Journal of Machine Learning Research 14 (2013) 3201-3205 Submitted 1/12; Revised 2/13; Published 10/13

GURLS: A Least Squares Library for Supervised Learning

Andrea Tacchetti[*] ATACCHET@MIT.EDU
Pavan K. Mallapragada PAVAN_M@MIT.EDU
Center for Biological and Computational Learning, McGovern Institute for Brain Research
Massachusetts Institute of Technology, Bldg. 46-5155
Cambridge, MA, 02139, USA

Matteo Santoro MATTEO.SANTORO@IIT.IT
Laboratory for Computational and Statistical Learning, Istituto Italiano di Tecnologia
Via Morego, 30
16163 Genova, Italy

Lorenzo Rosasco[*] LROSASCO@MIT.EDU
DIBRIS, Università degli Studi di Genova
via Dodecaneso, 35
16146 Genova, Italy

Editor: Geoff Holmes

Abstract

We present GURLS, a least squares, modular, easy-to-extend software library for efficient supervised learning. GURLS is targeted to machine learning practitioners, as well as non-specialists. It offers a number state-of-the-art training strategies for medium and large-scale learning, and routines for efficient model selection. The library is particularly well suited for multi-output problems (multi-category/multi-label). GURLS is currently available in two independent implementations: Matlab and C++. It takes advantage of the favorable properties of regularized least squares algorithm to exploit advanced tools in linear algebra. Routines to handle computations with very large matrices by means of memory-mapped storage and distributed task execution are available. The package is distributed under the BSD license and is available for download at `https://github.com/LCSL/GURLS`.

Keywords: regularized least squares, big data, linear algebra

1. Introduction and Design

Supervised learning has become a fundamental tool for the design of intelligent systems and the analysis of high dimensional data. Key to this success has been the availability of efficient, easy-to-use software packages. New data collection technologies make it easy to gather high dimensional, multi-output data sets of increasing size. This trend calls for new software solutions for the automatic training, tuning and testing of supervised learning methods. These observations motivated the design of GURLS (Grand Unified Regularized Least Squares). The package was developed to pursue the following goals: *Speed*: Fast training/testing procedures for learning problems with potentially large/huge number of points, features and especially outputs (e.g., classes). *Memory*: Flexible data management to work with large data sets by means of memory-mapped storage. *Performance*:

∗. Also in the Laboratory for Computational and Statistical Learning, Istituto Italiano di Tecnologia and Massachusetts Institute of Technology

State of the art results in high-dimensional multi-output problems. *Usability and modularity*: Easy to use and to expand. GURLS is based on Regularized Least Squares (RLS) and takes advantage of all the favorable properties of these methods (Rifkin et al., 2003). Since the algorithm reduces to solving a linear system, GURLS is set up to exploit the powerful tools, and recent advances, of linear algebra (including randomized solver, first order methods, etc.). Second, it makes use of RLS properties which are particularly suited for high dimensional learning. For example: (1) RLS has natural primal and dual formulation (hence having complexity which is the smallest between number of examples and features); (2) efficient parameter selection (closed form expression of the leave one out error and efficient computations of regularization path); (3) natural and efficient extension to multiple outputs. Specific attention has been devoted to handle large high dimensional data sets. We rely on data structures that can be serialized using memory-mapped files, and on a distributed task manager to perform a number of key steps (such as matrix multiplication) without loading the whole data set in memory. Efforts were devoted to to provide a lean API and an exhaustive documentation. GURLS has been deployed and tested successfully on Linux, MacOS and Windows. The library is distributed under the simplified BSD license, and can be downloaded from `https://github.com/LCSL/GURLS`.

2. Description of the Library

The library comprises four main modules. GURLS and bGURLS—both implemented in Matlab—are aimed at solving learning problems with small/medium and large-scale data sets respectively. GURLS^{++}and bGURLS^{++}are their C++ counterparts. The Matlab and C++ versions share the same design, but the C^{++}modules have significant improvements, which make them faster and more flexible. The specification of the desired machine learning experiment in the library is straightforward. Basically, it is a formal description of a *pipeline*, that is, an ordered sequence of steps. Each step identifies an actual learning task, and belongs to a predefined category. The core of the library is a method (a class in the C^{++}implementation) called *GURLScore*, which is responsible for processing the sequence of tasks in the proper order and for linking the output of the former task to the input of the subsequent one. A key role is played by the additional "options" structure, referred to as OPT. OPT is used to store all configuration parameters required to customize the behavior of individual tasks in the pipeline. Tasks receive configuration parameters from OPT in read-only mode and—upon termination—the results are appended to the structure by *GURLScore* in order to make them available to subsequent tasks. This allows the user to skip the execution of some tasks in a pipeline, by simply inserting the desired results directly into the options structure. Currently, we identify six different task categories: data set splitting, kernel computation, model selection, training, evaluation and testing and performance assessment and analysis. Tasks belonging to the same category may be interchanged with each other.

2.1 Learning From Large Data Sets

Two modules in GURLS have been specifically designed to deal with *big data* scenarios. The approach we adopted is mainly based on a memory-mapped abstraction of matrix and vector data structures, and on a distributed computation of a number of standard problems in linear algebra. For learning on big data, we decided to focus specifically on those situations where one seeks a linear model on a large set of (possibly non linear) features. A more accurate specification of what "large" means in GURLS is related to the number of features d and the number of training

data set	# of samples	# of classes	# of variables
optdigit	3800	10	64
landast	4400	6	36
pendigit	7400	10	16
letter	10000	26	16
isolet	6200	26	600

Table 1: Data sets description.

examples n: we require it must be possible to store a $\min(d,n) \times \min(d,n)$ matrix in memory. In practice, this roughly means we can train models with up-to $25k$ features on machines with $8Gb$ of RAM, and up-to $50k$ features on machines with $36Gb$ of RAM. We *do not* require the data matrix itself to be stored in memory: within GURLS it is possible to manage an arbitrarily large set of training examples. We distinguish two different scenarios. Data sets that can fully reside in RAM without any memory mapping techniques—such as swapping—are considered to be small/medium. Larger data sets are considered to be "big" and learning must be performed using either bGURLS or bGURLS^{++}. These two modules include all the design patterns described above, and have been complemented with additional big data and distributed computation capabilities. Big data support is obtained using a data structure called *bigarray*, which allows to handle data matrices as large as the space available on the hard drive: we store the entire data set on disk and load only small chunks in memory when required. There are some differences between the Matlab and C^{++}implementations. bGURLS relies on a simple, ad hoc interface, called *GURLS Distributed Manager* (GDM), to distribute matrix-matrix multiplications, thus allowing users to perform the important task of kernel matrix computation on a distributed network of computing nodes. After this step, the subsequent tasks behave as in GURLS. bGURLS^{++}(currently in active development) offers more interesting features because it is based on the MPI libraries. Therefore, it allows for a full distribution within every single task of the pipeline. All the processes read the input data from a shared filesystem over the network and then start executing the same pipeline. During execution, each process' task communicates with the corresponding ones. Every process maintains its local copy of the options. Once the same task is completed by all processes, the local copies of the options are synchronized. This architecture allows for the creation of hybrid pipelines comprising serial one-process-based tasks from GURLS^{++}.

3. Experiments

We decided to focus the experimental analysis in the paper to the assessment of GURLS' performance both in terms of accuracy and time. In our experiments we considered 5 popular data sets, briefly described in Table 1. Experiments were run on a Intel Xeon 5140 @ 2.33GHz processor with 8GB of RAM, and running Ubuntu 8.10 Server (64 bit).

	optdigit		**landsat**		**pendigit**	
	accuracy (%)	time (s)	accuracy (%)	time (s)	accuracy (%)	time (s)
GURLS (linear primal)	92.3	0.49	63.68	0.22	82.24	0.23
GURLS (linear dual)	92.3	726	66.3	1148	82.46	5590
LS-SVM linear	92.3	7190	64.6	6526	82.3	46240
GURLS (500 random features)	96.8	25.6	63.5	28.0	96.7	31.6
GURLS (1000 random features)	97.5	207	63.5	187	95.8	199
GURLS (Gaussian kernel)	98.3	13500	90.4	20796	98.4	100600
LS-SVM (Gaussian kernel)	98.3	26100	90.51	18430	98.36	120170

Table 2: Comparison between GURLS and LS-SVM.

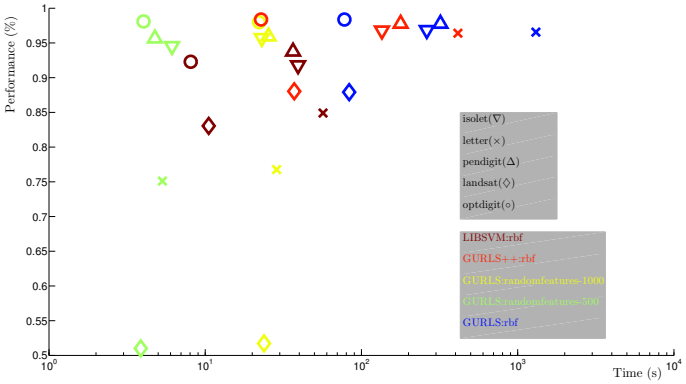

Figure 1: Prediction accuracy vs. computing time. The color represents the training method and the library used. In blue: the Matlab implementation of RLS with RBF kernel, in red: its C++ counterpart. In dark red: results of LIBSVM with RBF kernel. In yellow and green: results obtained using a linear kernel on 500 and 1000 random features respectively.

We set up different pipelines and compared the performance to SVM, for which we used the python modular interface to LIBSVM (Chang and Lin, 2011). Automatic selection of the optimal regularization parameter is implemented identically in all experiments: (i) split the data; (ii) define a set of regularization parameter on a regular grid; (iii) perform hold-out validation. The variance of the Gaussian kernel has been fixed by looking at the statistics of the pairwise distances among training examples. The prediction accuracy of GURLS and GURLS^{++}is identical—as expected—but the implementation in C++ is significantly faster. The prediction accuracy of standard RLS-based methods is in many cases higher than SVM. Exploiting the primal formulation of RLS, we further ran experiments with the random features approximation (Rahimi and Recht, 2008). As show in Figure 1, the performance of this method is comparable to that of SVM at a much lower computational cost in the majority of the tested data sets. We further compared GURLS with another available least squares based toolbox: the LS-SVM toolbox (Suykens et al., 2001), which includes routines for parameter selection such as *coupled simulated annealing* and line/grid search. The goal of this experiment is to benchmark the performance of the parameter selection with random data splitting included in GURLS. For a fair comparison, we considered only the Matlab implementation of GURLS. Results are reported in Table 2. As expected, using the linear kernel with the primal formulation—not available in LS-SVM—is the fastest approach since it leverages the lower dimensionality of the input space. When the Gaussian kernel is used, GURLS and LS-SVM have comparable computing time and classification performance. Note, however, that in GURLS the number of parameter in the grid search is fixed to 400, while in LS-SVM it may vary and is limited to 70. The interesting results obtained with the random features implementation in GURLS, make it an interesting choice in many applications. Finally, all GURLS pipelines, in their Matlab implementation, are faster than LS-SVM and further improvements can be achieved with GURLS^{++}.

Acknowledgments

We thank Tomaso Poggio, Zak Stone, Nicolas Pinto, Hristo S. Paskov and CBCL for comments and insights.

References

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems*, volume 21, pages 1313–1320, 2008.

R. Rifkin, G. Yeo, and T. Poggio. Regularized least-squares classification. *Nato Science Series Sub Series III Computer and Systems Sciences*, 190:131–154, 2003.

J. Suykens, T. V. Gestel, J. D. Brabanter, B. D. Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, 2001. ISBN 981-238-151-1.