Optimizing the Area Under the ROC Curve in Multilayer Perceptron-based Classifiers

Raúl Ramos-Pollán
Centro Extremeño de Tecnologías Avanzadas
CIEMAT
Trujillo, Spain
e-mail: raul.ramos@ciemat.es

Naimy González de Posada,
Miguel Angel Guevara López
INEGI-Faculty of Engeniering
University of Porto
Porto, Portugal
e-mail: {nposada,mguevaral}@inegi.up.pt

Abstract - **This paper proposes a method to adapt existing multilayer perceptron (MLP) training algorithms for optimizing the area under the receiver operating characteristic curve (AUC), in binary classification tasks. It is known that error rate minimization does not necessarily yield to optimal AUC and, rather than developing new MLP training algorithms for AUC optimization, we reuse the vast experience encoded into existing algorithms by replacing the error metrics used to guide their training processes, with a novel defined AUC loss function, leaving unmodified their core logic. The new method was evaluated over 2000 MLP configurations, using four different training algorithms (backpropagation, resilient propagation, simulated annealing and genetic algorithms) in 12 binary datasets from the UCI repository. AUC was improved in 5.86% in average and, in addition, the proposed definition preserves interesting properties of other error metrics. An efficient AUC calculation procedure was also developed to ensure the method remains computationally affordable.**

Keywords - Multilayer perceptron, AUC optimization, error measure, machine learning, binary classifiers.

I. INTRODUCTION

Receiver Operating Characteristic (ROC) analysis [1] was originally used in signal processing and, more recently, is commonly used in biomedicine [2]. ROC curves measure the capability of a binary test or classifier to correctly distinguish between positive and negative instances, accounting for the trade-off between true and false positives rates, and the area under the ROC curve (denoted by AUC or Az) is used as a scalar comparative metric. In medicine, ROC curves are used to analyze and compare diagnostic systems [3] or for mining biomedical data [4]. In machine learning they are increasingly used to evaluate and compare classifier performance in general [5, 6].

Machine learning classifier performance is often measured by its accuracy (number of dataset elements correctly classified), which is obtained by fixing a specific threshold on a score produced by the classifier for each dataset element. Both accuracy and AUC are complementary measures to evaluate classifier performance, and AUC has the property of being insensitive to class distribution (class skew). Although both are obviously related, it is well established that error rate minimization does not necessarily yield AUC optimization [7], and several efforts have been invested into using AUC in machine learning [6] and, most recently, to build AUC optimizing classifiers, mostly from scratch or by re-designing the core of existing algorithms to include AUC metrics, such as support vector machines [8], gradient descent [9], evolutionary algorithms [10] and others [11-16].

In a previous work [17], we successfully adapted multilayer perceptron (MLP) classifiers trained with simulated annealing for AUC optimization. Now we aim to do so in a generalized manner so that, rather than developing new algorithms, our approach is to reuse the vast experience encoded into existing MLP training algorithms, devising a method through which they can be adapted with little effort, leaving untouched their core logic. This method is based on a novel AUC error metric (loss function) herewith defined that replaces the error metrics used in existing training algorithms to guide their training processes, therefore requiring simply the substitution of the error calculation routines of any MLP implementation.

This paper is structured as follows. Section 2 defines the novel AUC error metrics just mentioned and describes the method used to compute them efficiently. Section 3 explains the experimental setup devised to validate our approach by injecting the proposed metric into four existing MLP training algorithms and the results obtained. Finally, in Section 4 we draw the conclusions and outline future work.

II. AUC OPTIMIZATION IN MULTILAYER PERCEPTRON BASED CLASSIFIERS

This section formally defines the proposed method to adapt existing MLP training algorithms for AUC optimization in the sense just described. First, we settle for the notation defined in Table 1 to refer to the different elements in of a binary classification task (dataset elements, binary MLP classifiers and error measures). Then, we use the definition of the Mann-Whitney statistic for AUC [3] to obtain an AUC based error measure (loss function) and discuss its theoretical properties. Finally, since AUC calculation is typically expensive, we also provide an algorithm for an efficient error-bound approximation of the AUC, ensuring our method does not render modified MLP training algorithms impracticable.

A. Generalities

Since we are dealing with AUC optimization we assume the case of MLP based binary classifiers having two output

 ISBN: 978-1-61208-154-0

neurons: one positive neuron (representing the *positiveness* assigned by the MLP to the input vector) and one negative neuron (representing its *negativeness*), and we use the definitions detailed in Table 1.

TABLE 1: DEFINITIONS FOR BINARY MULTILAYER PERCEPTRONS

$\mathcal{X} \subset \mathbb{R}^p$	Domain of elements consisting of input vectors (with p features)
$\mathcal{Y} = \{P, N\}$	The two classes into which input vectors are classified
$(x, y), x \in \mathcal{X}, y \in \mathcal{Y}$	Element with its associated class (for supervised training)
$SD = \{(x_1, y_1), \ldots, (x_n, y_n)\}$	Dataset (for supervised training)
$S_P = \{x \mid (x, P) \in SD\}$	Positive elements of dataset
$S_N = \{x \mid (x, N) \in SD\}$	Negative elements of dataset
$S = S_P \cup S_N$	Elements of a dataset
$S(x) = y$	Class associated to element x through dataset S
$\lvert S\rvert = n$	Size of dataset
$\mathcal{F} = \{ f: \mathcal{X} \to \mathcal{Y} \}$	Set of functions representing binary classifiers
$h \in \mathcal{F}$	A binary classifier
$h(x) \in \mathcal{Y}$	Output of binary classifier h when applied to element x
$h_{sc}(x) \in \mathbb{R}$	Score assigned by binary classifier h to element x it is typically used by h to determine $h(x)$ by applying some threshold, and obtain ROC curves
$\mathcal{E}(S, h)$	A global error measure of classifier h when applied to training set S
$e(x, h)$	An individual error measure of classifier h when applied to element x
$Az(S, h)$	Area under the ROC curve (AUC) of dataset S when classified with h
$\mathcal{O} = [nn_{min}, nn_{max}] \subset \mathbb{R}$	Range of output values for the two output neurons of a binary MLP
$h_P(x), h_N(x) \in \mathcal{O}$	Output of the positive and negative neurons of the MLP based binary classifier h upon element x
$d_P(x), d_N(x) \in \mathcal{O}$	Ideal value for positive and negative neurons for x
$e_P(x, h), e_N(x, h)$	Error measures for the output of the positive and negative neurons of binary MLP h upon element x

At this point, given a binary classifier h and a dataset S, we distinguish two kinds of MLP training algorithms: (1) those that iterate through all dataset elements, using the error measures of each element $x \in S$ at the positive and negative output neurons, denoted by $e_P(x, h)$ and $e_N(x, h)$, and (2) those using the global error measure for the whole dataset, $\mathcal{E}(S, h)$, (see Table 1). We name the first kind of algorithms as *element error training algorithms* and the second kind as *dataset error training algorithms*. Notice that dataset error training algorithms only use $\mathcal{E}(S, h)$ regardless how it is calculated. Although typically a global error measure $\mathcal{E}(S, h)$ for a whole dataset is obtained by iterating over the error measures of all elements (such as by calculating their mean), this might not always be the case.

For a given dataset element $x \in S$ we define the ideal values at the positive and negative output neurons as:

$$\begin{aligned} d_P(x) &= nn_{\max} \quad d_N(x) = nn_{\min} \quad if \quad S(x) = P \\ d_P(x) &= nn_{\min} \quad d_N(x) = nn_{\max} \quad if \quad S(x) = N \end{aligned} \tag{1}$$

and fix a score metric, which linearly transforms the output of the two neurons to the [0,1] interval according to eq. 1, so that a score of 0.0 corresponds to an ideal negative element ($h_P(x) = nn_{min}$ and $h_N(x) = nn_{max}$) and a score of 1.0 corresponds to an ideal positive element ($h_P(x) = nn_{max}$ and $h_N(x) = nn_{min}$)

$$h_{sc}(x) = \frac{h_P(x) - h_N(x)}{2(nn_{max} - nn_{min})} + \frac{1}{2} \tag{2}$$

It can be easily proven that this definition ensures that $h_{sc}(x)$ stays within the [0,1] interval. In fact, for ROC purposes this restriction is not strictly needed as what matters is the relative ordering between positive and negative dataset elements induced by the score h_{sc} assigned to each one.

Commonly, a distance metric measures the error at the neuron's output with respect to the ideal output:

$$\begin{aligned} \Delta_P(x, h) &= d_P(x) - h_P(x) \\ \Delta_N(x, h) &= d_N(x) - h_N(x) \end{aligned} \tag{3}$$

B. Root Mean Square error measures

Based on the previous definitions, a Root Mean Square (RMS) error measure is commonly established for a dataset element and for the whole dataset as follows:

$$\begin{aligned} e^{RMS}(x, h) &= \sqrt{\frac{\Delta_P(x, h)^2 + \Delta_N(x, h)^2}{2}} \\ \mathcal{E}^{RMS}(S, h) &= \frac{\sum_{x \in S} e^{RMS}(x, h)}{|S|} \end{aligned} \tag{4}$$

Having, therefore, $\mathcal{E}^{RMS}(S, h)$ as the mean of $e^{RMS}(x, h)$. Then, $e^{RMS}(x, h)$ is simply mapped to each output neuron using the distance metric of eq. 3 as follows:

$$\begin{aligned} e_P^{RMS}(x, h) &= \Delta_P(x, h) \\ e_N^{RMS}(x, h) &= \Delta_N(x, h) \end{aligned} \tag{5}$$

In this case, dataset error training algorithms would use $\mathcal{E}^{RMS}(S, h)$, whereas element error training algorithms would use $e_P^{RMS}(x, h)$ and $e_N^{RMS}(x, h)$.

C. AUC Error Metrics

We want to make a AUC based error measure so that it can be injected back into training algorithms by either substituting $\mathcal{E}^{RMS}(S, h)$ or $e_P^{RMS}(x, h)$ and $e_N^{RMS}(x, h)$

without altering the rest of the logic. For this, we use the definition of the Mann-Whitney statistic for AUC [3]:

$$AUC(S,h) = \frac{\sum_{p\in S_P}\sum_{n\in S_N}\mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P|\cdot|S_N|} \quad (6)$$

where $1[L]$ denotes the indicator function, yielding 1 if L is true and 0 otherwise. Through this, the contribution of dataset element x to $AUC(S,h)$ is established as:

$$AUC(x,h) = \begin{cases} \frac{\sum_{n\in S_N}\mathbf{1}[h_{sc}(n) < h_{sc}(x)]}{|S_P|\cdot|S_N|} & if\ x \in S_P \\ \frac{\sum_{p\in S_P}\mathbf{1}[h_{sc}(x) < h_{sc}(p)]}{|S_P|\cdot|S_N|} & if\ x \in S_N \end{cases} \quad (7)$$

Notice the fact that the maximum possible values for $AUC(h,x)$ are reached when the score of x is greater than the score of all negative elements if x is a positive element (and inversely when x is a negative element):

$$AUC_{MAX}(x,h) = \begin{cases} \frac{|S_N|}{|S_P|\cdot|S_N|} = \frac{1}{|S_P|} & if\ x \in S_P \\ \frac{|S_P|}{|S_P|\cdot|S_N|} = \frac{1}{|S_N|} & if\ x \in S_N \end{cases} \quad (8)$$

With this, we define the following error measures for dataset elements and for the whole dataset.

$$e^{AUC}(x,h) = 1 - \frac{AUC(x,h)}{AUC_{MAX}(x,h)}$$
$$\mathcal{E}^{AUC}(S,h) = 1 - AUC(S,h) \quad (9)$$

It would be tempting to define $e^{AUC}(x,h) = AUC_{MAX}(x,h) - AUC(x,h)$, however, $AUC_{MAX}(x,h)$ is usually a very small value (specially for large datasets), which would make it unpractical for MLP training purposes. In addition, this definition preserves the fact that the dataset error measure is the mean of the elements error measure, such as is the case between $\mathcal{E}^{RMS}(S,h)$ and $e^{RMS}(x,h)$.

Lemma 1: $\mathcal{E}^{AUC}(S,h)$ is the mean of $e^{AUC}(x,h)$ over the dataset elements as defined in eq. 9

Proof: Exploiting the fact that a binary dataset can be split into positive and negative elements:

$$\sum_{x\in S} e^{AUC}(x,h) = \sum_{p\in S_P} e^{AUC}(p,h) + \sum_{n\in S_N} e^{AUC}(n,h)$$

$$= \sum_{p\in S_P} 1 - \frac{AUC(p,h)}{AUC_{max}(p,h)} + \sum_{n\in S_N} 1 - \frac{AUC(n,h)}{AUC_{max}(n,h)}$$

$$= |S_P| - \sum_{p\in S_P}|S_P|\cdot AUC(p,h) + |S_N| - \sum_{n\in S_N}|S_N|\cdot AUC(n,h)$$

$$= |S_P| - \sum_{p\in S_P}|S_P|\cdot\frac{\sum_{n\in S_N}\mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P|\cdot|S_N|} + |S_N| - \sum_{n\in S_N}|S_N|\cdot\frac{\sum_{p\in S_P}\mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P|\cdot|S_N|}$$

$$= |S_P| - |S_P|\cdot\frac{\sum_{p\in S_P}\sum_{n\in S_N}\mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P|\cdot|S_N|} + |S_N| - |S_N|\cdot\frac{\sum_{n\in S_N}\sum_{p\in S_P}\mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P|\cdot|S_N|}$$

$$= |S_P| - |S_P|\cdot AUC(S,h) + |S_N| - |S_N|\cdot AUC(S,h)$$

$$= |S_P|\cdot\big(1 - AUC(S,h)\big) + |S_N|\cdot\big(1 - AUC(S,h)\big)$$

$$= (|S_P| + |S_N|)\cdot\big(1 - AUC(S,h)\big) = |S|\cdot\big(1 - AUC(S,h)\big)$$

$$= |S|\cdot\mathcal{E}^{AUC}(S,h)$$

$$\Rightarrow \mathcal{E}^{AUC}(S,h) = \frac{\sum_{x\in S} e^{AUC}(x,h)}{|S|}$$

■

However, $e^{AUC}(x,h)$ still needs to be mapped to each output neuron of a binary MLP classifier, which we do in the following way:

$$e_P^{AUC}(x,h) = e^{AUC}(x,h)\cdot\frac{\Delta_P(x,h)}{|\Delta_P(x,h)| + |\Delta_N(x,h)|}$$
$$e_N^{AUC}(x,h) = e^{AUC}(x,h)\cdot\frac{\Delta_N(x,h)}{|\Delta_P(x,h)| + |\Delta_N(x,h)|} \quad (10)$$

Therefore, $e^{AUC}(x,h)$ is distributed between the positive and negative neurons according to how far each one is from their ideal value, maintaining the direction of the distance metric (Δ_P and Δ_N). This way, using eq.10, dataset elements having a perfect AUC score, where $AUC(x,h)$ is maximum, do not produce any error even if the output values of the output neurons are not the ideal ones. In the limit case, where $|\Delta_P(x,h)| + |\Delta_N(x,h)| = 0$, we establish both $e_P^{AUC}(x,h) = 0$ and $e_N^{AUC}(x,h) = 0$.

With this, we inject the proposed $\mathcal{E}^{AUC}(S,h)$ error measure by replacing $\mathcal{E}^{RMS}(S,h)$ for dataset error training algorithms(such as simulated annealing and genetic algorithms as described below), whereas $e_P^{AUC}(x,h)$ and $e_N^{AUC}(x,h)$ replace $e_P^{RMS}(x,h)$ and $e_N^{RMS}(x,h)$ respectively forelement error training algorithms (such as backpropagation and resilient backpropagation as described below). Most importantly, in practical terms, using $\mathcal{E}^{AUC}$ and e^{AUC} in existing MLP training algorithms just amounts to substituting the error calculation routine without altering the rest of the algorithm logic.

D. Efficient AUC Calculation

One drawback of using the proposed error measures is that AUC calculation is computationally expensive, since it usually requires full sorting of the measured dataset. This is specially relevant when using AUC based metrics in iterative algorithms, such as in MLP, since it may render good theoretical or experimental results impractical to use. We observed that commonly used AUC calculation techniques, such as the one provided by Weka [18], slow down about 5 times the MLP training algorithms described in next section and modified to use $\mathcal{E}^{AUC}(S,h)$ and $e^{AUC}(x,h)$ as just explained. To overcome this, we developed an efficient AUC calculation method that approximates the actual value to an arbitrary maximum error established by the user. It is based on discretizing the score space for each dataset element and, therefore, removing the need to full sorting. It produces all necessary metrics described in previous section namely, $AUC(x,h)$ and $e^{AUC}(x,h)$ for each dataset element and, of course, $AUC(S,h)$ and $\mathcal{E}^{AUC}(S,h)$ for the whole dataset.

Recalling that $h_{sc}(x) \in [0,1]$ for all elements x of a dataset, our AUC error bounded approximation method is based on the observation that, to compute the contribution of each positive element to the dataset AUC, $AUC(x,h), x \in S_P$, we are interested only on the number of negative elements whose score is lower to the score of x, regardless their actual rank. So somehow, full sorting is not totally necessary. Intuitively, our method splits the [0,1] interval into contiguous non-overlapping subintervals of equal length and counts the number of positive and negative elements whose score falls within each subinterval. This operation requires one dataset scan and elementary arithmetic. Then, for each positive element $p \in S_P$, the expression $1[h_{sc}(n) < h_{sc}(p)]$ in eq. 6 or eq. 7 is approximated by counting the number of negative elements of the subintervals under the subinterval to which p belongs. Only the negative elements falling within the same interval as p will not be counted and constitute the source of the approximation error. The finer the [0,1] interval split, the more accurate the approximation will be. This process can be iterated until the approximation error falls below a user defined value involving only dataset elements falling within intervals producing the greatest errors. With the datasets used in this work, experiments showed that with two or three further iterations the approximation error always fell under 0.001, which is good enough to ensure dispensable influence in MLP accuracy. With this approximation, MLP training algorithms were slowed down only 1.5 times in average, when comparing the RMS error based original algorithms with the AUC error modified ones.

III. EXPERIMENTAL VALIDATION

A set of experiments was set up to measure the behavior of the proposed method for AUC optimization upon existing MLP training algorithms, which were modified to use the definitions described in previous section. Experiments were carried out by using selected datasets from the UCI machine learning repository and the goal was to compare the AUC performance of the original MLP training algorithms (aiming at minimizing the error rate) against their modified versions through the same training conditions.

A. MLP Training Algorithms

Four different MLP training algorithms were used as implemented within the Encog toolkit [19], which use RMS error metrics, and modified them to use the AUC error metrics defined in previous section:

Feed Forward Back Propagation (FFBP): The classical *element error training algorithm*, where per-element error measures at each output neuron, $e_P(x,h)$ and $e_N(x,h)$, are used to adjust neuron weights of the various layers of the MLP backwards from the output layer to the input layer, through a gradient descent method controlled by two user definable parameters: the *learning rate* and the *momentum.*

Feed Forward Resilient Propagation (FFRP): Also an *element error training algorithm*, since it is a variation of FFBP where each neuron has its own set of independent parameters to control the gradient descent (similar to the FFBP learning rate and momentum) that the algorithm adjusts automatically throughout the training process.

Feed Forward Simulated Annealing (FFSA): A *dataset error training algorithm*, where an MLP is taken through several "cooling" cycles. Starting at an initial top temperature, at each step in each cooling cycle the MLP weights are randomized according to the temperature (higher temperatures produce higher random variability) generating a new MLP. If the new MLP produces a lower error on the whole dataset, $\mathcal{E}(S,h)$, it is kept to the next cooling step. Otherwise it is discarded. Then, the temperature is lowered one step and the process continues. The user definable parameters it accepts are *start-temperature*, *end-temperature* and *number-of-cycles.*

Feed Forward Genetic Algorithms (FFGA): A dataset error training algorithm, where the vector of MLP weights is interpreted as a chromosome and a population of MLPs with identical structure and different weights is evolved through generations that mate and cross over. MLPs (chromosomes) yielding lower errors on the whole dataset, $\mathcal{E}(\mathrm{S},\mathrm{h})$, are considered as best suited and, therefore, with a higher probability of survival and mating to the next generation. The user definable parameters it accepts are *population-size*, *mutation-percent* and *percent-to-mate-with*.

For a given algorithm specific values of its required training parameters constitute a training configuration. We used the same training configuration in the original and the modified algorithms to facilitate comparisons. The term "original algorithms" is therefore used for FFBP, FFRP, FFGA and FFSA as originally delivered by Encog (using RMS based error measures), and the term "modified algorithms" is used for their counterparts (FFBPROC, FFRPROC, FFGAROC and FFSAROC) adapted by the authors to use the previously defined AUC based error measures.

Since many different MLP configurations were to be evaluated for different training algorithms and datasets, a software framework (named BiomedTK [20]) was specially developed to manage their evaluation over distributed

 ISBN: 978-1-61208-154-0

computer clusters. BiomedTK stands for Biomedical Data Analysis Toolkit, and implements different kinds of dataset normalizations, validation procedures (n fold cross-validation, leave one out, etc.), distribution of configuration into jobs, management of jobs and training results, etc. This allowed us to efficiently harness computing power to evaluate a complex variety of training algorithms, datasets and training configurations by using just a few configuration artifacts, such as the exploration definition file shown in Table 3.

B. Selected Datasets and MLP configurations

Table 2 shows the datasets selected for experimentation from the UCI repository. The criteria to select those datasets was: (1) they are binary datasets, (2) they provide a diversity of skews in their class distribution, (3) they represent classification tasks of different nature and (4) they contain less than 1000 elements, which makes MLP training affordable from a computational point of view. Class skew was considered important since AUC is known to be insensitive to class distribution.

TABLE 2: BINARY UCI DATASETS USED FOR VALIDATION

dataset	elements	features	class skew
pgene	106	57	50%
E. Coli promoter gene sequences (DNA) with partial domain theory			
mmass	961	5	54%
Discrimination of benign and malignant mammographic masses based on BI-RADS attributes and the patient's age.			
heartsl	270	13	56%
Heart disease database reformatted			
liver	345	6	58%
BUPA Medical Research Ltd. database on liver desease			
bcwd	569	30	63%
Diagnostic breast cancer Wisconsin database.			
pimadiab	768	8	65%
From National Institute of Diabetes, Digestive and Kidney Diseases; Includes cost data.			
tictac	958	9	65%
Possible configurations of tic-tac-toe game.			
echocard	131	8	67%
Patients surviving for at least one year after a heart attack.			
haber	306	3	74%
Dataset contains cases from study conducted on the survival of patients who had undergone surgery for breast cancer.			
park	195	22	75%
Oxford Parkinson's Disease Detection Dataset.			
glass	214	9	76%
From USA Forensic Science Service; 6 types of glass; defined in terms of their oxide content (i.e. Na, Fe, K, etc.).			
spectf	267	44	79%
Data on cardiac Single Proton Emission Computed Tomography (SPECT) images.			

For each dataset, a set of MLP configurations was defined for each original algorithm and its modified version (FFSA/FFSAROC, FFGA/FFGAROC, FFBP/FBPROC, FFRP/FFRPROC). Table 3 shows an example configuration file used through BiomedTK to manage the exploration of training configurations for FFSA and FFSAROC MLPs with the SPECTF dataset, which results in 24 configurations for FFSA and another 24 configurations for FFSAROC. Configurations include MLPs with one or two hidden layers, with 89 or 178 neurons in the first hidden layer, with the parameter start-temperature set to 30 or 100, etc.

Similar configurations sets were defined for each algorithm and dataset, fixing the particular parameters of each training algorithm to be the same for all datasets and varying only the number of input neurons according to the dataset input features, while keeping the same proportions in the number of neurons of the hidden layers with respect to the input layer (as in the example in Table 3).

TABLE 3: FFSA/FFSAROC EXPLORATIONS FOR SPECTF DATASET.

```
explore.neurons.input                = 44
explore.neurons.output               = 2
explore.neurons.layer.01             = 89:178
explore.neurons.layer.02             = 44:132
explore.activation.function          = sigm
explore.trainingsets                 = spectf
explore.trainengines                 = encog.ffsa:encog.ffsaroc
explore.validation                   = cvfolds 10
explore.encog.ffsa.starttemp         = 30:100
explore.encog.ffsa.endtemp           = 2:10
explore.encog.ffsa.cycles            = 50
explore.encog.ffsa.stop.epochs       = 3000
explore.encog.ffsa.stop.error        = 0.0001
explore.encog.ffsaroc.starttemp      = 30:100
explore.encog.ffsaroc.endtemp        = 2:10
explore.encog.ffsaroc.cycles         = 50
explore.encog.ffsaroc.stop.epochs    = 3000
explore.encog.ffsaroc.stop.error     = 0.0001
explore.numberofjobs                 = 40
```

Each MLP configuration was trained with 10-fold cross-validation. In total, 180 MLP configurations were trained for each of the 12 datasets, 90 MLP configurations corresponding to the original algorithms and 90 to their corresponding modified versions. Overall, 2160 MLP configurations were trained on a cluster with 50 computers, which, dedicated, took about 4 full physical days. Evaluation of all these configurations over the computing resources was managed through the BiomedTK software framework as mentioned above.

C. Results

For each dataset and training algorithms, results were processed and averaged in the following way: (1) each MLP configuration was trained both with the original algorithm from Encog and its modified version; (2) AUC results from all configurations trained with the original algorithms is averaged and its standard deviation is calculated and (3) AUC results from all configurations trained with the modified algorithms is averaged and its standard deviation is calculated. The percentage of improvement of the averages (positive or negative) obtained by the modified version was calculated with respect to the original version through the following formula:

 ISBN: 978-1-61208-154-0

$$improv = 100 \times \frac{MODIFIED_{avg} - ORIGINAL_{avg}}{ORIGINAL_{avg}} \quad (11)$$

Table 4 summarizes the average AUC obtained per dataset and category of algorithm (dataset error based or element error based), aggregating them in total (column 'OVERALL'). Finally, Table 5 provides some correlation measures between different obtained measures.

TABLE 4: AUC IMPROVEMENT PER DATASET AND ALGORITHM TYPE

	OVERALL AUC			**ELMENT BASED AUC (FFBP and FFRP)**			**DATASET BASED AUC (FFSA and FFGA)**		
	orig	*mod*	*impro*	*orig*	*mod*	*impro*	*orig*	*mod*	*impro*
pgene	0,742	0,731	-1,01%	0,786	0,765	-2,44%	0,698	0,698	0,43%
mmass	0,781	0,827	6,65%	0,708	0,782	11,23%	0,855	0,872	2,07%
heartsl	0,795	0,854	7,93%	0,740	0,813	10,37%	0,849	0,894	5,48%
liver	0,622	0,707	14,59%	0,605	0,684	13,91%	0,640	0,731	15,26%
bcwd	0,900	0,903	0,20%	0,831	0,826	-0,83%	0,969	0,980	1,23%
pimadiab	0,704	0,741	5,19%	0,635	0,665	4,32%	0,774	0,818	6,07%
tictac	0,733	0,786	7,81%	0,747	0,794	6,87%	0,720	0,778	8,75%
echocard	0,566	0,623	11,01%	0,507	0,605	19,45%	0,626	0,641	2,57%
haber	0,629	0,674	7,30%	0,580	0,635	9,33%	0,678	0,714	5,28%
park	0,840	0,846	1,18%	0,800	0,804	1,08%	0,879	0,889	1,27%
glass	0,876	0,899	3,15%	0,795	0,838	5,85%	0,956	0,960	0,46%
spectf	0,673	0,712	6,25%	0,616	0,660	7,47%	0,730	0,764	5,02%
averages	0,738	0,775	5,86%	0,696	0,739	7,22%	0,781	0,812	4,49%

As it can be observed, there is a generalized AUC improvement by our proposed method, having occasional degradations in particular datasets (mostly in pgene and bcwd). The global averaged improvement is 5.86% with a great variability on each dataset and algorithm. It can also be observed that improvement is greater in element error training algorithms (FFBP and FFRP) than in dataset error training algorithms, although this might be due to the fact that these later ones tend to give better results as shown by the correlation between improvement and ORIGINAL AUC avg in Table 5 line 2 (improvement is greater when ORIGINAL AUC is lower). We acknowledge that this last observation might be biased by the way *improv* is defined (eq. 11) since greater AUC leave less room for improvement.

TABLE 5: CORRELATIONS BETWEEN EXPERIMENT MEASURE PAIRS

	FFBP	FFRP	FFSA	FFGA	ALL
improvement vs. class skew	-0,257	0,259	0,103	-0,208	-0,026
improvement vs. ORIGINAL AUC avg	-0,365	-0,872	-0,664	-0,553	-0,717
improvement vs. ORIGINAL AUC stddev	0,063	0,454	0,362	0,208	0,178
ORIGINAL AUC stddev vs. MODIFIED AUC stddev	0,883	0,631	0,884	0,828	0,682
ORIGINAL AUC avg vs. MODIFIED AUC avg	0,848	0,956	0,981	0,956	0,971

Other interesting observations are the following:

A) Both AUC averages and standard deviations are strongly correlated between the original algorithms and their modified versions (Table 5 lines 4 and 5).

B) Small standard deviations result from MLP configurations producing similar AUC scores (all configurations classify the dataset as good or as bad), whereas larger standard deviations result from some MLP configurations producing significantly better AUC scores than others. The strong correlations observed in averages and standard deviations leads to think that modified algorithms behave similarly to the original ones in the sense that they respond in the same way to dataset particularities (difficulty or easiness to classify).

C) Class skew seems to have little influence on improvement (Table 5 line 1), or at least in a non-homogeneous way across the different training algorithms.

D) Except in the case of FFBP, there seems to be a significant correlation (Table 5 line 3) between the standard deviation of the ORIGINAL AUC and the improvement obtained by our method in the positive direction (increasing standard deviation with increasing improvement). Large standard deviations may occur in many scenarios, such as when a dataset is hard to separate and well performing MLP configurations are scarce. The observed correlation might suggest that our method could be more appropriate in these situations to increase overall MLP AUC performance.

All these issues might be subject of further research, seeking stronger statistical evidence to support the hypothesis outlined.

IV. CONCLUSIONS AND FUTURE WORK

This work presented a new method to insert AUC based error metrics in existing MLP training algorithms for AUC optimization. In practical terms, the proposed approach only requires the substitution of the error computing routines of the underlying training algorithms, respecting their core logic. Experimental evidence demonstrated a consistent improvement in AUC through a variety of datasets and training algorithms requiring little coding effort. In addition, and equally important, an efficient method has been develop providing an error bound approximation for the AUC, which ensures MLP training remains computationally affordable. Finally, we can conclude that the newly developed AUC error metrics show a consistent behavior in both its theoretical definition and experimental results.

Future work intends to further validate this approach in other kinds of machine learning algorithms and explore its theoretical implications, specially in those algorithms requiring continuity conditions on the loss or error functions (such as gradient descent). Also stronger statistical evidence for the hypotheses outlined in previous section will be pursued.

ACKNOWLEDGMENTS

This work is part of the GRIDMED research collaboration project between INEGI (Portugal) and CETA-CIEMAT (Spain). Prof. Guevara acknowledges POPH - QREN-Tipologia 4.2 – Promotion of scientific employment funded by the ESF and MCTES, Portugal. CETA-CIEMAT acknowledges the support of the European Regional Development Fund.

 ISBN: 978-1-61208-154-0

REFERENCES

[1] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters,* vol. 27, pp. 861-874, 2006.

[2] M. Zweig and G. Campbell, "Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine " *Clinical Chemistry,* vol. 39, pp. 561-577, 1993.

[3] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve," *Radiology,* vol. 143, pp. 29-36, 1982.

[4] J. Iavindrasana*, et al.*, "Clinical data mining: a review," *Yearb Med Inform,* pp. 121-33, 2009 2009.

[5] T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Researchers," HP Labs Tech Report HPL-2003-4, 2003.

[6] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition,* vol. 30, pp. 1145-1159, 1997.

[7] C. Cortes and M. Mohri, "AUC optimization vs. error rate minimization," *Advances in Neural Information Processing Systems 16,* vol. 16, pp. 313-320, 2004.

[8] U. Brefeld and T. Scheffer, "AUC Maximizing Support Vector Learning," *Proceedings of the ICML Workshop on ROC Analysis in Machine Learning,* 2005.

[9] T. Calders and S. Jaroszewicz, "Efficient AUC optimization for classification," *Knowledge Discovery in Databases: PKDD 2007, Proceedings,* vol. 4702, pp. 42-53, 2007.

[10] L. Costa*, et al.*, "Tuning Parameters of Evolutionary Algorithms Using ROC Analysis," in *2nd International Workshop on Practical Applications of Computational Biology and Bioinformatics (IWPACBB 2008)*. vol. 49, J. Corchado*, et al.*, Eds., ed: Springer Berlin / Heidelberg, 2009, pp. 217-222.

[11] F. Provost and T. Fawcett, "Robust Classification for Imprecise Environments," *Machine Learning,* vol. 42, pp. 203-231, 2001.

[12] C. Marrocco*, et al.*, "Maximizing the area under the ROC curve by pairwise feature combination," *Pattern Recogn.,* vol. 41, pp. 1961-1974, 2008.

[13] C. L. Castro and A. P. Braga, "Optimization of the Area under the ROC Curve," in *Neural Networks, 2008. SBRN '08. 10th Brazilian Symposium on*, 2008, pp. 141-146.

[14] K. A. Toh*, et al.*, "Maximizing area under ROC curve for biometric scores fusion," *Pattern Recognition,* vol. 41, pp. 3373-3392, Nov 2008.

[15] A. K. S. Wong*, et al.*, "Improving text classifier performance based on AUC," *18th International Conference on Pattern Recognition, Vol 3, Proceedings,* pp. 268-271, 2006.

[16] G. Han and C. Zhao, "AUC maximization linear classifier based on active learning and its application," *Neurocomputing,* vol. 73, pp. 1272-1280, 2010.

[17] R. Ramos-Pollan*, et al.*, "Introducing ROC curves as error measure functions. A new approach to train ANN-based biomedical data classifiers," in *15th Iberoamerican Congress on Pattern Recognition*, Sao Paolo, Brasil, 2010.

[18] Mark Hall*, et al.*, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations,* vol. 11, 2009.

[19] J. Heaton, "Programming Neural Networks with Encog 2 in Java," ed: Heaton Research, Inc., 2010.

[20] R. Ramos-Pollán et al., "A Software Framework for Building Biomedical Machine Learning Classifiers through Grid Computing Resources," *Journal of Medical Systems*, 1-13, DOI 10.1007/s10916-011-9692-3

 ISBN: 978-1-61208-154-0