Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 21 (2008) 951–961
Neural Networks
www.elsevier.com/locate/neunet

Multilogistic regression by means of evolutionary product-unit neural networks

C. Hervás-Martínez[a], F.J. Martínez-Estudillo[b,*], M. Carbonero-Ruz[b]

[a] Department of Computer Science and Numerical Analysis of the University of Córdoba, Campus de Rabanales, 14071, Córdoba, Spain
[b] Department of Management and Quantitative Methods, ETEA, Escritor Castilla Aguayo 4, 14005, Córdoba, Spain

Received 5 November 2006; accepted 13 December 2007

Abstract

We propose a multilogistic regression model based on the combination of linear and product-unit models, where the product-unit nonlinear functions are constructed with the product of the inputs raised to arbitrary powers. The estimation of the coefficients of the model is carried out in two phases. First, the number of product-unit basis functions and the exponents' vector are determined by means of an evolutionary neural network algorithm. Afterwards, a standard maximum likelihood optimization method determines the rest of the coefficients in the new space given by the initial variables and the product-unit basis functions previously estimated. We compare the performance of our approach with the logistic regression built on the initial variables and several learning classification techniques. The statistical test carried out on twelve benchmark datasets shows that the proposed model is competitive in terms of the accuracy of the classifier.
© 2008 Elsevier Ltd. All rights reserved.

Keywords: Evolutionary neural networks; Multi-class classification; Multilogistic regression

1. Introduction

Multi-class pattern recognition has a wide range of applications including handwritten digit recognition (Chiang, 1998), speech tagging and recognition (Athanaselis et al., 2005), bioinformatics (Mahony et al., 2006) and text categorization (Massey, 2003). Multi-class pattern recognition is a problem of building a system that accurately maps an input feature space to an output space of more than two pattern classes. Whereas a two-class classification problem is well understood, multi-class classification is relatively less-investigated. In general, the extension from two-class to the multi-class pattern classification problem is not trivial, and often leads to unexpected complexity or weaker performances. This paper presents a comprehensive and competitive study in multi-class neural learning which combines different elements such as multilogistic regression, neural networks and evolutionary algorithms.

The Logistic Regression model (LR) has been widely used in statistics for many years and has recently been the object of extensive study in the machine learning community. This traditional statistical tool arises from the desire to model the posterior probabilities of the class level given its observation via linear functions in the predictor variables. Although logistic regression is a simple and useful procedure, it poses problems when applied to a real-problem of classification, where we cannot always make the stringent assumption of additive and purely linear effects of the covariates. A technique to overcome these difficulties is to augment/replace the input vector with new variables, basis functions, which are transformations of the input variables, and then to use linear models in this new space of derived input features. One first approach is to augment the inputs with polynomial terms to achieve higher-order Taylor expansions, for example, with quadratic terms and multiplicative interactions. Once the number and the structure of the basis functions have been determined, the models are linear in these new variables and the fitting is a standard procedure. Methods like sigmoidal feed-forward neural networks (Bishop, 1995), projection pursuit learning (Friedman & Stuetzle, 1981), generalized additive models (Hastie & Tibshirani, 1990), and PolyMARS (Kooperberg,

* Corresponding author. Tel.: +34 957222120; fax +34 957222107.
E-mail addresses: chervas@uco.es (C. Hervás-Martínez), fjmestud@etea.com (F.J. Martínez-Estudillo), mcarbonero@etea.com (M. Carbonero-Ruz).

0893-6080/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.12.052

Bose, & Stone, 1997), which is a hybrid of multivariate adaptive splines (MARS) (Friedman, 1991) specifically designed to handle classification problems, can all be seen as different nonlinear basis function models. The major drawback of these approaches is in stating the typology and the optimal number of the corresponding basis functions.

Our approach, named Multilogistic Regression by using Linear and Product-Unit models (MRLPU), overcomes the nonlinear effects of the covariates by proposing a multilogistic regression model based on the combination of linear and product-unit models, where the nonlinear basis functions of the model are given by the product of the inputs raised to arbitrary powers. These basis functions express the possible strong interactions between the covariates, where the exponents are not fixed and may even take real values. The nonlinear basis functions of the proposed model correspond to a special class of feed-forward neural network, namely Product-Unit Neural Networks, PUNN, introduced by Durbin and Rumelhart (1989). They are an alternative to standard sigmoidal neural networks and are based on multiplicative nodes instead of additive ones.

One reason for adding linear terms to the nonlinear product unit is to yield models that are simpler and easier to interpret. In particular, if a covariate appears only linearly in the final model, then the model is a traditional parametric model with respect to that covariate. A second reason is to reduce the variance associated with the overall modeling procedure, and a third is to reduce the likelihood of ending up with unnecessary terms in the final model.

Logistic regression models are usually fit by maximum likelihood, where the Newton–Raphson algorithm is the traditional way to estimate the maximum a posteriori parameters. Usually, the algorithm converges since the log-likelihood is concave. However, in our approach, the nonlinearity of the PUNN implies that the corresponding Hessian matrix is generally indefinite and the likelihood has more local maximum. Besides, it is important to point out that the computation of the Newton–Raphson algorithm becomes prohibitive when the number of variables is large. These reasons justify, in our opinion, the use of an alternative heuristic procedure to estimate the parameters of the model.

The estimation of the coefficients is carried out in several steps. In a first step, an evolutionary algorithm determines the number of nonlinear basis functions in the model and their corresponding exponents in the basis functions. The nonlinear part of the model can be represented in a neural network framework, the aforementioned product-unit neural network. Evolutionary Artificial Neural Networks (EANNs) have been a key research area in the past decade providing a better platform for optimizing both network performance and its architecture simultaneously. Therefore, an evolutionary algorithm is applied to the design of the structure and training of the weights in a product-unit neural network. This step can be seen as a global search in the coefficients' model space. Once the basis functions have been determined by the evolutionary algorithm, we consider a transformation of the input space by adding the nonlinear transformations of the input variables given by the basis functions obtained by the evolutionary algorithm. The final model is linear in the set of variables formed by these new variables and the initial covariates. Now, the Hessian matrix is definite and fitting proceeds with the standard maximum likelihood optimization method. Finally, we use a backward stepwise procedure, pruning variables sequentially to the model previously obtained, until no further pruning can be made to improve the fit.

We evaluate the performance of our methodology on twelve datasets taken from the UCI repository (Blake & Merz, 1998). First, we compare MRLPU to linear multilogistic regression and then, we evaluate our approach by comparing it to other state-of-the-art learning schemes.

This paper is organized as follows: Section 2 shows the main related works; Section 3 is devoted to a description of the multilogistic regression model based on product-unit neural networks; Section 4 describes the MRLPU methodology; Section 5 explains the experiments carried out; and finally, Section 6 summarizes the conclusions of our work.

2. Related works

In this section, we give a brief overview of the methods which use different basis functions for moving beyond linearity. We also point out some recent studies which show a close relationship between logistic regression and machine learning methods.

The conventional statistical method of discriminant analysis (Hastie, Tibshirani, & Friedman, 2001) for solving classification problems assumes that the measurement vectors in each class follow a multivariate normal distribution. If the covariance matrices of the measurements in each class are the same, the method shows that the regions created by Bayes' decision rule are separated by boundaries which are linear in the input variables. Dropping the conventional assumption that covariate matrices are equal, Bayes' decision rule gives quadratic boundaries. In many examples, the inadequacy of linear or quadratic discriminant analysis for the purpose of classification made it necessary to look for approaches able to approximate highly nonlinear class boundaries. Instead of assuming specific distributions for the inputs and using them to calculate conditional class probabilities, these classes can be estimated directly from the training sample cases.

A number of methods based on nonparametric regression, which are capable of approximating highly nonlinear class boundaries in classification problems, have been developed in the last few years. Those methods more closely related to our proposal are listed below.

Generalized additive models (Hastie & Tibshirani, 1990) comprise automatic and flexible statistical methods that may be used to identify and characterize nonlinear effects. They provide a natural first approach to relaxing strong linear assumptions. The generalized additive model approximates multidimensional functions as a sum of univariate curves. Univariate functions are estimated in a flexible manner, using an algorithm whose basic building block is a scatter plot smoother; for example, the cubic smoothing spline. The additive model manages to retain interpretability by restricting nonlinear effects in the predictors so that they enter into the model independently of one another. Additive logistic regression (Friedman, Hastie, & Tibshirani, 1998) is an example of a generalized additive model.

In order to make interactions between predictors possible, the generalized additive model can be further refined. Kooperberg et al. (1997) propose an automatic procedure that uses linear splines and their tensor products. This method is a hybrid of the MultiAdaptive Regression Splines (MARS) (Friedman, 1991) called PolyMars, specifically designed to handle classification problems. It makes the model grow in a forward stage-wise fashion like MARS, but uses at each stage quadratic approximation to multinomial log-likelihood to search for the next basis function pair. Once found, the enlarged model is fit by maximum likelihood, and the process is repeated.

The spline model has been used by other authors. For example, Bose (1996) presented a method named Classification Using Splines (CUS), somewhat similar to the neural network methods, which used additive cubic splines to estimate conditional class probabilities. Later, the same author presented a modification of CUS, named successive projection method, to solve more complex classification problems (Bose, 2003). Although this method was presented using CUS, it is possible to replace CUS by any nonparametric regression-based classifier.

From a different point of view, neural networks have been an object of renewed interest among researchers, both in statistics and computer science, due to the interesting results obtained in a wide range of classification problems where they have been applied. Many different types of neural network architectures have been used, but the most popular one has been the single-hidden-layer feed-forward network. Related to our nonlinear model, we underline the Polynomial Feed-forward Neural Networks (PFNNs), which have been used in various practical tasks requiring high-order nonlinear descriptions. In Nikolaev and Iba (2003), the authors find the polynomial network structure by means of a population-based search technique relying on the genetic programming paradigm, and then adjust the best discovered network weights by an specially derived back-propagation algorithm. A more complete perspective and development of this approach can be found in Nikolaev and Iba (2006).

Another remarkable approach to the combination of regression techniques and neural networks can be found in Song et al. (2006), where the authors propose a method to integrate regression formulas as prior knowledge with kernel functions as a Knowledge-Based Neural Networks model (KBNN) for improved prediction and knowledge adaptation. The models incorporate different nonlinear regression functions as neurons in their hidden layer and adapt these functions through incremental learning from data in particular areas of the space.

Amongst the numerous approaches using neural networks in classification problems, we focus our attention on Evolutionary Artificial Neural Networks (EANNs) which have been a key research area in the past decade, providing a better platform for optimizing both network performance and architecture simultaneously. Miller, Todd, and Hedge (1989) proposed evolutionary computation as a very good candidate to search the space of architectures because the fitness function associated with that space is complex, noisy, non-differentiable, multi-modal and deceptive. Since then, many evolutionary programming methods have been developed for evolving artificial neural networks, see for example Angeline, Saunders, and Pollack (1994), Fogel (1993), García-Pedrajas, Hervás-Martínez, and Muñoz-Pérez (2002), Kasobov (2006), and Yao and Liu (1997). In these works we find several methods that combine architectural evolution with weight learning and use different mutation operators including, in some cases, partial training after each architectural mutation, or approaches that hybridize EANNs with a local search technique to improve the slowness of the convergence. For a very interesting review on the matter, see Yao (1999).

On the other hand, logistic regression has recently gained popularity in the machine learning community due to its close relations to well-known techniques such as Support Vector Machine (SVM) (Vapnik, 1999), AdaBoost (Freund & Shapire, 1996) and artificial neural networks (Schumacher, Robner, & Vach, 1996; Vach, Robner, & Schumacher, 1996). Vapnik (1999) compared LR and SVM in terms of minimizing the loss function, and showed that the loss function of LR can be very well approximated by SVM loss with multiple knots (SVMn). Friedman et al. (1998), discussed SVM, LR and boosting on top of their different loss functions. Lebanon and Lafferty (2002) showed that the only difference between AdaBoost and LR is that the latter requires the model to be normalized to a probability distribution. The logistic regression model can be seen as equivalent to a perceptron with a logistic activation function representing the simplest neural network. A comparative study of logistic regression and neural networks can be found in Schumacher et al. (1996) and Vach et al. (1996).

Finally, in a previous work (Hervás & Martínez-Estudillo, 2007), the authors combine logistic regression and neural network models to solve binary classification problems. The error function considered and the neural network framework used there were designed specifically for two-class classification problems.

3. Multilogistic regression and product-unit neural networks

In the classification problem, measurements x_i, $i = 1, 2, \ldots, k$, are taken on a single individual (or object), and the individuals are to be classified into one of J classes on the basis of these measurements. It is assumed that J is finite, and the measurements x_i are random observations from these classes. A training sample $D = \left\{(\mathbf{x}_n, \mathbf{y}_n); n = 1, 2, \ldots, N\right\}$ is available, where $\mathbf{x}_n = (x_{1n}, \ldots, x_{kn})$ is the vector of measurements taking values in $\Omega \subset \mathbb{R}^k$, and $\mathbf{y}_n$ is the class level of the nth individual. We adopt the common technique of representing the class levels using a "1-of-J" encoding vector $\mathbf{y} = \left(y^{(1)}, y^{(2)}, \ldots, y^{(J)}\right)$, such as $y^{(l)} = 1$ if $\mathbf{x}$ corresponds to an example belonging to class l and $y^{(l)} = 0$ otherwise. Based on the training sample, we wish to find a decision function

$C : \Omega \to \{1, 2, \ldots, J\}$ for classifying the individuals. In other words, C provides a partition, say $D_1, D_2, \ldots, D_J$, of Ω, where D_l corresponds to the lth class, $l = 1, 2, \ldots, J$, and measurements belonging to D_l will be classified as coming from the lth class. A misclassification occurs when a decision rule C assigns an individual (based on measurements vector) to a class j when it is actually coming from a class $l \neq j$.

To evaluate the performance of the classifiers we define the corrected classified rate by $CCR = \frac{1}{N}\sum_{n=1}^{N} I(C(\mathbf{x}_n) = \mathbf{y}_n)$, where $I(\cdot)$ is the zero-one loss function. A good classifier tries to achieve the highest possible CCR in a given problem. It is usually assumed that the training data are independent and identically distributed samples from an unknown probability distribution. Suppose that the conditional probability that $\mathbf{x}$ belongs to class l verifies: $p\left(y^{(l)} = 1 \middle| \mathbf{x}\right) > 0$, $l = 1, 2, \ldots, J$, $\mathbf{x} \in \Omega$, and set the function:

$$f_l(\mathbf{x}, \boldsymbol{\theta}_l) = \log \frac{p\left(y^{(l)} = 1 \middle| \mathbf{x}\right)}{p\left(y^{(J)} = 1 \middle| \mathbf{x}\right)}, \quad l = 1, 2, \ldots, J, \ \mathbf{x} \in \Omega,$$

where $\boldsymbol{\theta}_l$ is the weight vector corresponding to class l and $f_J(\mathbf{x}, \boldsymbol{\theta}_J) \equiv 0$. Under a multinomial logistic regression, the probability that $\mathbf{x}$ belongs to class l is then given by

$$p\left(y^{(l)} = 1 \middle| \mathbf{x}, \boldsymbol{\theta}\right) = \frac{\exp f_l\left(\mathbf{x}, \boldsymbol{\theta}_l\right)}{\sum_{j=1}^{J} \exp f_j\left(\mathbf{x}, \boldsymbol{\theta}_j\right)}, \quad l = 1, 2, \ldots, J,$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{J-1})$.

For binary problems ($J = 2$), this is known as logistic regression (or soft-max in neural network literature).

The classification rule coincides with the optimal Bayes' rule. In other words, an individual should be assigned to the class which has the maximum probability, given the vector measurement $\mathbf{x}$:

$$C(\mathbf{x}) = \hat{l}, \quad \text{where } \hat{l} = \arg\max_l f_l(\mathbf{x}, \hat{\boldsymbol{\theta}}_l), \text{ for } l = 1, \ldots, J.$$

On the other hand, due to the normalization condition we have:

$$\sum_{l=1}^{J} p\left(y^{(l)} = 1 \middle| \mathbf{x}, \boldsymbol{\theta}\right) = 1,$$

and the probability for one of the classes (the last one, in our case) need not be estimated. Observe that we have considered $f_J(\mathbf{x}, \boldsymbol{\theta}_J) \equiv 0$.

The usual parametric approach to the multinomial logistic regression problem is to use the linear model in the input variables:

$$f_l(\mathbf{x}, \boldsymbol{\theta}_l) = \boldsymbol{\theta}_{0l} + \boldsymbol{\theta}_{1l}x_1 + \cdots + \boldsymbol{\theta}_{kl}x_k, \quad l = 1, 2, \ldots, J-1,$$

where $\boldsymbol{\theta}_l = (\boldsymbol{\theta}_{0l}, \boldsymbol{\theta}_{1l}, \ldots, \boldsymbol{\theta}_{kl})$, $l = 1, 2, \ldots, J-1$.

In practice, however, it may be desirable to model the predictor effects by using smooth, nonlinear functions. A generalized additive model (Hastie & Tibshirani, 1990) for multinomial logistic regression is given by

$$f_l(\mathbf{x}, \boldsymbol{\theta}_l) = \boldsymbol{\theta}_{1l}(x_1) + \cdots + \boldsymbol{\theta}_{kl}(x_k), \quad l = 1, 2, \ldots, J-1,$$

where $\boldsymbol{\theta}_{il}(x_i)$, $i = 1, 2, \ldots, k$ are one-dimensional functions.

In order to make interactions between predictors possible, the additive model can be further refined. For example, the model proposed by Kooperberg et al. (1997) is given by sums of polynomial splines and their tensor products. They consider interactions of two variables at most to ameliorate the course of dimensionality.

Our logistic regression model proposal is based on the combination of the standard linear model and a nonlinear term constructed with basis functions given by products of the inputs raised to real powers, which capture possible strong interactions between the variables.

The general expression of the model is given by:

$$f_l(\mathbf{x}, \boldsymbol{\theta}_l) = \alpha_0^l + \sum_{i=1}^{k} \alpha_i^l x_i + \sum_{j=1}^{m} \beta_j^l \prod_{i=1}^{k} x_i^{w_{ji}}, \quad l = 1, 2, \ldots, J-1,$$

where $\boldsymbol{\theta}_l = (\boldsymbol{\alpha}^l, \boldsymbol{\beta}^l, \mathbf{W})$, $\boldsymbol{\alpha}^l = (\alpha_0^l, \alpha_1^l, \ldots, \alpha_k^l)$, $\boldsymbol{\beta}^l = \left(\beta_1^l, \ldots, \beta_m^l\right)$ and $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$, with $\mathbf{w}_j = (w_{j1}, w_{j2}, \ldots, w_{jk})$, $w_{ji} \in \mathbb{R}$.

The nonlinear part of $f_l(\mathbf{x}, \boldsymbol{\theta}_l)$ corresponds to a special class of feed-forward neural networks, namely Product-Unit Neural Networks (PUNNs), introduced by Durbin and Rumelhart (1989) and subsequently developed by other authors (Ismail & Engelbrecht, 2000; Janson & Frenzel, 1993; Leerink et al., 1995; Martínez-Estudillo et al., 2006a, 2006b). They are an alternative to standard sigmoidal neural networks and are based on multiplicative nodes instead of additive ones. This class of multiplicative neural networks comprises such types as sigma–pi networks and product-unit networks. In contrast to the sigma–pi unit, product units use a power function to control the strength of propagation for each link, the exponents are not fixed and may even take real values. Advantages of PUNNs include: increased information capacity and the ability to form higher-order combinations of inputs. They are universal approximators and it is possible to obtain upper bounds of the VC dimension of product-unit neural networks that are similar to those obtained for sigmoidal neural networks (Schmitt, 2001). Despite these obvious advantages, — PUNNs have a major drawback that their training is more difficult than the training of standard sigmoidal-based networks (Durbin & Rumelhart, 1989). The main reason for this difficulty is that small changes in the exponents can cause large changes in the total error surface. Hence, networks based on product units have more local minima and a greater probability of becoming trapped in them. It is well known (Janson & Frenzel, 1993) that back-propagation is not efficient in training product units. Several efforts have been made to develop learning methods for product units (Leerink et al., 1995; Martínez-Estudillo et al., 2006b; Saito & Nakano, 2002), mainly in a regression context.

4. Multilogistic regression linear product-unit neural networks methodology (MRLPU)

In the supervised learning context, the components of the weight vectors $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{J-1})$ are estimated from the training dataset D. To perform the maximum likelihood (ML) estimation of $\boldsymbol{\theta}$, one can minimize the negative log-likelihood function

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N} \log p\left(\mathbf{y}_n \middle| \mathbf{x}_n, \boldsymbol{\theta}\right) = \frac{1}{N}\sum_{n=1}^{N}\left[-\sum_{l=1}^{J} y_n^{(l)} f_l(\mathbf{x}_n, \boldsymbol{\theta}_l) + \log\sum_{l=1}^{J}\exp f_l(\mathbf{x}_n, \boldsymbol{\theta}_l)\right].$$

The error surface associated with the model is very convoluted with numerous local optima. The nonlinearity of the model with respect to the parameters $\boldsymbol{\theta}_l$, and the indefinite character of the associated Hessian matrix do not recommend the use of gradient-based methods to maximize the log-likelihood function. Moreover, the optimal number of basis functions of the model (i.e. the number of hidden nodes in the product-unit neural network) is unknown. Thus, the estimation of the vector parameter $\hat{\boldsymbol{\theta}}$ is carried out by means of a hybrid procedure described below.

The methodology proposed is based on the combination of an evolutionary algorithm (global explorer) and a local optimization procedure (local exploiters) carried out by the standard maximum likelihood optimization method. In a first step, an evolutionary algorithm (EA) is applied to design the structure and training of the weights of a product-unit neural network. The evolutionary process determines the number m of product-unit basis functions in the model, and the corresponding vector $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$ of exponents. Once the basis functions have been determined by the evolutionary algorithm, we consider a transformation of the input space by adding the nonlinear transformations of the input variables given by the basis functions obtained by the evolutionary algorithm. The model is linear in these new variables and the initial covariates. The remaining coefficient vector $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are calculated by the maximum likelihood optimization method. Finally, we use a backward stepwise procedure, pruning variables sequentially to the model previously obtained, until no further pruning can be made to improve the fit.

4.1. Estimation of the model coefficients

In this paragraph we describe the different aspects of the MRLPU methodology in detail. The process is structured in four steps:

Step 1. We apply an evolutionary neural network algorithm to find the basis functions

$$B(\mathbf{x}, \hat{\mathbf{W}}) = \left\{B_1(\mathbf{x}, \hat{\mathbf{w}}_1), B_2(\mathbf{x}, \hat{\mathbf{w}}_2), \ldots, B_m(\mathbf{x}, \hat{\mathbf{w}}_m)\right\}$$

corresponding to the nonlinear part of $f(\mathbf{x}, \boldsymbol{\theta})$. We have to determine the number of basis functions m and the weight vector $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$.

To apply evolutionary neural network techniques, we consider a product-unit neural network with the following structure (Fig. 1): an input layer with a node for every input variable; a hidden layer with several nodes; and an output layer with nodes, one for each category. There are no connections between the nodes of a layer and none between the input and output layers either. The activation function of the jth node in the hidden layer is given by $B_j(\mathbf{x}, \mathbf{w}_j) = \prod_{i=1}^{k} x_i^{w_{ji}}$ where w_{ji} is the weight of the connection between input node i and hidden node j and $\mathbf{w}_j = (w_{j1}, \ldots, w_{jk})$. The activation function of the output node l is given by

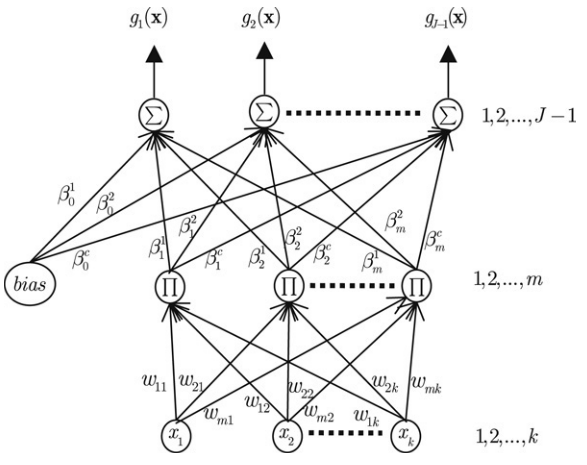

Fig. 1. Model of a product-unit-based neural network.

$$g_l(\mathbf{x}, \boldsymbol{\beta}^l, \mathbf{W}) = \beta_0^l + \sum_{j=1}^{m} \beta_j^l B_j(\mathbf{x}, \mathbf{w_j}),$$

where β_j^l is the weight of the connection between the hidden node j and the output node l. The transfer function of all output nodes is the identity function.

The weight vector $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$ is estimated by means of an evolutionary neural network algorithm (detailed in Section 4.2) that optimizes the error function given by the negative log-likelihood for N observations associated with the product-unit model:

$$L^*(\boldsymbol{\beta}, \mathbf{W}) = \frac{1}{N}\sum_{n=1}^{N}\left[-\sum_{l=1}^{J-1} y_n^{(l)} g_l(\mathbf{x}_n, \boldsymbol{\beta}^l, \mathbf{W}) + \log\sum_{l=1}^{J-1}\exp g_l(\mathbf{x}_n, \boldsymbol{\beta}^l, \mathbf{W})\right].$$

Although in this step the evolutionary process obtains a concrete value for the $\boldsymbol{\beta}$ vector, we only consider the estimated weight vector $\hat{\mathbf{W}} = (\hat{\mathbf{w}}_1, \hat{\mathbf{w}}_2, \ldots, \hat{\mathbf{w}}_m)$, which builds the basis functions. The value for the $\boldsymbol{\beta}$ vector will be determined in step 3 together with the $\boldsymbol{\alpha}$ coefficient vector.

Step 2. We consider the following transformation of the input space by including the nonlinear basis functions obtained by the evolutionary algorithm in step 1:

$$H : \mathbb{R}^k \to \mathbb{R}^{k+m}$$
$$(x_1, x_2, \ldots, x_k) \to (x_1, x_2, \ldots, x_k, z_1, \ldots, z_m),$$

where $z_1 = B_1(\mathbf{x}, \hat{\mathbf{w}}_\mathbf{1}), \ldots, z_m = B_m(\mathbf{x}, \hat{\mathbf{w}}_m)$.

Step 3. We minimize the negative log-likelihood function for N observations:

$$L(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{N}\sum_{n=1}^{N}\left[-\sum_{l=1}^{J} y_n^{(l)}(\boldsymbol{\alpha}^l \mathbf{x}_n + \boldsymbol{\beta}^l \mathbf{z}_n) + \log \sum_{l=1}^{J}\exp(\boldsymbol{\alpha}^l \mathbf{x}_n + \boldsymbol{\beta}^l \mathbf{z}_n)\right],$$

where $\mathbf{x}_n = (1, x_{1n}, \ldots, x_{kn})$. Now, the Hessian matrix of the negative log-likelihood in the new variables $x_1, x_2, \ldots, x_k, z_1, \ldots, z_m$ is semi-definite positive. Therefore, we could apply Newton's method, also known in this case as Iteratively Reweighted Least Squares (IRLS). Although there are other methods to perform this optimization, none clearly outperforms IRLS (Minka, 2003). The estimated vector coefficient $\hat{\boldsymbol{\theta}} = (\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}, \hat{\mathbf{W}})$ determines the model:

$$f_l(\mathbf{x}, \hat{\boldsymbol{\theta}}) = \hat{\alpha}_0^l + \sum_{i=1}^{k}\hat{\alpha}_i^l x_i + \sum_{j=1}^{m}\hat{\beta}_j^l \prod_{i=1}^{k} x_i^{\hat{w}_{ji}}, \quad l = 1, 2, \ldots, J-1.$$

Step 4. In order to select the final model, we use a backward stepwise procedure: we start with the full model with all the covariates, initial and PU, pruning variables to the model sequentially and successively, until no further pruning can be made to improve the fit. At each step, we select the least significant covariate in the first discriminant function, i.e., the one which shows the greatest critical value (p-value) in the hypothesis test, where the associated coefficient equal to zero is the hypothesis to be contrasted. The selected covariate is deleted if this does not reduce the fit. If it holds and the deletion of the covariate reduces the fit, we repeat the same process with the second discriminant function, choosing its least significant covariate, which does not necessarily coincide with the one selected for the first function. We continue with this process until we get to the last discriminant function. If none of the covariates is deleted, we restart with the second least significant covariate in each discriminant function, following the procedure previously described. The procedure ends when all the tests for each covariate in each function provide p-values smaller than the fixed significance level, or when none of the two chosen covariates in any of the functions is deleted.

4.2. Evolutionary algorithm

Among the different paradigms of Evolutionary Computation, we have chosen Evolutionary Programming (EP) due to the fact that we are evolving artificial neural networks. The population-based evolutionary algorithm for architectural design and the estimation of real-coefficients have points in common with other evolutionary algorithms in the bibliography (Angeline et al., 1994; García-Pedrajas et al., 2002; Martínez-Estudillo et al., 2006b; Yao & Liu, 1997). The search begins with an initial population and, with each iteration, the population is updated using a population-update algorithm. The population is subjected to the operations of replication and mutation. Crossover is not used due to its potential disadvantages in evolving artificial networks (Angeline et al., 1994).

The general structure of the EA is organized as follows:

(1) Generate a random population of size N.
(2) Repeat until the stopping criterion is fulfilled.
  (a) Calculate the fitness of every individual in the population.
  (b) Rank the individuals with respect to their fitness.
  (c) The best individual is copied into the new population.
  (d) The best 10% of population individuals are replicated and substitute the worst 10% of individuals.
  Over that intermediate population we:
  (e) Apply parametric mutation to the best 10% of individuals.
  (f) Apply structural mutation to the remaining 90% of individuals.

We consider $L^*(\boldsymbol{\beta}, \mathbf{W})$ as the error function of an individual $g(\cdot, \boldsymbol{\beta}, \mathbf{W})$ of the population. Observe that g is a product-unit neural network and can be seen as a multi-valuated function:

$$g\left(\mathbf{x}, \boldsymbol{\beta}, W\right) = \left(g_1\left(\mathbf{x}, \boldsymbol{\beta}^1, \mathbf{W}\right), \ldots, g_{J-1}\left(\mathbf{x}, \boldsymbol{\beta}^{J-1}, \mathbf{W}\right)\right).$$

The fitness measure is a strictly decreasing transformation of the error function $L^*(\boldsymbol{\beta}, W)$ given by $A(g) = \frac{1}{1+L^*(\boldsymbol{\beta}, W)}$, where $0 < A(g) \leq 1$.

Parametric mutation is accomplished for each coefficient w_{ji}, β_j^l of the model with Gaussian noise:

$$w_{ji}(t+1) = w_{ji}(t) + \xi_1(t), \qquad \beta_j^l(t+1) = \beta_j^l(t) + \xi_2(t),$$

where $\xi_k(t) \in N(0, \alpha_k(t))$, $k = 1, 2$, represents a one-dimensional normally distributed random variable with mean 0 and variance $\alpha_k(t)$. Once the mutation is performed, the fitness of the individual is recalculated and the usual simulated annealing (Kirkpatric, Gellat, & Vecchi, 1983) is applied. Thus, if ΔA is the difference in the fitness function before and after the random step, the criterion is: if $\Delta A \geq 0$, the step is accepted, and if $\Delta A < 0$, the step is accepted with a probability $\exp(\Delta A / T(g))$, where the temperature $T(g)$ of an individual g is given by $T(g) = 1 - A(g)$, $0 < T(g) < 1$.

The variance $\alpha_k(t)$ is updated throughout the evolution. There are different methods to update the variance. We use one of the simplest methods: the 1/5 success rule of Rechenberg (1975). This rule states that the ratio of successful mutations should be 1/5. Therefore, if the ratio of successful mutations is greater than 1/5, the mutation deviation should increase; otherwise, the deviation should decrease. Thus:

$$\boldsymbol{\alpha}_k(t+s) = \begin{cases}(1+\lambda)\alpha_k(t) & \text{if } s_g > 1/5\\ (1-\lambda)\alpha_k(t), & \text{if } s_g < 1/5\\ \alpha_k(t) & \text{if } s_g = 1/5,\end{cases}$$

where $k = 1, 2$, s_g is the frequency of successful mutations over s generations and $\lambda = 0.1$. The adaptation tries to avoid being trapped in local minima and to speed up the evolutionary process when the searching conditions are suitable. It should be pointed out that the modification of the exponents w_{ji} is different from the modification of the coefficients β_j^l, therefore $\alpha_1(t) \ll \alpha_2(t)$.

Table 1
Datasets used for the experiments

Data set	Instances	Numeric attributes	Binary attributes	Nominal attributes	Inputs	Classes	Class distribution
Pima-indians	768	8	0	0	8	2	500, 268
Australian card	690	6	4	5	51	2	307, 383
Ionosphere	351	33	1	–	34	2	126, 225
Heart-statlog	270	13	–	–	13	2	150, 120
Breast-cancer	286	–	3	6	15	2	201, 85
German	1000	6	3	11	61	2	700, 300
Balance-scale	625	4	–	–	4	3	288, 49, 288
Vehicle	846	18	–	–	18	4	240, 240, 240, 226
Hypothyroid	3772	7	20	2	29	4	3481, 194, 95, 2
Iris	150	4	0	0	4	3	50, 50, 50, 50
Lymphografy	148	3	9	6	38	4	2, 81, 61, 4
Glass	214	9	0	0	9	6	70, 17, 76, 13, 9, 29

Structural mutation implies a modification in the structure of the function and allows the exploration of different regions in the search space, helping to keep the diversity of the population. There are five different structural mutations: node addition, node deletion, connection addition, connection deletion, and node fusion. These five mutations are applied sequentially to each network. The mutations are performed as follows:

- Node addition. One or more nodes are added to the hidden layer. The connections with the output nodes are chosen randomly and have a random value in the interval $[-I, I]$. The connections from the input layer are chosen randomly and its values are also random values in an interval $[-O, O]$.
- Connection addition. Connection addition mutations are first performed in the hidden layer and then in the output layer. When adding a connection from the input layer to the hidden layer, a neuron from each layer is selected randomly, and then the connection is added with a random weight. A similar procedure is performed from the hidden to the output layer.
- Node deletion. One or more nodes, together with their connections, are selected randomly and deleted.
- Connection deletion. In the same way, connection deletion mutation is first performed in the hidden layer and then in the output layer, choosing randomly the origin neuron from the previous layer and the target neuron from the mutated layer.
- Node fusion. Two randomly selected nodes, a and b, are replaced by a new node c, which is a combination of the two. The connections common to both nodes are kept with a weight given by:

$$\beta_c^l = \beta_a^l + \beta_b^l, \qquad w_{jc} = \frac{w_{ja} + w_{jb}}{2}.$$

Those connections not shared by the nodes are inherited by c with probability 0.5 and their weight is unchanged.

The number of hidden nodes added in node addition, node deletion and node fusion mutations is calculated as $\Delta_{MIN} + uT(g)[\Delta_{MAX} - \Delta_{MIN}]$, u being a random uniform variable in the interval $[0, 1]$, $T(g) = 1 - A(g)$ the temperature of the neural net, and Δ_{MIN} and Δ_{MIN} a minimum and maximum number of hidden nodes to be added. However, the connection addition and deletion mutations are performed in a slightly different way. For each mutated neural net, we apply connection mutations sequentially, first, adding or deleting $1 + u[\Delta_O n_O]$ connections from the hidden layer to the output layer and then, adding or deleting $1 + u[\Delta_H n_H]$ connections from the input layer to the hidden layer, u being a random uniform variable in the interval $[0, 1]$, Δ_O and Δ_H a previously defined ratio of number of connections in the hidden and the output layer, and n_O and n_H the current number of connections in the output and the hidden layer.

Parsimony is also encouraged in evolved networks by attempting the five structural mutations sequentially, where node or connection deletion and node fusion is always attempted before addition. Moreover, the deletion and fusion operations are made with higher probability ($T(g)$ for deletion and fusion mutations and $T^2(g)$ for addition ones). If a deletion or fusion mutation is successful, no other mutation will be made. If the probability does not select any mutation, one of the mutations is chosen at random and applied to the network.

The parameters used in the evolutionary algorithm are the same in all twelve problems analyzed below. We have considered $\alpha_1(0) = 0.5$, $\alpha_2(0) = 1$, $\lambda = 0.1$ and $s = 5$. The exponents w_{ji} are initialized in the $[-2, 2]$ interval, and the coefficients β_j^l are initialized in $[-5, 5]$. The maximum number of hidden nodes is $m = 9$. The size of the population is $N = 1000$. The number of nodes that can be added or removed in a structural mutation is within the $[1, 2]$ interval and the ratio of the number of connections in the hidden and the output layer is $\Delta_O = 0.05$ and $\Delta_H = 0.3$.

The stop criterion is reached whenever one of the following two conditions is fulfilled: a number of generations is reached (200 generations in our experiments) or the variance of the fitness of the best ten percent of the population is less than 10^{-4}.

We have done a simple linear rescaling of the input variables in the interval $[1, 2]$, X_i^* being the transformed variables. The lower bound is chosen to avoid input values near 0, which could produce very large values of the outputs for negative exponents. The upper bound is chosen to avoid dramatic changes in the outputs of the network when there are weights with large values (especially in the exponents).

5. Experiments

In this section we compare MRLPU to a variety of learning algorithms on twelve benchmark datasets taken from the UCI

Table 2
Sample-mean difference between MRLPU and Mlogistic acc for each dataset and its p-value

	Difference	p-value
Pima-indians	0.12	0.888
Australian card	2.46	0.006[b]
Ionosphere	2.58	0.167
Heart-statlog	2.19	0.084[a]
Breast-cancer	1.43	0.268
German	1.20	0.024[b]
Balance-scale	7.61	0.007[b]
Vehicle	−0.63	0.357
Hypothyroid	0.58	0.078[a]
Iris	0.65	0.733
Lymphografy	−0.72	0.591
Glass	1.90	0.462

[a] Significant at 0.1 level.
[b] Significant at 0.05 level.

Table 3
Average size of Mlogistic and MRLPU models for each dataset

	Covariates		
	Mlogistic	MRLPU	
		Original	PU
Pima-indians[a]	8	3.4	2.6
Australian card[a]	38	27.9	1.8
Ionosphere[a]	33	29.5	2.2
Heart-statlog[a]	13	4.3	1.6
Breast-cancer[a]	15	4.5	3.0
German[a]	48	39.5	2.9
Balance-scale	4	1.7	3.7
Vehicle	18	14.7	4.4
Hypothyroid	29	25.9	4.6
Iris	4	3.0	2.3
Lymphografy[a]	38	29.6	2.3
Glass	9	8.3	9.1

[a] MRLPU size is smaller than Mlogistic size.

repository (see Table 1). First, we compare MRLPU to linear logistic regression using the initial covariates. Afterwards, we evaluate the performance of our method compared to other machine learning algorithms.

5.1. *Linear logistic regression comparison*

We compare our MRLPU approach to a standard version of logistic regression that builds a full logistic model on all attributes present in the data (Mlogistic). We use a 10-fold stratified cross validation for the datasets considered. For every dataset we perform ten runs of each fold. This gives a hundred data points for each algorithm and dataset, from which the average classification accuracy in the generalization phase and the number of variables (original variables and PU) are calculated. We consider this average as a measure of its classification accuracy and number of variables.

We use the t-statistic for two related samples to compare average percentages of correct classification, after having tested the hypothesis of normality using the Saphiro–Wilks test.

Table 2 presents the sample-mean difference between Mlogistic and MRLPU with its p-value for each dataset, three of them corresponding to Australian Card, German, and Balance-scale, which are significant in favour of MRLPU at 0.05 significance level, whereas two others, Heart-statlog and Hypothoroid are significant at 0.1. The difference seems to indicate that Mlogistic is more accurate than MRLPU in only two cases. This all suggests that MRLPU is able to improve logistic regression results in at least some of the problems.

Finally, Table 3 indicates the average size of Mlogistic and MRLPU models in the 10-folds for each dataset. Size coincides with the number of covariates in the dataset for logistic regression. For the MRLPU model, we distinguish between original covariates and product-unit basis functions (PU). Note that the average size of the MRLPU (original covariates plus PU) is smaller than that of Mlogistic in seven out of twelve cases. These results combined with previous results showed in Table 2 indicate that in some cases, even if MRLPU is not more effective than Mlogistic as far as accuracy is concerned, we may reduce the model size if we use PU variables without a significant reduction in the average classification results, as Heart-statlog or Breast-Cancer examples show.

In short, we can highlight that the proposed model sometimes outperforms Mlogistic significantly, determining a good balance between the linear and nonlinear part.

5.2. *Machine learning methods comparison*

In this section we compare our MRLPU approach to recent results (Landwehr, Hall, & Eibe, 2005) obtained using ten different methodologies: Logistic model tree algorithm, LMT; logistic regression with characteristics selection, SLogistic; induction trees (C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984)); three logistic tree algorithms: NBTree, LTreeLin and LTreeLog (Gama, 2004), and finally, multiple-tree models M5’ (Wang & Witten, 1997) for classification, and boosted C4.5 trees using AdaBoost.M1 with 10 and 100 boosting iterations.

Table 4 shows the average test error rates of learners generated by the previously mentioned algorithms (the results have been taken from Landwehr et al. (2005)) and the MRLPU model proposed.

Under the hypothesis of normality of the results, we test for significant differences between the MRLPU model and the aforementioned methods. Our decisions on the comparisons will be determined by using confidence intervals for mean differences. We choose the Dunnett (Hochberg & Tamhane, 1987) procedure to build these intervals: this is a multiple comparison with a control method (MRLPU in this case) to identify which of these algorithms are superior, similar or inferior to MRLPU regarding CCR for each dataset.

The Dunnett method for multiple mean comparisons uses k samples, each resulting from one of k different treatments, to determine the following $k-1$ simultaneous confidence intervals for the difference between each treatment mean and the mean of control for a fixed confidence level:

$$\mu_k - \mu_i \in \hat{\mu}_k - \hat{\mu}_i \mp D\hat{\sigma}\sqrt{\frac{1}{n_i} + \frac{1}{n_k}} \quad i = 1, \ldots, k-1,$$

Table 4
Mean classification accuracy and standard deviation for Logistic Model Tree (LMT), Simple Logistic Regression (SLogistic), C4.5, CART, NBTree, LTreeLin, M5' for classification, AdaBoost with 10 and 100 interactions and MRLPU

	LMT	SLogistic	C4.5	CART	NBTree	LTreeLin
Pima-indians	77.08 ± 4.65	77.10 ± 4.65	74.49 ± 5.27	74.50 ± 4.70	75.18 ± 5.05	76.73 ± 4.83
Australian card	85.04 ± 3.84	85.04 ± 3.97	85.57 ± 3.96	84.55 ± 4.20	85.07 ± 4.03	84.99 ± 3.91
Ionosphere	92.99 ± 4.13	87.78 ± 4.99	89.74 ± 4.38	89.80 ± 4.78	89.49 ± 5.12	88.95 ± 5.10
Heart-statlog	83.22 ± 6.50	83.30 ± 6.48	78.15 ± 7.42	78.00 ± 8.25	80.59 ± 7.12	83.52 ± 6.28
Breast-cancer	74.91 ± 6.29	74.94 ± 6.25	74.28 ± 6.05	69.40 ± 5.25	70.99 ± 7.94	70.58 ± 6.90
German	75.37 ± 3.53	75.34 ± 3.50	71.25 ± 3.17	73.34 ± 3.66	74.07 ± 4.10	74.90 ± 3.47
Balance-scale	89.71 ± 2.68	88.74 ± 2.91	77.82 ± 3.42	78.09 ± 3.97	75.83 ± 5.32	92.86 ± 3.22
Vehicle	83.04 ± 3.70	80.45 ± 3.37	72.28 ± 4.32	72.37 ± 4.12	71.03 ± 4.55	79.52 ± 3.81
Hypothyroid	99.54 ± 0.39	96.78 ± 0.73	99.54 ± 0.36	99.66 ± 0.30	99.58 ± 0.38	98.98 ± 0.52
Iris	95.80 ± 4.89	95.93 ± 4.82	94.73 ± 5.30	96.00 ± 4.74	93.53 ± 5.64	98.00 ± 3.35
Lymphografy	84.10 ± 10.00	84.37 ± 9.97	75.84 ± 11.05	76.86 ± 9.87	80.89 ± 8.77	80.30 ± 8.98
Glass	69.15 ± 8.99	65.29 ± 8.03	67.63 ± 9.31	68.09 ± 10.49	70.16 ± 9.67	65.27 ± 10.42

	LTreeLog	M5'	AdaBoost(10)	AdaBoost(100)	MRLPU
Pima-indians	76.64 ± 4.69	76.56 ± 4.71	71.81 ± 4.85	73.89 ± 4.75	76.98 ± 5.23
Australian card	84.64 ± 4.09	85.39 ± 3.87	84.01 ± 4.36	86.43 ± 3.98	87.55 ± 3.89
Ionosphere	88.18 ± 5.06	89.92 ± 4.18	93.05 ± 3.92	94.02 ± 3.83	91.99 ± 5.67
Heart-statlog	83.00 ± 6.83	82.15 ± 6.77	78.59 ± 7.15	80.44 ± 7.08	85.77 ± 8.61
Breast-cancer	70.45 ± 6.78	70.40 ± 6.84	66.75 ± 7.61	66.36 ± 8.18	72.50 ± 4.47
German	74.94 ± 3.41	74.99 ± 3.31	70.91 ± 3.60	74.53 ± 3.26	76.90 ± 5.20
Balance-scale	92.78 ± 3.49	87.76 ± 2.23	78.35 ± 3.78	76.11 ± 4.09	96.30 ± 3.50
Vehicle	79.32 ± 3.72	78.66 ± 4.38	75.59 ± 3.99	77.87 ± 3.58	78.34 ± 5.24
Hypothyroid	99.25 ± 0.40	99.44 ± 0.38	99.65 ± 0.31	99.69 ± 0.31	97.16 ± 0.77
Iris	97.13 ± 4.57	94.93 ± 5.62	94.33 ± 5.22	94.53 ± 5.05	95.99 ± 4.66
Lymphografy	76.90 ± 10.07	80.35 ± 9.32	80.87 ± 8.63	84.72 ± 8.41	84.28 ± 11.57
Glass	64.77 ± 9.90	71.30 ± 9.08	75.15 ± 7.59	78.78 ± 7.80	67.62 ± 10.48

where in this expression we denote by $\hat{\mu}_i$ and $\hat{\sigma}$:

$$\hat{\mu}_i = \bar{Y}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} Y_{ij} \quad i = 1, \ldots, k$$

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n_i}\left(Y_{ij} - \bar{Y}_i\right)^2}{\sum_{i=1}^{k}(n_i - 1)}.$$

Y being the compared variable, n_i the sample size from ith population and D a tabulated value depending on k, on the sample sizes, and on the confidence level.

These intervals may be used for decision making about differences in mean between populations as follows: we must accept that compared populations have a similar mean if zero belongs to the corresponding interval; if it does not hold the sign of sample average difference points to the sign of population mean difference.

In our case, we have $k = 11$, $n_1 = \cdots = n_{11} = 100$ and $D = 2.45$, (Hochberg & Tamhane, 1987) for a 95% confidence level. In addition, we can use the following expression to calculate $\hat{\sigma}$ since we know the sample deviation $\hat{\sigma}_i$ for each dataset and procedure

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{k}(n_i - 1)\,\hat{\sigma}_i^2}{\sum_{i=1}^{k}(n_i - 1)}.$$

Table 5 shows the results of these comparisons. In this table, 0 is placed when MRLPU is similar to the corresponding dataset and procedure, −1 when MRLPU is inferior, and 1 when it is superior. With respect to the data, MRLPU sometimes outperforms all or almost all alternative procedures, as it does in the German or Balance-scale datasets. However, the algorithm does not achieve good results in some problems, such as the Hypothyroid one. Nevertheless, from a procedural point of view, MRLPU seems to be a competitive scheme.

If we take the whole set of datasets into consideration, we can affirm that none of the other procedures outperforms MRLPU. In order to assert this affirmation, we present in Table 6 the summary of MRLPU comparisons (Table 5) and similar summaries for AdaBoost (100) and LMT, surely two of the best algorithms in Machine Learning. For every database and algorithm we marked the most usual result: wins (W), draws (D) or losses (L).

Therefore, we can conclude that the results obtained by MRLPU make it a competitive method when compared to the learning schemes previously mentioned.

6. Conclusions

To the best of our knowledge, the approach presented in this paper is the first study in multi-class neural learning which combines three tools used in machine learning research: logistic regression, the product-unit neural network model and the evolutionary neural network paradigm. The method presents an adequate combination of the three elements to solve classification problems.

Table 5
Results comparing MRLPU method with Logistic Model Tree (LMT), Simple Logistic Regression (SLogistic), C4.5, CART, NBTree, LTreeLin, M5' for classification, AdaBoost with 10 and 100 iterations

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
Pima-indians	0	0	1	1	0	0	0	0	1	1
Australian Card	1	1	1	1	1	1	1	1	1	0
Ionosphere	0	1	1	1	1	1	1	1	0	−1
Heart-statlog	0	0	1	1	1	0	1	1	1	1
Breast-Cancer	0	0	0	1	0	0	0	0	1	1
German	1	1	1	1	1	1	1	1	1	1
Balance	1	1	1	1	1	1	1	1	1	1
Vehicle	−1	−1	1	1	1	0	0	0	1	0
Hypothyroid	−1	1	−1	−1	−1	−1	−1	−1	−1	−1
Iris	0	0	0	0	1	−1	0	0	0	0
Lymphografy	0	0	1	1	0	1	1	1	0	0
Glass	0	0	0	0	0	0	0	−1	−1	−1

(1) LMT; (2) Slogistic; (3) C4.5; (4) CART; (5) NBTree; (6) LTreeLin; (7) LTreeLog; (8) M5'; (9) AdaBoost(10); (10) AdaBoost(100).

Table 6
Results comparing MRPLU method with Logistic Model Tree (LMT) and AdaBoost 100 iterations (resume)

	MRLPU			Adaboost(100)			LMT		
	W	D	L	W	D	L	W	D	L
Pima-indians	4	**6**	0	1	3	**6**	**5**	**5**	0
Australian Card	**9**	1	0	3	**7**	0	0	**9**	1
Ionosphere	**7**	2	1	**8**	2	0	**7**	3	0
Heart-statlog	**7**	3	0	0	**7**	3	3	**7**	0
Breast-cancer	3	**7**	0	0	1	**9**	**7**	3	0
German	**10**	0	0	2	**7**	1	3	**6**	1
Balance-scale	**10**	0	0	0	1	**9**	**6**	1	3
Vehicle	**4**	**4**	2	**4**	3	3	**10**	0	0
Hypothyroid	1	0	**9**	**5**	**5**	0	4	**6**	0
Iris	1	**8**	1	0	**8**	2	1	**8**	1
Lymphografy	**5**	**5**	0	**7**	3	0	4	**6**	0
Glass	0	**7**	3	**10**	0	0	3	**5**	2

One of the main contributions of this paper is in suggesting that adequate product-unit-based functions can learn the data and build new variables for each problem. Product units have the ability to capture the nonlinear interactions between input variables and to reduce the dimension of the new space built with the transformed variables. One of the main reasons for evolving product units is that they feature a much more parsimonious structure than other learnable nonlinear models, like MLP perceptron networks for example, due to the expressive power of the activation functions used in the hidden layer. The evolutionary neural networks algorithm discovers the basic structure of the model: number of basis functions and input variables, making up each basis function and the most complex parameters of the model defined by their corresponding exponents. Thus, it tailors the network to the task and overcomes to a great degree any problems arising from mis-specified network architectures. Subsequently, the new variables are incorporated into multilogistic regression with the initial covariates. Therefore, the process of building the model carried out in two phases is flexible and adapted to each problem. The classifier obtained is a mixture of linear and nonlinear models, where the nonlinear model is given by the product-unit neural network. The right balance between the two parts of the model for each dataset allows us to express the underlying structure of the data.

We compare our MRLPU approach to a standard version of logistic regression that uses initial covariates. The proposed model outperforms standard logistic regression in five of the twelve datasets and obtains similar results in the remainder datasets. Finally, using a multiple comparison test (Dunnett's test) in the experimental study, we obtain results which prove competitive performance of the MRLPU algorithm over algorithms like Logistic Model Tree (LMT) and AdaBoost (100), proving also clear outperformance over the rest of the algorithms tested.

Acknowledgements

This work has been supported in part by the Spanish Inter-Ministerial Commission of Science and Technology (MICYT) under the project TIN 2005-08386-C05-02 and with FEDER funds.

References

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(1), 54–65.

Athanaselis, T., et al. (2005). ASR for emotional speech: Clarifying the issues and enhancing performance. *Neural Networks*, *18*(4), 437–444.

Bishop, M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Blake, C., & Merz, C.J. (1998). UCI repository of machine learning data bases. www.ics.uci.edu/mlearn/MLRepository.thml.

Bose, S. (1996). Classification using splines. *Computational Statistics & Data Analysis*, *22*, 505–525.

Bose, S. (2003). Multilayer statistical classifiers. *Computational Statistics & Data Analysis*, *42*, 685–701.

Breiman, L., et al. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Chiang, J.-H. (1998). A hybrid neural network model in handwritten word recognition. *Neural Networks*, *11*(2), 337–346.

Durbin, R., & Rumelhart, D. (1989). Products units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, *1*, 133–142.

Fogel, D. B. (1993). Using evolutionary programming to greater neural networks that are capable of playing Tic-Tac-Toe. In *International conference on neural networks*. San Francisco, CA: IEEE Press.

Freund, Y., & Shapire, R. (1996). Experiments with a new boosting algorithm. In *Machine learning: Proceedings of the thirteenth international conference*. San Francisco: Morgan Kauffman.

Friedman, J. (1991). Multivariate adaptive regression splines (with discussion). *The Annals of Statistics*, *19*, 1–141.

Friedman, J., Hastie, T., & Tibshirani, R. (1998). Additive logistic regression: A statistical view of boosting. Dept. of Statistics, Standford University.

Friedman, J., & Stuetzle, W. (1981). Proyection pursuit regression. *Journal of the American Statistical Association*, *76*(376), 817–823.

Gama, J. (2004). Functional trees. *Machine Learning*, *55*(3), 219–250.

García-Pedrajas, N., Hervás-Martínez, C., & Muñoz-Pérez, J. (2002). Multiobjetive cooperative coevolution of artificial neural networks. *Neural Networks*, *15*(10), 1255–1274.

Hastie, T. J., & Tibshirani, R. J. (1990). *Generalized additive models*. London: Chapman & Hall.

Hastie, T., Tibshirani, R. J., & Friedman, J. (2001). *The elements of statistical learning. Data mining, inference and prediction*. Springer.

Hervás, C., & Martínez-Estudillo, F. J. (2007). Logistic regression using covariates obtained by product-unit neural network models. *Pattern Recognition*, *40*, 52–64.

Hochberg, Y., & Tamhane, A. C. (1987). *Multiple comparison procedures*. John Wiley and Sons.

Ismail, A., & Engelbrecht, A. P. (2000). Global optimization algorithms for training product units neural networks. In *International joint conference on neural networks*.

Janson, D. J., & Frenzel, J. F. (1993). Training product unit neural networks with genetic algorithms. *IEEE Expert*, *8*(5), 26–33.

Kasobov, N. K. (2006). *Evolving connectionist systems 'the knowledge engineering approach'*. London: Springer-Verlag.

Kirkpatric, S., Gellat, C. D. J., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Kooperberg, C., Bose, S., & Stone, C. J. (1997). Polychotomous regression. *Journal of the American Statistical Association*, *92*, 117–127.

Landwehr, N., Hall, M., & Eibe, F. (2005). Logistic model trees. *Machine Learning*, *59*, 161–205.

Lebanon, G., & Lafferty, J. (2002). Boosting and maximun likelihood for exponential models. In *Proc. NIPS*.

Leerink, L. R., et al. (1995). Learning with products units. *Advances in Neural Networks Processing Systems*, *7*, 537–544.

Mahony, S., et al. (2006). Self-organizing neural networks to support the discovery of DNA-binding motifs. In *Advances in self organising maps — WSOM'05*. *Neural Networks*, *19*(6–7), 950–962.

Martínez-Estudillo, A. C., et al. (2006a). Hybridation of evolutionary algorithms and local search by means of a clustering method. *IEEE Transactions on Systems, Man and Cybernetics, Part. B: Cybernetics*, *36*(3), 534–546.

Martínez-Estudillo, A. C., et al. (2006b). Evolutionary product unit based neural networks for regression. *Neural Networks*, *19*, 477–486.

Massey, L. (2003). On the quality of ART1 text clustering. In *Advances in neural networks research: IJCNN '03*. *Neural Networks*, *16*(5–6), 771–778.

Miller, G. F., Todd, P. M., & Hedge, S. U. (1989). Designing neural networks using genetic algorithms. In *Proc. 3rd int. conf. genetic algorithms and their applications*. San Mateo, CA: Morgan Kaufmann.

Minka, T. (2003). A comparison of numerical optimizers for logistic regression. Dept. of Statistics, Carnegie Mellon Univ.

Nikolaev, N. Y., & Iba, H. (Eds.) (2006). *Backpropagation and Bayesian methods, adaptive learning of polynomial networks: Genetic programming*. New York: Springer.

Nikolaev, N. Y., & Iba, H. (2003). Learning polynomial feedforward neural networks by genetic programming and backpropagation. *IEEE Transactions on Neural Networks*, *14*(1), 337–350.

Quinlan, R. (1993). *C4.5: Programs for machine learning*. Morgan Kauffman.

Rechenberg, I. (1975). *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Evolution*. Stuttgart Framman-Holzboog Verlag.

Saito, K., & Nakano, R. (2002). Extracting regression rules from neural networks. *Neural Networks*, *15*, 1279–1288.

Schmitt, M. (2001). On the complexity of computing and learning with multiplicative neural networks. *Neural Computation*, *14*, 241–301.

Schumacher, M., Robner, R., & Vach, W. (1996). Neural networks and logistic regression: Part I. *Computational Statistics & Data Analysis*, *21*, 661–682.

Song, Q., et al. (2006). Integrating regression formulas and kernel functions into locally adaptive knowledge-based networks: A case study on renal function evaluation. *Artificial Intelligence in Medicine*, *36*, 235–244.

Vach, W., Robner, R., & Schumacher, M. (1996). Neural networks and logistic regression: Part II. *Computational Statistics & Data Analysis*, *21*, 683–701.

Vapnik, V. (Ed.) (1999). *The nature of statistical learning theory*. Springer.

Wang, Y., & Witten, I. (1997). Inducing model trees for continuous classes. In *Proceedings of poster papers, European conference on machine learning*.

Yao, X. (1999). Evolving artificial neural network. *Proceedings of the IEEE*, *9*(87), 1423–1447.

Yao, X., & Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, *8*(3), 694–713.