J Med Syst
DOI 10.1007/s10916-011-9692-3

ORIGINAL PAPER

A Software Framework for Building Biomedical Machine Learning Classifiers through Grid Computing Resources

Raúl Ramos-Pollán · Miguel Ángel Guevara-López · Eugénio Oliveira

Received: 14 December 2010 / Accepted: 28 March 2011

Abstract This paper describes the BiomedTK software framework, created to perform massive explorations of machine learning classifiers configurations for biomedical data analysis over distributed Grid computing resources. BiomedTK integrates ROC analysis throughout the complete classifier construction process and enables explorations of large parameter sweeps for training third party classifiers such as artificial neural networks and support vector machines, offering the capability to harness the vast amount of computing power serviced by Grid infrastructures. In addition, it includes classifiers modified by the authors for ROC optimization and functionality to build ensemble classifiers and manipulate datasets (import/export, extract and transform data, etc.). BiomedTK was experimentally validated by training thousands of classifier configurations for representative biomedical UCI datasets reaching in little time classification levels comparable to those reported in existing literature. The comprehensive method herewith presented represents an improvement to biomedical data analysis in both methodology and potential reach of machine learning based experimentation.

Keywords Machine learning classifiers · Biomedical data analysis · ROC analysis · Grid infrastructures

R. Ramos-Pollán (✉)
CETA-CIEMAT Centro Extremeño
de Tecnologías Avanzadas,
Calle Sola 1,
10200 Trujillo, Spain
e-mail: raul.ramos@ciemat.es

M. Á. Guevara-López
INEGI-Faculdade de Engenharia,
Universidade do Porto,
Rua Roberto Frias 400,
4200–465 Porto, Portugal
e-mail: mguevaral@inegi.up.pt

E. Oliveira
LIACC-DEI-Faculdade de Engenharia,
Universidade do Porto,
Rua Roberto Frias s/n,
4200–465 Porto, Portugal
e-mail: eco@fe.up.pt

Introduction

The integration of biomedical information has become an essential task for biology, biotechnology and health care professionals. Current progress in Information Technologies allows for affordable huge data storages and powerful computational possibilities; thus, they have become quite common. Researchers are gradually becoming aware of the importance of keeping together diverse data pertaining to a specific medical entity and successful attempts to create and maintain such databases are becoming known to the scientific community [1–4]. Data models specified by standards are often included in databases, without taking into account inherent limitations posed by the procedure of acquiring original data. 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 for facilitating procedures of knowledge extraction and analysis [2].

Machine learning classifiers (MLC) are able to learn from experience (observed examples) with respect to some class of tasks and a performance measure. Due to this, they

Published online: 09 April 2011

are suitable for biomedical data classification, based on learning algorithms' ability to construct classifiers that can explain complex relationships in the data. MLC constitute the backbone of biomedical data analysis on high dimensional quantitative data provided by the state-of-the-art medical imaging and high-throughput biology technologies. The general strategy relies on expert-curated ground truth datasets providing the categorical associations of all available data samples. Ground truth datasets are then used as the basis for statistical learning to construct classifiers using one of a host of methods such as support vector machines (SVM), artificial neural networks (ANN), nearest neighbor classifiers, discriminant functions and so on [5]. Machine learning techniques have been successfully applied in various biomedical domains, for example the detection of tumors, the diagnosis and prognosis of cancers and other complex diseases [6–8].

From a computational point of view MLC are expensive to train, and seamless access to computing power is key to undertake successfully any comprehensive MLC based data analysis endeavor. Grid infrastructures federate distributed computing resources owned by different institutions to offer one large computing and storage facility, bringing together resources scattered throughout different data centers through a *middleware* layer (such as Globus [9] or gLite [10]) providing a homogeneous view and access model to the final users. In the last years there has been great public investment to build transnational Grid facilities providing generalized access to large amounts of computing resources for scientific communities to perform their research (eScience). The European Grid Initiative (EGI, [11]) is the major public European effort aggregating the national Grids of virtually every EU country to conform a pan-European federation of approximately 330 data centers and 70,000 CPU cores [12]. However, in spite of having been around for the last 10 years, Grids are still far from achieving the initial expectations of penetration and dissemination throughout scientific disciplines and business domains, remaining mostly within specific academic areas. Among others, one of the reasons behind this is the difficulty of usage of the middleware, constituting a steep learning curve and cost barrier for new communities not having the tradition nor the resources to work with Grids [13–16]. In fact, the lifecycle through which new applications are adapted to use existing middleware is long and slow and this has caused even the more privileged scientific communities (such as High Energy Physics, focused at CERN, the European Laboratory for Particle Physics) to develop their own particular tools and methods to reduce usage costs and facilitate their users' research, such as, among others, DIRAC [17] and AliEn [18] used respectively by the CERN LHCb and ALICE experiments.

In our previous works we proposed some approaches related with the usage of distributed Grid infrastructures to support large repositories of medical images (mostly mammography) and preliminary explorations of machine learning classifiers for breast cancer computer-aided diagnosis (CAD) [1, 3, 19–21]. Now, we set forth to take advantage of Grid power in a systematic manner to discover well performing machine learning classifiers for biomedical data mining, aiming at reducing the cost and simplifying development lifecycles of using Grid infrastructures for biomedical applications. In particular, we are interested in breast cancer diagnosis (CADx) applications, and test them with the breast cancer digital repository currently being gathered at Hospital São João - Faculty of Medicine of Porto University, in Portugal, in the context of the current collaboration project between our institutions.

This paper is structured as follows. Section 2 describes the BiomedTK software framework and the MLC exploration methods it supports. BiomedTK was developed to harness Grid computing power for biomedical data analysis. It also describes the experimental set up through which the method was validated with different biomedical datasets of the UCI machine learning repository [22, 23]. Section 3 shows the results obtained by the experiments carried out, which are finally discussed in Section 4.

Materials and methods

Biomedical data analysis toolkit (BiomedTK)

BiomedTK is a Java software tool that uses distributed Grid computing power to exploit third party libraries for data analysis augmenting them with methods and metrics commonly used in the biomedical field. It provides the means to massively search, explore and combine different configurations of machine learning classifiers provided by underlying libraries to build robust biomedical data analysis tools. BiomedTK trains Artificial Neural Networks (ANN) and Support Vector Machines (SVM) based binary and multiclass classifiers with many different configurations, searches for best ensemble classifiers, generates different types of ROC (Receiver Operating Characteristic) curve analysis, etc. In addition, it is possible to manipulate datasets, including export/import to/from data formats of commonly used applications, allowing users to feed BiomedTK with datasets preprocessed by other tools to, for instance, filter, or transform the data, normalize it, reduce its dimensionality, etc. For researchers, it offers a command line interface to access its functionality (manage datasets, launch MLC explorations, analyze results, etc.). For programmers, it offers a simple API (Application Programming Interface) so that new data analysis engines

can be integrated in a modular manner with reasonable effort.

Currently, BiomedTK integrates engines from the Encog toolkit [24] (for ANNs), the libsvm toolkit [25] (for SVMs) and the ANN Encog engines modified by the authors for ROC optimization [23]. Table 1 lists the engines currently integrated in BiomedTK along with the parameters each one accepts. Particular values of those parameters for a specific engine constitute a *classifier configuration* for that engine. For a given data analysis task in hand, classifier design amounts to finding configurations yielding acceptable performance results and, therefore, the researcher is often confronted with the need to explore and evaluate several classifier configurations. As mentioned, MLC are usually computationally expensive to train and so are, in a higher degree, explorations of MLC configurations. For any engine, BiomedTK allows the evaluation of binary classifiers through plotting ROC curves and computing their area under the curve (ROC Az) [26, 27], offering the bi-normal distribution method as provided by JLABROC4 [28] and the Mann–Whitney statistic provided by WEKA [29]. It also includes the Encog ANN engines modified by the authors [23], allowing optimization of the ROC Az complementing the classical optimization tasks aimed at reducing some Root Mean Squared Error measure to increase accuracy (percentage of dataset elements correctly classified). Accuracy and ROC Az measures are related, but it is known that optimization of accuracy does not necessarily yield optimization of ROC Az [30], fact that is specially relevant for biomedical data analysis. The following summarizes the basic elements of BiomedTK:

Dataset or training set Contains the data to analyze. A dataset is made of a set of elements (or instances), each one containing a set of features used as input values for classifiers and, optionally, an ideal class to which it belongs for supervised training (expected classifier output). Depending on validation strategies, elements of a dataset are usually split into two subsets, one for training and one for testing. The training subset is used in training classifiers built to distinguish automatically the classes to which each element belongs. The test subset is used to measure the generalization capabilities of classifiers when applied to unseen data.

Binary dataset A dataset whose elements belong to only two classes. As opposed to a Multiclass Dataset, whose elements may belong to several (more than two) classes. BiomedTK provides the tools to create multiple binary training sets for a given multi-class training set, each one to build a classifier specific for each class.

Engine Engines encapsulate third party classifiers (such as ANNs from Encog or SVMs from libsvm). Each engine accepts a different set of parameters for training (ANN specification, learning parameters, etc.)

Engine or classifier configuration An engine configuration specifies the parameters with which a particular engine is used to train given a dataset. For instance, a configuration of an ANN feedforward backpropagation engine might specify an ANN with 3 layers having 10 neurons each, using the sigmoid activation function, with 0.1 as learning rate and 0.5 as momentum over 10,000 iterations.

Exploration An exploration over a dataset defines a set of engine configurations to train in batch mode in order to find the one (s) that best classify the dataset, or to later use them to build ensemble classifiers.

Jobs Each exploration is split into a number of user defined jobs. Each job will train a subset of the engine configurations defined in a given exploration. Jobs can be then

Table 1 Currently available engines in BiomedTK

engine name	description	accepted parameters	source
ffbp	feed forward ANN trained with backpropagation	ANN structure, learn rate, momentum	encog
ffbproc	ffbp modified for ROC optimization	ANN structure, learn rate, momentum	modified encog
ffrp	feed forward ANN trained with resilient propagation	ANN structure (no more params)	encog
ffrproc	ffrp modified for ROC optimization	ANN structure (no more params)	modified encog
ffsa	feed forward ANN trained with simulated annealing	ANN structure, start temp, end temp, cycles	encog
ffsaroc	ffsa modified for ROC optimization	ANN structure, start temp, end temp, cycles	modified encog
ffga	feed forward ANN trained with genetic algorithms	ANN structure, population size, mating size, survival rate	encog
ffgaroc	ffga modified for ROC optimization	ANN structure, population size, mating size, survival rate	modified encog
csvc	cost based Support Vector Machine	kernel, cost, degree, gamma, coef0, weight, shrink, probestimates	libsvm
nusvc	*v* Support Vector Machine	kernel, nu, degree, gamma, coef0, shrink, probestimates	libsvm

executed in sequentially over the same computer or in parallel over a distributed computing infrastructure.

BiomedTK interfaces seamlessly with distributed computing resources serviced by gLite Grid infrastructures (such as EGI) to train classifier configurations as described in section 2.2. It offers researchers the capability to exploit Grid computing power in an agile manner, allowing them to gradually gain understanding on what engine configurations better classify their biomedical data. This constitutes the basis of the exploration method described below.

Finally, it also offers different validation methods when using any of the available engine configurations. These are: fixed test/train dataset split, cross-validation with user definable number of folds, bootstrapping [31, 32] and leave-one-out validation [33], referred to as *one-to-all* in BiomedTK. *One-to-all* validation is in general seldom used since it requires one full classifier training process for each dataset element and engine configuration which makes it extremely demanding computationally. With BiomedTK *one-to-all* becomes a feasible and manageable alternative provided the required computing resources are available in the underlying Grid service used.

Software architecture

Figure 1 shows the different components of the BiomedTK architecture. The *BiomedTK Engine* coordinates all functionality invoking the *Exploration Generator* to process explorations and the *JobSet Generator* to effectively execute the explorations. The *JobSet Generator* handles jobs to a *Local Job Launcher* (on the user computer) or to a *gLite Job Launcher* according to the user command. Datasets and exploration results are stored in a common database. Jobs, whether launched locally or over gLite, feed the BiomedTK database with results so that they can be later inspected through standard SQL tools. Any JDBC compliant database can be used and, for convenience, BiomedTK embeds the H2 database engine [34] which uses the local file system for storage, so there is no need to install another third party database if it is not desired.

An *Engine API* allows the integration into BiomedTK of new classifier engines. Java classifier engines must deliver their implementation as a jar file containing the interface classes implementing the BiomedTK *Engine API* and their own classes (the actual engine). The following shows an excerpt of the its main interface that each classifier engine must implement:

```
public interface TrainedClassifier {

    public void initialize (ClassifierParameters p, Dataset d);

    public String getEncodedEngine();

    public void setEncodedEngine(String encodedEngine);

    public ClassifierParameters getParameters();

    public Double getTrainError();

    public void train(TrainListener l);

    public void ClassifierParameters[] generateParameterSweep(ExploreProperties p);

    public void classify (List<DatasetElement> e);

}
```

It is a straight forward Java interface requiring mostly implementations of methods to train and classify datasets, to generate parameters sweeps from an exploration definition and create and restore a trained engine to/from a String based representation, which allows BiomedTK to store and reconstruct any third party trained engine. BiomedTK also supports native implementations of classifier engines. In this case, classifier engines must deliver a set of precompiled binaries for different hardware platforms so that BiomedTK can find the required specific binary program whenever needed and it will be able to use engines for as many hardware platforms as precompiled binaries are provided. This is the case of the already integrated *libsvm* C library, for which binaries for Windows XP/7, MacOS and different Ubuntu and Scientific Linux kernels were generated. This allows researchers, for instance, to start using BiomedTK to classify a given dataset on their desktop MacOS or Windows machine and then send larger explorations to a gLite Grid service, usually running on Scientific Linux machines, in a completely transparent manner.

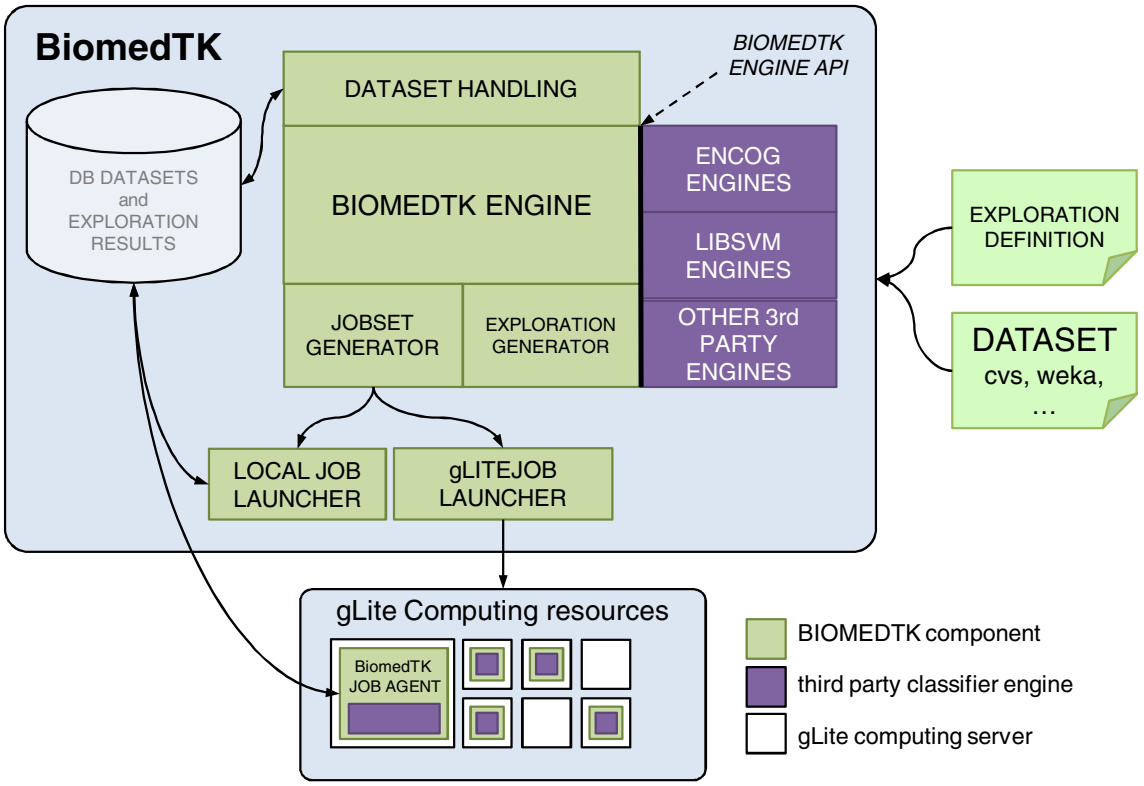

Fig. 1 BiomedTK architecture

As mentioned in section 1, Grid infrastructures are difficult to use in an agile manner as required by the diversity and complexity of biomedical datasets and explorations of MLC configurations. BiomedTK compensates this by hiding middleware complexities from the final user and by using a fast communication mechanism with gLite computing resources through a *Job Agents* based mechanism (such as in [17] and [18]), providing the means to make classifier explorations, analyze their results, define finer grain explorations, etc. in an efficient way adapted to the processes of analyzing biomedical data. In total, BiomedTK is approximately made of 18,000 lines of code and 200 Java classes.

A method to explore classifier configurations

The classifier exploration method based on BiomedTK and herewith described, is aimed at addressing two issues: 1) the need to manage, in a simple and unified way, many classifier configurations integrating third party engine implementations of different nature (implementing different MLC models such as ANNs, SVMs, etc. on different programming languages such as C, Java, etc.) and 2) the need to efficiently exploit distributed resources required to evaluate such a diversity of classifier configurations. In this sense, *efficiency* denotes both an economy in the researcher's effort to setup and run experiments and a rational usage of available computing resources. BiomedTK supports this method through a series of configuration artifacts and tools that jointly constitute the material means through which researchers can efficiently use Grid computing resources to perform complex MLC explorations.

Explorations of classifier configurations are the key element of this method. Through them, researchers can gradually build understanding on the search space of possible classifiers (engine configurations) for a given dataset. With BiomedTK explorations are defined in text files with a specific format. Figure 2 shows an example exploration file producing configurations to train classifiers for two datasets derived from the original BREAST-TISSUE one from the UCI repository [22] with the engine using *ffrp*, *ffsaroc*, *ffga* and *nusvc* (see Table 1). For ANN based engines, this exploration will generate configurations with ANNs having three hidden layers with 18 or 36 neurons in the first hidden layer, 9 or 18 in the second one, different neuron activation functions, etc. It also contains classifier specific parameter sweeps. For instance this exploration generates *ffsaroc* configurations with the *starttemp* parameter set to 10 and 15, the *endtemp* parameter set to 2 and 4, *nusvc* configurations with *radial basis* and *polynomial* kernel functions, etc. This exploration generates, for each dataset, 32 ANN based configurations for the *ffrp* engine (it is an engine with no parameters, 32 is the number of combinations of ANNs generated with the

Fig. 2 Sample BiomedTK exploration file

```
explore.neurons.input       = 9
explore.neurons.output      = 2
explore.neurons.layer.01    = 18:36
explore.neurons.layer.02    = 9:18
explore.neurons.layer.03    = 5:9

explore.activation.input    = tanh
explore.activation.output   = tanh:sigm
explore.activation.layer.01 = tanh:sigm
explore.activation.layer.02 = tanh
explore.activation.layer.03 = tanh

explore.nblayers.fixed      = yes

explore.datasets = BREAST-TISSUE.CAR
                   BREAST-TISSUE.NORM

explore.stop.error          = 0.1
explore.stop.epochs         = 2000
```

```
explore.engines     = ffrp:ffga:
                      ffsaroc:nusvc

explore.validation          = testpct 40

explore.numberofjobs        = 50

explore.encog.ffga.matepercent  = 0.5
explore.encog.ffga.percentmate  = 0.2
explore.encog.ffga.population   = 100:200

explore.encog.ffsaroc.starttemp = 10:15
explore.encog.ffsaroc.endtemp   = 2:4
explore.encog.ffsaroc.cycles    = 100

explore.libsvm.nusvc.kernel        = rbf:pol
explore.libsvm.nusvc.nu            = 0.4:0.35
explore.libsvm.nusvc.degree        = 2:3:4
explore.libsvm.nusvc.gamma         = 0.03:0.035
explore.libsvm.nusvc.coef0         = 0.0
explore.libsvm.nusvc.shrink        = yes
explore.libsvm.nusvc.probestimates = yes
```

specified layers, neurons and activation functions per layer), 64 configurations for the *ffga* engine, 128 for *ffsaroc* and 24 for *nusvm* (which does not use ANNs). In total this exploration generates 496 engine configurations (248 for each dataset) split into 50 jobs as specified in the same exploration file. This way, 49 jobs will train 10 engine configurations, and one job that will train 6 configurations.

Exploration jobs are sent to computing resources for training the MLC configurations they contain. Explorations can be made as large or small as desired, depending their feasibility on the capacity of the available computing resources. Training times for different classifier engines vary greatly depending on dataset size, parameters, ANN complexities, etc. When confronted to a new dataset for which one wants to explore classifier configurations, our method proposes the following steps:

1. Start with small explorations launched locally to get a sense on the computing times needed by each different classifier and their initial performance.
2. Perform increasingly larger explorations, first locally and then over a larger computing infrastructure, to understand what classifier parameters might work better with each classifier.
3. Devise very large explorations (either by number of configurations and/or by computing time each configuration takes) and send them to a large computing infrastructure.

For this method to be effective, an agile and fast interaction between the researcher and the computing infrastructure becomes essential, so that, if computing resources are available, multiple explorations can be tested in reasonable time. It is fundamentally this agile interaction that allows researchers to efficiently build exploration strategies and gain understanding on what classifiers suit better a certain problem in hand.

This method is supported by BiomedTK which, for performing explorations, offers the researcher two options: (1) launch the jobs sequentially over his local machine, or (2) submit them to a gLite Grid infrastructure to use in parallel available distributed computing resources. Results (accuracy and ROC Az measures for test and train data) of each trained engine configuration are stored in a database that can be later inspected through standard SQL sentences. A command line tool exposes BiomedTK functionality to researchers. There are commands to import and manipulate datasets, to create a job set from an exploration file, to launch locally and exploration, to launch and monitor explorations to a gLite Grid service, to inspect and test exploration results, to build ensemble classifiers, etc. In practice, when faced with the task of exploring MLC configurations for a given dataset, BiomedTK enables researchers to carry out the method above by cycling through the following steps

1. make initial exploration issuing BiomedTK commands *jobset* + *launch* (local launch)
2. inspect database for results
3. refine and enlarge initial exploration by issuing commands *jobset* + *glite.prepare* + *glite.submit*
4. inspect database for results
5. repeat from step 1 or step 3

Initially confronted with a large availability of Grid resources, but suffering large job latency times hindering our capability to cycle through these steps, we faced the need to have an efficient cycle through which understanding about explorations and classifiers was to be gained for a particular dataset. Otherwise, available Grid resources become very hard to be used with agility. This is what

motivated the inclusion within BiomedTK of a Job Agents mechanism as described previously.

Figure 3 shows a typical BiomedTK session, importing a dataset and launching an exploration locally. This method and its supporting tool are focused on providing the means to exploit Grid resources for exploring MLC configurations, regardless the sources and previous manipulations of data. For this purpose, BiomedTK provides basic functionality for data normalization and tools for importing and exporting data to commonly used formats (CSV, WEKA arff, etc.), so that researchers can use their usual tools for data preprocessing before feeding datasets to BiomedTK.

A method to build ensemble classifiers

Ensemble methods in machine learning [35] combine existing classifiers as an attempt to improve the performance of single classifiers either by compensating with each other in cases where a single classifier outperforms other, or by combining redundancy classification offered by several single classifiers.

For their simplicity and generality we are interested in Error Correcting Output Codes, a technique developed in the context of digital signal processing theory [36, 37] providing redundancy in communications to compensate for noise and applied in machine learning since [38]. In short, for a given multiclass classification problem with n classes, an Error Correcting Output Code (ECOC) is defined by a matrix of size $m \times n$ describing, for each of the n possible final classes, how the m available binary classifiers are combined. This matrix is referred to as the *codematrix*, and Fig. 4 shows an example where 7 binary classifiers are used to produce a 6 class ensemble classifier. Each class is assigned a codeword that represents the participation of each binary classifier in that class. When an input vector is fed into each one of the 7 classifiers, a binary codeword is produced combining the output of all the binary classifiers. Then, a distance measure between codewords is defined and the class with the closest codeword to the one produced by the binary classifiers is selected to be the one assigned to the initial input vector. ECOCs are being used frequently in machine learning [39–41] and also in biomedical contexts [42].

BiomedTK supports the ECOC based method for building ensemble classifiers. This includes (1) the definition of code matrices for a certain multiclass dataset, (2) the generation of binary datasets for each column in the code matrix and (3) assembling previously trained binary

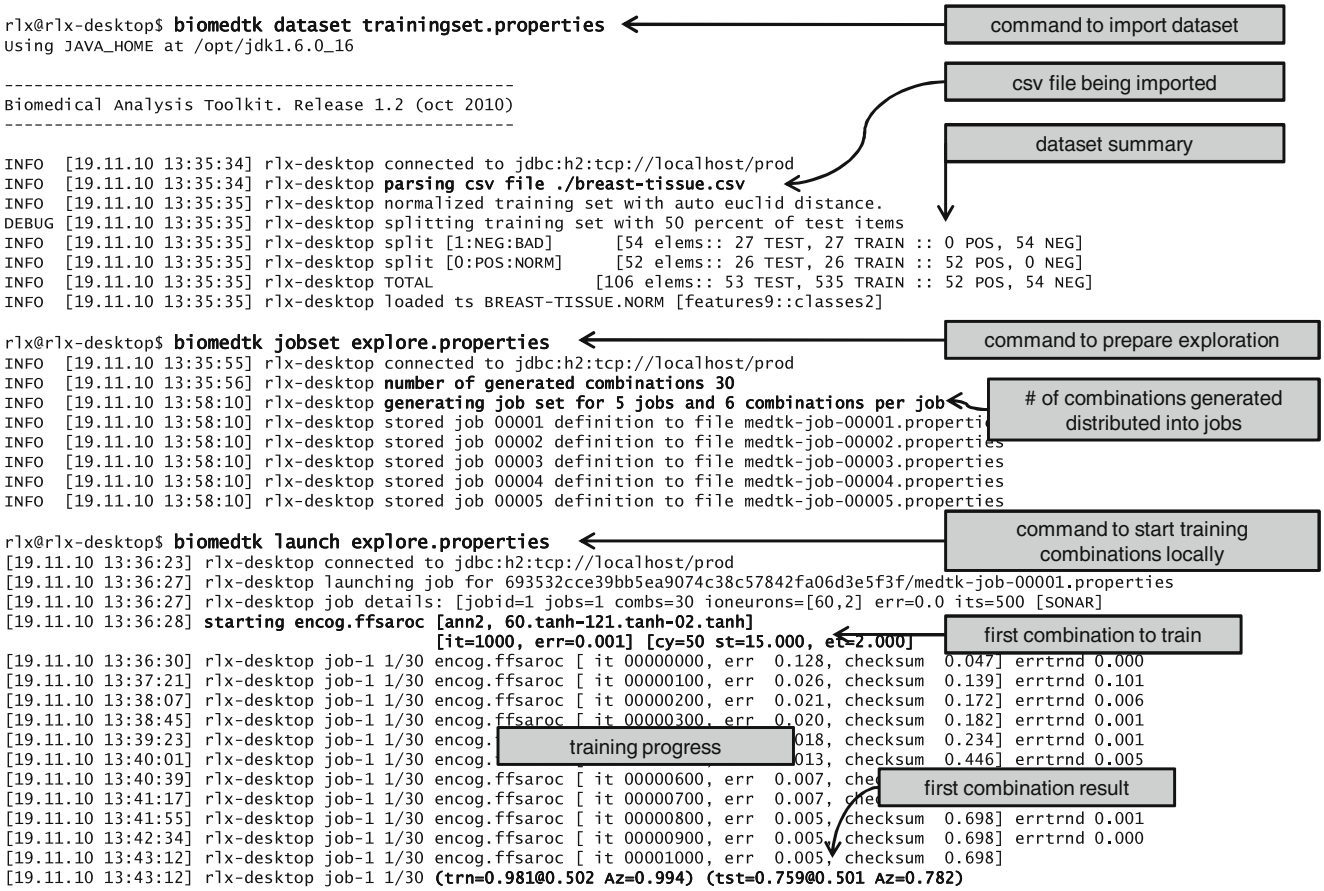

Fig. 3 Sample BiomedTK session

Fig. 4 Sample BiomedTK classifier ensemble configuration file

```
ensemble.trainingset           = BREAST-TISSUE

ensemble.classifiers.validation = cvfolds 10
ensemble.classifiers.select     = bestPct any  bestAz 4035  bestPct bestPct any
ensemble.classifiers.names      =  car    fad    gla   mas    con     adi    norm
ensemble.codematrix.car         =   1      0      0     0      0       0      0
ensemble.codematrix.fad         =   0      1      0     0      0       0      0
ensemble.codematrix.gla         =   0      0      1     0      0       0      1
ensemble.codematrix.mas         =   0      0      0     1      0       0      0
ensemble.codematrix.con         =   0      0      0     0      1       0      1
ensemble.codematrix.adi         =   0      0      0     0      0       1      1
```

classifiers for each column into ensemble classifiers. Figure 4 represents an actual codematrix definition file as accepted by BiomedTK, where BREAST-TISSUE is a six-class dataset (from the UCI repository, *dataset with electrical impedance measurements of freshly excised tissue samples from the breast*) and the codematrix specifies seven binary classifiers (columns). The first six columns would be used to train classifiers distinguishing each class from the rest. The *norm* column will be used to train classifiers distinguishing elements tagged as *car* (for carcinoma), *fad* (fibro-adenoma) or *mas* (mastopathy) from the rest (representing normal tissue tagged as *glandular*, *connective* or *adipose*).

The following summarizes the method supported by BiomedTK for building ensemble MLC:

1. Start with a multiclass dataset with *n* classes and a code matrix specification with *n* rows and *m* columns, such as the one in Fig. 4. Then, create for each column in the code matrix a new binary dataset according to the column specification (for column *c*, each element is labeled as "c" or "non-C" according to whether its ideal class in the original dataset is marked as 1 or 0 in column *c*). This generates *m* binary datasets.
2. Explore MLC configurations for each newly created binary dataset. In fact, Fig. 2 shows an exploration file to search binary classifier configurations for the *car* and *norm* derived binary datasets as created in the previous step starting from the code matrix shown in Fig. 4. This generates, for each column, a set of MLC. BiomedTK supports this step through the exploration method explained in section 2.1.
3. Choose, one MLC for each column and ensemble them through the ECOC method. This is done through the *ensembles* command, which takes MLCs for each column, ensembles them and measures the performance on the ensemble classifier.

Observe that, for each column in a code matrix, one might have generated several binary classifiers in step 2 above; hence, the researcher needs to decide which specific classifier to use. BiomedTK supports this by interpreting the *ensemble.classifiers.select* line in the codematrix in

Table 2 Selected UCI datasets

dataset	description	number of elements	number of input features
bcw	Cell-based metrics for breast cancer	699	9
bcwd	Diagnostic breast cancer Wisconsin database	569	30
btcat	Carcinoma in breast tissue	106	9
echocard	Data for classifying if patients will survive for at least one year after a heart attack	131	8
haber	Survival of patients who had undergone surgery for breast cancer	306	3
heartsl	Patient indicators for presence of heart disease	270	13
hepat	Patient indicators for presence of hepatitis	155	19
liver	BUPA Medical Research Ltd. database on liver desease	345	6
mmass	Benign/malignant mammographic masses based on BI-RADS attributes and the patient's age	961	5
park	Oxford Parkinson's Disease Detection Dataset	195	22
pgene	E. Coli promoter gene sequences (DNA) with partial domain theory	106	57
pimadiab	Patient indicators for presence of diabetes	768	8
spambase	Classifying email as spam or not spam	4601	57
spectf	Data on cardiac Single Proton Emission Computed Tomography (SPECT) images	267	44
tictac	Binary classification task on possible configurations of tic-tac-toe game	958	9

Table 3 Exploration summary per engine and dataset

engine		bcw	bcwd	btcat	echocard	haber	heartsl	hepat	liver	mmass	park	pgene	pimadiab	spam	spectf	tictac
ffbp	configs	32	20	72	96	128	48	16	48	96	48	18	60	16	12	36
	CPU hours	77.82	102.04	95.96	64.42	36.92	123.30	50.80	46.13	180.65	213.00	164.17	192.80	305.38	64.22	149.01
	best pct	86.59%	95.00%	74.40%	68.30%	69.82%	89.52%	88.63%	71.33%	83.73%	93.47%	89.09%	70.50%	78.76%	57.95%	81.10%
	best az	0.860	0.943	0.729	0.680	0.717	0.908	0.884	0.695	0.849	0.942	0.889	0.682	0.779	0.598	0.799
ffbproc	configs	32	20	72	96	128	48	16	48	96	48	18	60	16	12	36
	CPU hours	118.55	171.54	128.33	118.53	57.13	211.50	83.97	65.74	293.50	363.29	333.84	313.40	433.12	104.67	241.50
	best pct	90.17%	95.06%	80.40%	77.33%	73.93%	91.41%	89.11%	80.85%	91.74%	98.28%	93.38%	74.24%	82.90%	67.18%	88.06%
	best az	0.889	0.958	0.819	0.744	0.718	0.932	0.873	0.817	0.913	0.991	0.918	0.715	0.823	0.646	0.875
ffrp	configs	4	4	4	4	4	4	4	4	4	4	4	2	2	4	4
	CPU hours	8.81	17.78	4.86	2.81	1.13	9.47	9.85	3.48	5.95	15.09	36.49	4.94	30.64	19.13	18.45
	best pct	81.41%	95.09%	80.07%	63.80%	62.72%	93.08%	92.33%	75.71%	88.43%	98.29%	94.36%	79.90%	83.63%	90.80%	88.64%
	best az	0.804	0.966	0.786	0.614	0.634	0.907	0.939	0.772	0.890	0.970	0.921	0.813	0.809	0.885	0.899
ffrproc	configs	4	4	4	4	4	4	4	4	4	4	4	2	2	4	4
	CPU hours	13.70	31.10	7.04	4.57	1.84	12.78	15.25	4.64	11.60	23.57	65.80	7.93	57.74	34.45	24.60
	best pct	88.10%	96.47%	82.29%	81.06%	92.34%	93.12%	93.59%	80.02%	94.96%	98.19%	96.14%	81.73%	85.86%	93.74%	94.09%
	best az	0.892	0.938	0.804	0.822	0.902	0.931	0.943	0.813	0.938	0.957	0.932	0.818	0.856	0.933	0.926
ffsa	configs	32	18	60	48	54	36	16	36	48	36	16	48	16	24	32
	CPU hours	116.76	150.39	124.15	51.69	25.50	131.22	68.97	45.26	141.41	247.85	296.47	249.00	430.88	243.26	246.76
	best pct	97.14%	92.70%	90.56%	75.93%	82.17%	90.03%	94.69%	80.78%	95.23%	97.32%	88.50%	83.12%	80.42%	90.30%	84.91%
	best az	0.933	0.901	0.822	0.770	0.822	0.901	0.937	0.818	0.939	0.967	0.890	0.809	0.793	0.910	0.867
ffsaroc	configs	32	18	60	48	54	36	32	36	48	36	16	48	16	24	32
	CPU hours	131.53	147.65	148.70	71.76	35.25	166.53	197.77	55.38	195.51	328.29	289.83	328.65	599.21	259.80	259.39
	best pct	92.75%	95.04%	90.56%	85.90%	84.65%	96.30%	95.33%	81.62%	90.83%	98.03%	87.74%	82.41%	90.81%	90.83%	94.85%
	best az	0.923	0.969	0.901	0.863	0.852	0.958	0.955	0.822	0.925	0.996	0.883	0.815	0.902	0.922	0.925
ffga	configs	24	24	36	32	29	36	16	24	36	24	12	36	6	12	24
	CPU hours	104.84	270.42	90.20	52.04	15.94	197.31	103.45	46.91	148.84	233.33	270.20	261.35	249.07	179.40	226.25
	best pct	82.26%	92.41%	77.94%	80.55%	82.01%	96.30%	87.63%	67.05%	88.50%	94.40%	73.37%	81.23%	68.69%	85.67%	72.57%
	best az	0.865	0.936	0.770	0.808	0.807	0.994	0.877	0.687	0.870	0.960	0.751	0.801	0.698	0.845	0.74
ffgaroc	configs	24	24	36	32	29	36	16	24	36	24	12	36	6	12	24
	CPU hours	121.58	323.06	114.30	55.01	21.01	218.34	113.09	48.85	187.89	252.11	320.13	278.42	281.44	207.38	271.21
	best pct	86.77%	96.89%	80.21%	82.57%	81.74%	92.91%	87.19%	76.04%	92.98%	96.52%	80.27%	83.35%	78.68%	86.86%	80.49%
	best az	0.854	0.955	0.786	0.835	0.826	0.944	0.875	0.770	0.900	0.977	0.780	0.808	0.766	0.854	0.822
c-svc	configs	48	48	256	256	256	160	128	256	48	256	256	192	48	256	256
	CPU hours	0.92	0.86	0.67	0.83	1.98	1.23	0.53	2.11	1.17	1.54	0.68	3.59	5.01	1.70	7.37
	best pct	97.14%	97.65%	90.33%	86.60%	93.75%	94.71%	95.25%	80.26%	92.34%	97.67%	93.16%	82.25%	91.93%	93.38%	95.71%

Table 3 (continued)

engine		bcw	bcwd	btcat	echocard	haber	heartsl	hepat	liver	mmass	park	pgene	pimadiab	spam	spectf	tictac
	best az	0.930	0.977	0.815	0.864	0.940	0.931	0.951	0.804	0.939	0.953	0.952	0.810	0.919	0.935	0.931
nu-svc	configs	48	48	256	256	256	160	128	256	48	256	256	192	48	256	256
	CPU hours	0.86	0.83	0.64	2.00	2.18	1.02	0.49	2.57	1.11	1.18	0.73	4.35	4.49	1.95	6.02
	best pct	96.91%	96.64%	89.40%	84.70%	93.75%	95.47%	96.44%	81.01%	95.83%	96.94%	95.62%	81.14%	88.55%	93.20%	96.24%
	best az	0.924	0.962	0.812	0.857	0.931	0.944	0.957	0.814	0.965	0.962	0.946	0.809	0.894	0.947	0.947
total	configs	280	228	856	872	942	568	376	736	464	736	612	676	176	616	704
total	CPU hours	695.38	1215.67	714.87	423.65	198.89	1072.72	644.16	321.06	1167.61	1679.26	1778.35	1644.43	2396.98	1115.97	1450.55
best az	overall	0.933	0.977	0.901	0.864	0.940	0.994	0.957	0.822	0.965	0.996	0.952	0.818	0.919	0.947	0.947
best pct	overall	97.14%	97.65%	90.56%	86.60%	93.75%	96.30%	96.44%	81.62%	95.83%	98.29%	96.14%	83.35%	91.93%	93.74%	96.24%
best pct	in literature	98.94%	82.33%	86%	90.9%	80.48%	89.5%	93.51%	72.5%	0.890	95.8%	91.5%	81.83%	93.88%	82.8%	90.9%

which, for each column, the researcher specifies what criteria to follow to select binary classifiers. With *bestPct*, the binary classifier for that column with best classification rate for test data is used. With *bestAz*, the one with best area under the ROC curve measure for test data is chosen. A number (such as in the *mas* column) refers to the ID of a specific binary classifier present in the database that the researcher wants to use. With *any*, we instruct BiomedTK to try all binary classifiers available for that column.

Experimental setup

A set of experiments was setup in order to show the contribution of the proposed method to researchers' efficiency and experimental reach. Our aim was to demonstrate how (1) complex experiments consisting of several datasets, engines and classifier configuration can be managed with relative simplicity and (2) the Grid resources required to execute the proposed experiments (about 16,000 CPU hours) could be exploited in an efficient manner. Therefore, we set forth to use BiomedTK to find classifiers to selected biomedical datasets from the UCI machine learning repository [22] by exploiting Grid computing resources, and aiming at reaching, in reasonable time, accuracy levels comparable to the ones found in different literature sources for the given datasets. By being able to obtain efficiently these results, we are then positioned to pursue research in a timely manner using the methods herewith described. We selected the UCI datasets shown in Table 2 and defined increasingly larger explorations for each dataset including all engines supported by BiomedTK (see Table 1). The Area under the ROC Curve (ROC Az) measures the capability of a binary classifier to correctly distinguish positive from negative instances, even if their scores are away from the ideal ones. BiomedTK allows evaluating classifier ROC Az and we use it to test our own modifications to existing classifiers by allowing them to optimize ROC Az and not only accuracy. This is the reason behind choosing all datasets binary (with only two classes). Explorations for each dataset included many classifier configurations (see Table 3), each configuration using different classifier parameters, ANN layers, neurons per layer, etc. Each exploration was tuned for each dataset to account for their different characteristics. For instance, the input layer of any ANN must have as many neurons as dataset features, datasets harder to classify might require exploring more configurations, larger datasets require more training time so explorations cannot be as large, etc.

The selected UCI datasets were imported from a CSV formatted file and basic data normalization was performed by BiomedTK before the explorations. This consisted in mapping each input feature to the [0,1] interval in the

following way: for each element of a dataset the value v_i of feature $_i$ was normalized to $v_i' = \frac{v_i}{Max_i - Min_i}$ where Min_i and Max_i are the minimum and maximum values of feature $_i$ in all elements of the dataset.

The gLite infrastructure at CETA-CIEMAT with 200 CPU cores was used for this experiment. For each dataset, a few explorations were made on a local machine before sending larger explorations to gLite. This was done to acquire a sense on how much computing time each classifier configuration would take, what results could be expected and how large explorations would need to be. For instance, a dataset whose instances are easily separable (such as the BCW dataset) would start giving good classification results even with simple classifier configurations and one would expect minor improvements from a larger exploration.

Our results were compared mostly with those found in [43–57]. We did not aim to make an exhaustive literature review, but to sample referenced works and reach comparable results to demonstrate the utility of the MLC exploration method and its underlying tool (BiomedTK). Experimental conditions change between authors in validation methods used, how results are summarized, etc. so comparisons must be carefully interpreted. To be as coherent as possible with the different setups, we used 10-fold cross validation. Furthermore, in some works it is not clear what specific classifier configuration or validation method was used, which somehow also constitutes a challenge in our exploration endeavor.

Results

A total of about 16,000 CPU hours were consumed to train 8,842 different classifier configurations over 15 datasets. Table 3 shows, for each dataset and engine: (1) how many configurations were trained, (2) how many CPU hours took to train them, (3) the best percentage of elements correctly classified on the test part of the dataset (*accuracy*), and (4) the best ROC Az obtained on the test part of the dataset. Finally, the bottom lines in Table 3 show the best results obtained overall in our exploration (accuracy and ROC Az) and those found in our literature review (accuracy in all datasets, except the *mmass* dataset, where reference [54] gave their results in ROC Az). Figure 5 shows the two ROC curves generated by BiomedTK for one classifier configuration over the BCW dataset. The curve on the left corresponds to the Mann–Whitney statistic and the curve on the right is its smoothed version using the bi-normal distribution method from JLABROC.

Note that for some datasets we made a larger exploration than for others, as we had to make more exploration refinement cycles until we found satisfactory results. A key factor was acquiring a notion on the requirements of computing time for each classifier and dataset so that explorations can be adjusted to the available computing resources. Observe in this sense how datasets with larger number of elements or input features take longer time to train with ANN engines and, were not for the possibility to harness Grid computing resources, exploring even a few classifier configurations for them would simply be impos-

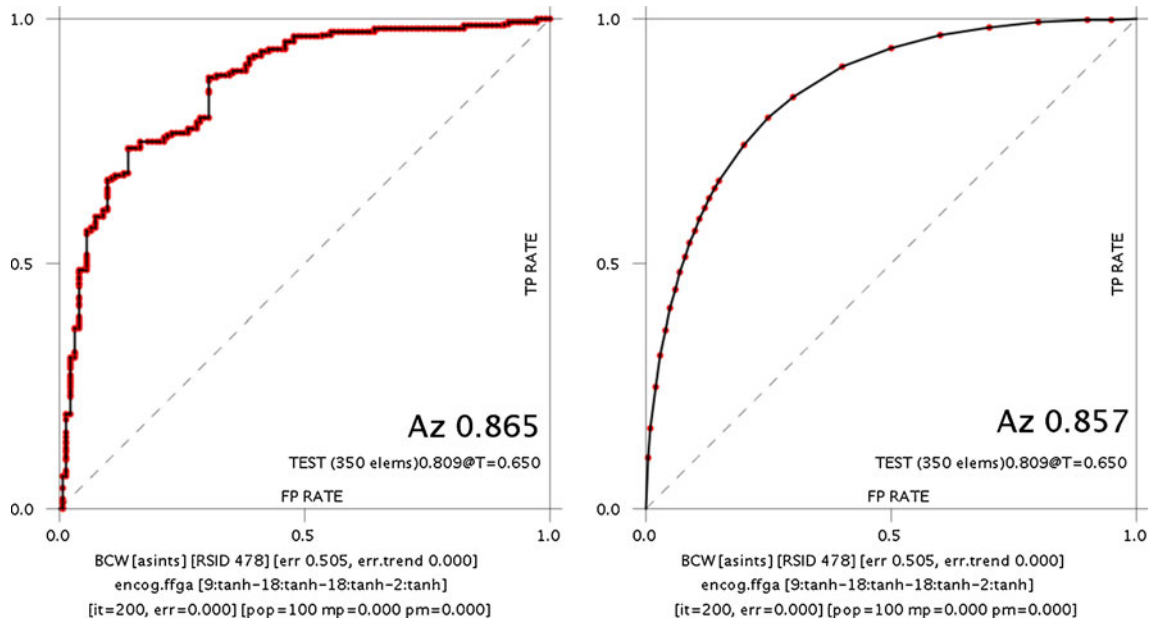

Fig. 5 Binormal smoothed and Mann–Whitney ROC Curves as produced by BiomedTK for the ANN-genetic algorithm classifier configuration with best ROC Az over the BCW dataset

sible. Without an appropriate supporting tool, following the method herewith proposed requiring such large explorations, would imply, on one side, providing the logistics to manage different output formats for the different underlying engines, organize binaries for gLite infrastructure, etc. Then, it would require preparing gLite jobs for each dataset and engine type, monitor their execution, keeping accounting of failed jobs and resubmitting them, gather different outputs for each job and consolidate them in a single file or database, etc. It is the reduction of the effort cost of taking care of all these logistics that makes it possible to efficiently harness Grid resources for machine learning in a systematic way.

In practical terms, for each dataset a single exploration configuration file was defined, containing the specification for all configurations for each engine used (very similar to the one showed in Fig. 2). Each exploration file was first used to make a few explorations in our desktop machines, then parameters sets were fine tuned and enlarged and, finally, sent to the Grid. From a researcher point of view, in addition to the large amount of CPU hours managed with reasonable effort, the proposed method is easily reproducible rendering classifier exploration requiring large computing resources a systematic task. This allows researchers to focus on their core research, devising exploration strategies, evaluating dataset preprocessing algorithms, new classification methods, etc. instead of sorting out the logistics of access methods to Grid infrastructures, preparing datasets differently for each third party engine, etc. In fact, this method has enabled the authors to validate the modifications made to the Encog engines for ROC optimization with satisfactory results [23].

Discussion

This paper described a method to efficiently harness computing power serviced by different Grid infrastructures for building biomedical machine learning classifiers, along with its supporting tool, BiomedTK. Together, they demonstrated be robust and efficient for exploring search spaces of possible configurations of machine learning classifiers provided by arbitrary third party toolkits, using Grid computing resources in an agile manner. With this method, exploring large sets of configurations of machine learning classifiers with different data sources and third party classifier toolkits becomes a task that can be accomplished in reasonable and predictable time, opening the way to systematic experimentation on many of the issues around machine learning for biomedical data analysis (classifier and algorithm design, data analysis, preprocessing, etc.) In this sense, it has served the authors to validate the introduction of ROC based optimization in existing algorithms.

At present our work is centered in two aspects. First, we want to use the BiomedTK framework to find good performing classifiers in a real hospital environment where a digital repository for breast cancer is currently being gathered at Hospital São João - Faculty of Medicine of Porto University, in Portugal, in the context of the current collaboration between INEGI and CETA-CIEMAT through which CAD methods are being built to assist on breast cancer diagnosis [3, 21]. Second, tuning classifier parameters is mostly a heuristic task, not existing rules providing knowledge about what parameters to choose when training a classifier. Through BiomedTK we are gathering data about performance of many classifiers, trained each one with different parameters, ANNs, SVM, etc. This by itself constitutes a dataset that can be data mined to understand what set of parameters yield better classifiers for given situations or even generally. Therefore, we intend to use BiomedTK on this bulk of classifier data to gain insight on classifier parameter tuning.

Acknowledgements 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

References

1. Ramos-Pollan, R., *et al.*, "Exploiting eInfrastructures for medical image storage and analysis: A Grid application for mammography CAD," in *The Seventh IASTED International Conference on Biomedical Engineering*. Austria: Innsbruck, 2010.
2. Drakos, J., et al., A perspective for biomedical data integration: Design of databases for flow cytometry. *BMC Bioinform.* 9:99, 2008.
3. Ramos-Pollan, R., *et al.*, "Grid computing for breast cancer CAD. A pilot experience in a medical environment," in *4th Iberian Grid Infrastructure Conference*. Portugal: Minho, pp. 307–318, 2010.
4. Blanquer Espert, I., et al., Content-based organisation of virtual repositories of DICOM objects. *Future Generation Comput. Syst.* 25:627–37, 2009.
5. KaraçalI, B., Quasi-supervised learning for biomedical data analysis. *Pattern Recognit.* 43:3674–82, 2010.
6. Peng, Y., et al., A novel feature selection approach for biomedical data classification. *J. Biomed. Inform.* 43:15–23, 2010.
7. López, Y., *et al*., "Breast Cancer Diagnosis Based on a Suitable Combination of Deformable Models and Artificial Neural Networks Techniques." in Progress is Pattern Recognition, Image Analysis and Applications. *Lect. Notes Comput. Sci.* 4756/ 2007:803–811, 2007.
8. López, Y., *et al.*, "Computer aided diagnosis system to detect breast cancer pathological lesions," in *Progress in Pattern Recognition, Image Analysis and Applications*. Volume 5197/ 2008, ed. Berlin, Heidelberg: Springer, pp. 453–460, 2008.
9. *The Globus Alliance and Middleware*. Available: http://www.globus.org/
10. *The gLite middleware*. Available: http://glite.web.cern.ch
11. *The European Grid Initiative (EGI)*. Available: http://www.egi.eu

12. *EGI Availability/Reliability results for October 2010*. Available: https://documents.egi.eu/public/ShowDocument?docid=238
13. Halling-Brown, M., et al., A computational Grid framework for immunological applications. *Philos. Transact. Series A Math. Phys. Eng. Sci.* 367:2705–16, 2009.
14. Kacsuk, P., "Extending the services and sites of production grids by the support of advanced portals" in *Proceedings of High Performance Computing for Computational Science - VECPAR 2006*. Rio de Janeiro, Brazil: pp. 644–655, 2007.
15. Schwiegelshohn, U., *et al.*, "Perspectives on grid computing," in *Dagstuhl Seminar Proceedings*. Leibniz: 2009.
16. *Grid Computing: A Vertical Market Perspective 2006–2011*. Available: http://www.insight-corp.com/reports/grid06.asp
17. *The DIRAC project*. Available: http://lhcbweb.pic.es/DIRAC/
18. Bagnasco, S., et al., AliEn: ALICE environment on the GRID. *J. Phys. Conf. Ser.* 119:062012, 2008.
19. Ramos-Pollan, R., *et al.*, "Grid-based architecture to host multiple repositories: A mammography image analysis use case," in *3rd Iberian Grid Infrastructure Conference Proceedings*. Valencia, Spain: pp. 327–338, 2009.
20. Ramos-Pollan, R., *et al.*, "Building medical image repositories and CAD systems on grid infrastructures: A Mammograms Case," in *15th edition of the Portuguese Conference on Pattern Recognition.* Aveiro, Portugal: University of Aveiro, 2009.
21. Ramos-Pollan, R., and Guevara, M., "Grid infrastructures for developing mammography CAD systems" in *32nd Annual International Conference of the IEEE Engineering in Medicine and Biology Society.* Argentina: Buenos Aires, 2010.
22. Frank, A., and Asuncion, A., *UCI Machine Learning Repository* http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science, 2010.
23. Ramos Pollan,R., *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.
24. Heaton, J., "Programming neural networks with encog 2 in Java," ed.: Heaton Research, Inc, 2010.
25. Chang, C.-C., and Lin, C.-J., *LIBSVM: a library for support vector machines*. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm, 2001.
26. Yoon, H. J., et al., Evaluating computer-aided detection algorithms. *Med. Phys.* 34:2024–38, 2007.
27. Fawcett, T., An introduction to ROC analysis. *Pattern Recognit. Lett.* 27:861–74, 2006.
28. John Eng, M. D., *ROC analysis: web-based calculator for ROC curves*. Available: http://www.jrocfit.org, 2006.
29. Mark Hall, *et al.*, "The WEKA data mining software: An update," *SIGKDD Explorations,* vol. 11: 2009.
30. Cortes, C., and Mohri, M., AUC optimization vs. error rate minimization. *Adv. Neural Inf. Process. Syst.* 16:313–20, 2004.
31. Kim, J.-H., Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Comput. Stat. Data Anal.* 53:3735–45, 2009.
32. Efron, B., and Gong, G., A leisurely look at the bootstrap, the jackknife, and cross-validation. *Am. Stat.* 37:36–48, 1983.
33. Efron, B., Estimating the error rate of a prediction rule: Improvement on cross-validation. *J. Am. Stat. Assoc.* 78:316–31, 1983.
34. *The H2 Database Engine*. Available: http://www.h2database.com
35. Dietterich, T. G., "Ensemble methods in machine learning," presented at the Proceedings of the First International Workshop on Multiple Classifier Systems. 2000.
36. Bose, R., and Ray-Chaudhuri, "On a class of error-correcting binary group codes," *Information Control.* vol. 3: pp. 68–79, 1960.
37. Hocquenghen, A., Codes correcteurs d'erreurs. *Chiffres* 2:147–56, 1959.
38. Dietterich, T., and Bakiri, G., "Error-correcting output codes: A general method for improving multiclass inductive learning programs," in *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*. Anaheim, CA: pp. 572–577, 1991.
39. Passerini, A., et al., New results on error correcting output codes of kernel machines. *IEEE Trans. Neural Net.* 15:45–54, 2004.
40. Escalera, S., et al., Subclass problem-dependent design for error-correcting output codes. *IEEE Trans. Patt. Anal. Mach. Intell.* 30:1041–54, 2008.
41. Huiqun, D., *et al.*, "Error-correcting output coding for the convolutional neural network for optical character recognition," in *Document Analysis and Recognition, 2009. ICDAR'09 10th International Conference on 2009*. pp. 581–585, 2009.
42. Escalera, S., *et al.*, "Coronary damage classification of patients with the Chagas disease with error-correcting output codes," in *Intelligent Systems, 2008. IS'08. 4th International IEEE Conference*. pp. 12-17-12-22, 2008.
43. Urbanowicz, R. J., and Moore, J. H., Learning classifier systems: a complete introduction, review, and roadmap. *J. Artif. Evol. App.* 2009:1–25, 2009.
44. Kotsiantis, S., et al., Machine learning: a review of classification and combining techniques. *Artif. Intell. Rev.* 26:159–90, 2006.
45. Lorena, A. C., et al., A review on the combination of binary classifiers in multiclass problems. *Artif. Intell. Rev.* 30:19–37, 2008.
46. Soares, C., "Is the UCI repository useful for data mining?" in *Progress in Artificial Intelligence*. vol. 2902, ed. Berlin, Heidelberg: Springer, pp. 209–223, 2003.
47. Estrela da Silva, J., et al., Classification of breast tissue by electrical impedance spectroscopy. *Med. Biol. Eng. Comput.* 38:26–30, 2000.
48. Sebban, M., et al., Stopping criterion for boosting based data reduction techniques: From binary to multiclass problem. *J. Mach. Learn. Res.* 3:863–85, 2003.
49. Wilson, D. R., and Martinez, T. R., "Improved center point selection for probabilistic neural networks," in *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, (ICANNGA'97)*. pp. 514–517, 1997.
50. Jiang, Y., and Zhou, Z.-H., Editing training data for knn classifiers with neural network ensemble. *Lect. Notes Comput. Sci.* 3173:356–61, 2004.
51. Fung,G., *et al.*, "A fast iterative algorithm for fisher discriminant using heterogeneous kernels," in *Proceedings of the twenty-first international conference on Machine learning*. Alberta, Canada: Banff, p. 40, 2004.
52. Vlachos,M., *et al.*, "Non-linear dimensionality reduction techniques for classification and visualization," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. Edmonton, Alberta, Canada, pp. 645–651, 2002.
53. Esmeir, S., and Markovitch, S.,"Lookahead-based algorithms for anytime induction of decision trees," in *Proceedings of the twenty-first international conference on Machine learning*. Alberta, Canada: Banff, p. 33, 2004.
54. Elter, M., et al., The prediction of breast cancer biopsy outcomes using two CAD approaches that both emphasize an intelligible decision process. *Med. Phys.* 34(11):4164–72, 2007.
55. Little, M. A., et al., Suitability of dysphonia measurements for telemonitoring of Parkinson's disease. *IEEE Trans. Biomed. Eng.* 56:1015–22, 2009.
56. Li, J., and Wong, L., "Using rules to analyse bio-medical data: A comparison between C4.5 and PCL," in *Advances in Web-Age Information Management*. vol. 2762, ed. Berlin, Heidelberg: Springer, pp. 254–265, 2003.
57. Domeniconi, C., and Yan, B., "Nearest neighbor ensemble," in *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1 - Volume 01*. pp. 228–231, 2004.