corresponding author is Alain Guerrero-Enamorado.