Introducing ROC Curves as Error Measure Functions: A New Approach to Train ANN-Based Biomedical Data Classifiers

Raúl Ramos-Pollán[1], Miguel Ángel Guevara-López[2], and Eugénio Oliveira[3]

[1] CETA-CIEMAT Centro Extremeño de Tecnologías Avanzadas,
Calle Sola 1, 10200 Trujillo, Spain,
raul.ramos@ciemat.es
[2] INEGI Instituto de Engenharia, Mecanica e Gestão Industrial, Universidade do Porto,
Campus da FEUP, Rua Roberto Frias 400, 4200-465 Porto, Portugal
mguevaral@inegi.up.pt
[3] LIACC-DEI-Faculdade de Engenharia, Universidade do Porto, Rua Roberto Frias s/n,
4200-465 Porto, Portugal
eco@fe.up.pt

Abstract. This paper explores the usage of the area (Az) under the Receiver Operating Characteristic (ROC) curve as error measure to guide the training process to build machine learning ANN-based classifiers for biomedical data analysis. Error measures (like root mean square error, RMS) are used to guide training algorithms measuring how far solutions are from the ideal classification, whereas it is well known that optimal classification rates do not necessarily yield to optimal Az's. Our hypothesis is that Az error measures can guide existing training algorithms to obtain better Az's than other error measures. This was tested after training 280 different configurations of ANN-based classifiers, with simulated annealing, using five biomedical binary datasets from the UCI machine learning repository with different test/train data splits. Each ANN configuration was trained both using the Az and RMS based error measures. In average Az was improved in 7.98% in testing data (9.32% for training data) when using 70% of the datasets elements for training. Further analysis reveals interesting patterns (Az improvement is greater when Az are lower). These results encourage us to further explore the usage of Az based error measures in training methods for classifiers in a more generalized manner.

Keywords: ROC Curves, Artificial Neural Networks, Machine learning Classifiers, Biomedical Data.

1 Introduction

After preliminary data preparation, pattern recognition systems consist of two major stages: (1) feature extraction and selection and (2) classification. This means that a set of features is extracted from the pattern to be recognized and then classified into one of the possible classes. To achieve high recognition accuracy, the feature extractor is

I. Bloch and R.M. Cesar, Jr. (Eds.): CIARP 2010, LNCS 6419, pp. 517–524, 2010.

required to discover salient characteristics suited for classification and the classifier is required to set class boundaries accurately in the feature space. Progress made in sensor technology and data management allows researchers to gather datasets of ever increasing sizes [1].

The integration of biomedical information has become an essential task for health care, biology and biotechnology professionals and researchers. Integration is therefore much more than a plain collection of digital biomedical data. Homogenization of data description and storage, followed by normalization across the various experimental conditions would be a prerequisite to enable procedures of knowledge extraction [2].

The area under a ROC curve (or *Az* [3]) is a decisive factor used in many applications to measure classifier quality (performance). However, it is known that optimal classification rates do not necessarily yield to optimal Az's [4] and a few attempts have tried to use Az in optimization problems [5-6]. This paper explores the usage of the ROC Az as error measure to guide the training process to build machine learning ANN-based classifiers for biomedical data analysis. Our hypothesis is that by doing this, we will obtain better Az's than those obtained through other error measures.

This paper is structured as follows. Section 2 establishes the theoretical background of this work. Section 3 describes the technological framework used to experimentally validate our hypothesis on a Grid infrastructure. Section 4 describes the experiments performed and Section 5 discusses the results obtained. Section 6 draws some conclusions and outlines future work.

2 ROC Az Based Error Measures

2.1 ANNs Trained with Simulated Annealing

Simulated annealing was first proposed in [9] and it is inspired by the physical process of annealing to find good values of functions depending on many parameters. In short, the algorithm includes a parameter, simulating temperature starting at a given value, which is lowered gradually at known steps. For each temperature, the function parameters are randomized and the range of possible values that they can take is proportional to the temperature, so that at lower temperatures that range is smaller. At each temperature step the process is repeated a predetermined number of times and the set of parameters giving the best function value are retained and passed on to the following cycle iteration.

We use the simulated annealing approach to train ANNs as described and implemented in [7] which the weights of an ANN population are randomized iteratively and, at each step, the ANN with the minimum root mean square (RMS) error is retained. Notice that, in this case, the error used by the algorithms is the RMS of the whole training set, as opposed to other algorithms such as backpropagation, where it is the individual RMS error of each element of the training set with respect to the output neurons the one that is used.

More formally, for binary classifiers, we use the following definitions:

Table 1. Definitions

$\mathcal{X} \subset \mathbb{R}^p$	Domain of input vectors (with p features)
$\mathcal{Y} = \{-1,1\}$	The two classes into which input vectors are classified
$(x_i, y_i)\ \ x_i \in \mathcal{X}, y_i \in \mathcal{Y}$	Input vector with its associated class (for supervised training)
$S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$	Training set (for supervised training)
$\lvert S \rvert = n$	Size of training set
$\mathcal{F} = \{\, f: \mathcal{X} \to \mathcal{Y}\ \}$	Set of functions representing binary classifiers
$h \in \mathcal{F}$	A binary classifier
$h(x_i) \in \mathcal{Y}$	Output of binary classifier h when applied to input vector x_i, $h(x_i)$ typically applies some threshold notion to $h_{score}(x_i)$ to obtain the final class assigned to x_i
$h_{score}(x_i) \in \mathbb{R}$	Score assigned by binary classifier h to input vector x_i,
$\mathcal{E}(S,h)$	A global error measure of classifier h when applied to training set S
$e(x_i,h)$	An individual error measure of classifier h when applied to input vector x_i
$Az(S,h)$	Area under the ROC curve of training set S when classified with classifier h

In particular, an RMS error measure is typically defined as follows:

$$\mathcal{E}_{RMS}(S,h) = \frac{\sum e_{RMS}(x_i,h)}{|S|} \tag{1}$$

where $e_{RMS}(x_i,h)$ represents some distance measure between $h_{score}(x_i)$ and y_i, possibly using the output values of the output neurons in case of ANN based classifiers. Algorithms such as backpropagation in ANN based classifiers use the individual values $e_{RMS}(x_i,h)$ iterating through each element of the training set to incrementally correct the ANN weights, whereas simulated annealing uses only the global $\mathcal{E}_{RMS}(S,h)$ value to select the best classifier at each cycle of each cooling step.

2.2 ROC Az Error with Simulated Annealing

We now use ROC Az to define a global error measure as follows:

$$\mathcal{E}_{ROC}(S,h) = 1 - Az(S,h) \tag{2}$$

and use this definition, instead of (1), as error measure in the simulated annealing based ANN training process described above. $\mathcal{E}_{ROC}$ is implemented in the *ffsaroc* engine included in **Biomedtk** (see Section 3) whereas $\mathcal{E}_{RMS}$ is used in the *ffsa* engine.

Note that when using RMS there is a direct relation between the individual error measures of the elements of the training set (e_{RMS}) and the global error ($\mathcal{E}_{RMS}$) which is given by equation (1) and this is why it can be used by backpropagation-like

training algorithms, whereas there is no such direct relation in $\mathcal{E}_{ROC}$ since $Az(S,h)$ is a global measure of a classified set. It is the fact that simulated annealing does not use individual error measures for each element of the training set that allows us to replace $\mathcal{E}_{RMS}$ by $\mathcal{E}_{ROC}$ in a straight forward manner.

3 The Biomedtk Framework

The Biomedical Data Analysis Toolkit (**Biomedtk**) is a Java software tool developed by the authors that exploits existing libraries for data analysis with methods and metrics commonly used in the biomedical field. In addition, it provides the means to massively search, explore and combine different configurations of data classifiers provided by the underlying libraries to build robust data analysis tools. With this, it is possible manipulate datasets, train Artificial Neural Networks (ANN) based binary and multiclass classifiers with many different configurations, search for best ensemble classifiers, generate different types of ROC curve analysis, etc.

An ANN-based configuration specifies a certain network structure (number of layers and neurons per layer), a training algorithm to use (such as backpropagation or simulated annealing) and algorithm dependant train parameters (such as learning rate, start/end temperatures, etc.). **Biomedtk** allows defining explorations of ANN configurations (see Section 4) and sending them for massive training to a Grid computing infrastructure. Currently, **biomedtk** supports training engines from the Encog [7] and Weka [8] toolkits, as listed in table 2.

Table 2. Biomedtk supported training engines

Engine name	Description
ffbp	Feedforward with backpropagation-based training.
ffga	Feedforward with genetic algorithm-based training.
ffsa	Feedforward with simulated annealing-based training.
ffsaroc	ffsa with WEKA ROC based error evaluation.
Rb	Radial basis.
som	Self-organizing feature map.

It also includes the *ffsaroc* which is a modification of the *ffsa* engine including the ROC Az based error evaluation described in Section 3. **Biomedtk** uses the Mann-Whitney statistic to calculate ROC Az as implemented in Weka[8].

4 Experimental Setup

A set of experiments was set up in order to test whether the error measure proposed in Section 2 effectively improves the Az of the trained ANN classifiers. The tests were carried out using the binary biomedical UCI datasets [10] listed in table 3.

Table 3. UCI Datasets used in the experiments

Dataset	Description	# elements
Haber	Survival of patients with breast cancer surgery	306
Liver	Liver disorders from excessive alcohol consumption	345
Mmass	Benign/malign mammographic masses	961
Pimadiab	Diabetes diagnoses for Pima Indian populations	768
Spectf	Data on cardiac Single Proton Emission Computed Tomography (SPECT) images	267

For each UCI dataset we generated two dataset splits, one using 50% of the dataset for training and 50% for testing, and one using 70% for training and 30% for testing. A **Biomedtk** exploration was defined for each dataset split with ANN configurations having one to three hidden layers and different classifier parameters. Each dataset split and configuration was then trained with both the *ffsa* and the *ffsaroc* engines. This is an extract of the exploration definition file for the two *haber* dataset splits:

```
explore.neurons.input       = 3
explore.neurons.output      = 2
explore.neurons.layer.01    = 8:18
explore.neurons.layer.02    = 8:18
explore.neurons.layer.03    = 3:8
explore.trainingsets        = haber-30:haber-50
explore.trainengines        = ffsa:ffsaroc
explore.stop.epochs         = 200
explore.stop.error          = 0.0001
explore.starttemp           = 10:20
explore.ffsa.endtemp        = 2
explore.ffsa.cycles         = 100
```

This exploration includes ANNs with one, two or three hidden layers, where the first and second hidden layers may have 8 or 18 neurons, with a start temperature of 10 or 20, etc. For each dataset, this exploration generates 112 ANN configurations, 56 using the *ffsa* engine and 56 using the *ffsaroc* engine. In total, for all datasets, 560 ANN configurations were trained on a gLite [11] Grid infrastructure using **Biomedtk** and consuming about 140 CPU hours.

Finally, to check whether each *ffsa* configuration is improved by using *ffsaroc* instead, we compared each ANN configuration trained with the *ffsa* engine with the same configuration (same number of hidden layers, neurons and parameters) but trained with the *ffsaroc* engine. We name such pair a *classifier pair*. Therefore 56 classifier pairs are made for each dataset, 28 for the 50/50 test/train data split and 28 for the 30/70 split.

5 Results and Discussion

Table 4 below summarizes the average improvement obtained by *ffsaroc* over *ffsa* classifier pairs for each data set both for training and testing data. Results are also shown for the 30/70 and 50/50 test/train split for each dataset.

Table 4. FFSAROC improvement of FFSA per training set

	test Az increase		**train Az increase**		**avg test Az**	
Dataset	*avg*	*std dev*	*avg*	*std dev*	*ffsaroc*	*ffsa*
haber-30	7.92%	5.58	6.90%	1.53	0.691	0.642
haber-50	2.33%	3.94	8.25%	2.13	0.729	0.713
liver-30	3.91%	3.81	4.50%	1.94	0.698	0.672
liver-50	0.98%	2.46	7.60%	2.99	0.725	0.718
mmass-30	3.24%	5.43	6.83%	3.27	0.807	0.782
mmass-50	-0.59%	4.94	7.46%	2.78	0.786	0.790
pimadiab-30	5.44%	4.42	7.91%	3.88	0.704	0.669
pimadiab-50	1.45%	6.69	11.83%	5.35	0.691	0.684
spectf-30	19.38%	20.57	20.45%	8.43	0.830	0.712
spectf-50	8.34%	17.35	25.85%	16.40	0.757	0.711
total 30% test sets	7.98%	11.77	9.32%	7.27	0.746	0.695
total 50% test sets	2.50%	9.37	12.20%	10.62	0.738	0.723
total all datasets	**5.24%**	**10.98**	**10.76%**	**9.21**	**0.742**	**0.709**

Table 5 shows the same classifier pairs grouped by *ffsa testAz* value ranges.

Table 5. FFSAROC improvement over FFSA per FFSA Test Az range

FFSA Test Az range	**Number of Classifier Pairs**	**test Az increase**		**train Az increase**	
		avg	*std dev*	*avg*	*std dev*
(0.00, 0.60]	8	51.16%	17.31	43.81%	20.36
(0.60, 0.65]	30	12.94%	7.12	10.73%	6.13
(0.65, 0.70]	82	5.17%	6.77	9.10%	7.01
(0.70, 0.75]	81	2.26%	5.39	9.39%	5.34
(0.75, 0.80]	67	1.28%	5.90	9.80%	6.79
(0.85, 1.00]	12	-1.96%	5.65	14.76%	6.60
total all ranges	**280**	**5.24%**	**10.98**	**10.76%**	**9.21**

As it can be seen in the tables above there is in general a significant average improvement in Az both for train and test parts of the datasets. Note that the improvement percentages shown are the average of the compared classifier pairs (between *ffsaroc* and *ffsa* trained classifiers with the same network structure and training parameters), which is different from the average *testAz* of **all** *ffsaroc* and *ffsa* classifiers in each training set shown in the last two columns of table 4.

Improvement is constantly better when splitting datasets in 30/70 test/train data than when splitting in 50/50. Even if the 30/70 datasets have better average *testAz* than 50/50 (such as in the *mmass*, *pimadiab* and *spectf* datasets). Improvement is also constantly better in the train parts of the dataset than in the test parts and, in any case, its standard deviation is always high. This seems to indicate that the method presented in this paper may be hard for classifiers to generalize and does not behave homogeneously for all classifiers. From table 5, it can be clearly seen that for test data, improvement is better when the *testAz* for the *ffsa* classified dataset is worse. Finally, the

following plots show the ROC curves of the classifier pairs for which the best improvement was obtained (*ffsaroc* over *ffsa*) for the *haber* and *spectf* datasets both for the 30/70 and 50/50 data splits

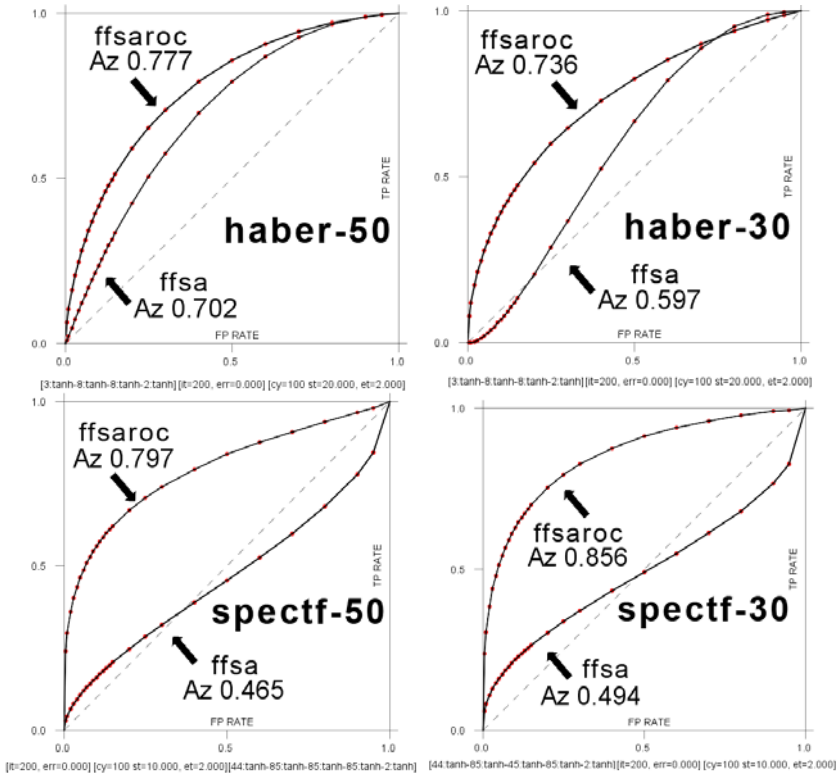

Fig. 1. Most improved classifiers for *spectf* and *haber*, 30% and 50% test data splits

To simplify visual comparison, these plots use the bi-normal distribution method as provided by JLABROC [12] which is also supported by **Biomedtk**. Also, each plot shows the ANN layers structure (neurons per layer and activation function) and the training parameters for the simulated annealing processes (start temperature, end temperature and number of cycles)

6 Conclusions

The experimental results obtained here confirm that the usage of a ROC Az based error function to guide a simulated annealing algorithm for training ANNs improves the ROC Az of the obtained classifiers with respect to an RMS error function. In addition, **Biometk** demonstrated to be a robust framework to massively explore large amounts of configurations of data classifiers exploiting computing power harnessed by Grid infrastructures.

Future work is focused on (1) better understanding of the behavior of the proposed method to better explain deviations observed in the experiments and (2) on applying the method in a generalized manner to machine learning classifiers and validate them in real computer-aided detection/diagnosis systems.

Acknowledgements

Prof. Guevara acknowledges POPH - QREN-Typology 4.2 – Promotion of scientific employment funded by the ESF and MCTES, Portugal. Prof. Ramos-Pollán acknowledges the support of the European Regional Development Fund.

References

1. Kostka, P., Tkacz, E.J.: Feature extraction and selection algorithms in biomedical data classifiers based on time-frequency and principle component analysis. In: Proc. 11th Mediterranean Conference on Medical and Biomedical Engineering and Computing 2007, vol. 16, pp. 70–73. Springer, Heidelberg (2007)
2. Drakos, J., Karakantza, M., Zoumbos, N., Lakoumentas, J., Nikiforidis, G., Sakellaropoulos, G.: A perspective for biomedical data integration: Design of databases for flow cytometry. BMC Bioinformatics 9(1), 99 (2008)
3. Fawcett, T.: An introduction to ROC analysis. Pattern Recognition Letters 27(8), 861–874 (2006)
4. Castro, C.L., Braga, A.P.: Optimization of the Area under the ROC Curve. In: Proc. of 10th Brazilian Symposium on Neural Networks, SBRN 2008, pp. 141–146 (2008)
5. Cortes, C., Mohri, M.: AUC optimization vs. error rate minimization. In: Advances in Neural Information Processing Systems. MIT Press, Cambridge (2003)
6. Rakotomamonjy, A.: Optimizing Area under ROC Curve with SVMs. In: Proc. Workshop of ROC Analysis in Artificial Intelligence, pp. 71–80. ROCAI (2004)
7. Heaton, J.: Programming Neural Networks with Encog 2 in Java. Heaton Research, Inc. (2010)
8. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P.: Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations 11(1) (2009)
9. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science 220(4598), 671–680 (1983)
10. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine, CA (2007), http://www.ics.uci.edu/~mlearn/MLRepository.html
11. EGEE: The gLite middleware, vol. 2010 (2009)
12. John Eng, M.D.: ROC analysis: web-based calculator for ROC curves, vol. 2010. Johns Hopkins University, Baltimore (2006)