BGURLS User's Guide

July 1, 2013

1 Introduction

The module BGURLS has been specifically designed to deal with the so-called *big learning* scenario. We consider to be big those data that cannot fully reside in RAM without any memory mapping techniques – such as swapping. Conversely data that can fully reside in RAM are considered to be small/medium. Learning with big data can be carried out with the dedicated module BGURLS, under the conditions described in the following. Without the ambition to develop a good solution for all the infinte possible variants of big learning, 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 directly related to the number d of features: we require it must be possible to store a $d \times d$ 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. It is important to remark we *do not* require the data matrix itself to be stored in memory. Indeed, in BGURLS it is possible to manage an arbitrarily large set of training examples.

2 BGURLS design

BGURLS include all the design patterns described for GURLS, and have been complemented with additional big data capabilities. Big data support is obtained using a data structure called *bigarray*, which allows to handle data matrices as large as a machine's available space on hard drive instead of its RAM: we store the entire dataset on disk and load only small chunks in memory when required. BGURLS relies on a simple interface – developed ad-hoc and 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 (cfr. Fig. 1).

The BGURLS Core is identified with the `bgurls` command, which behaves as `gurls`. As `gurls` it accepts exactly four arguments:

1. the bigarray of the input data.
2. the bigarray of the labels vector.
3. An options' structure.
4. A job-id number.

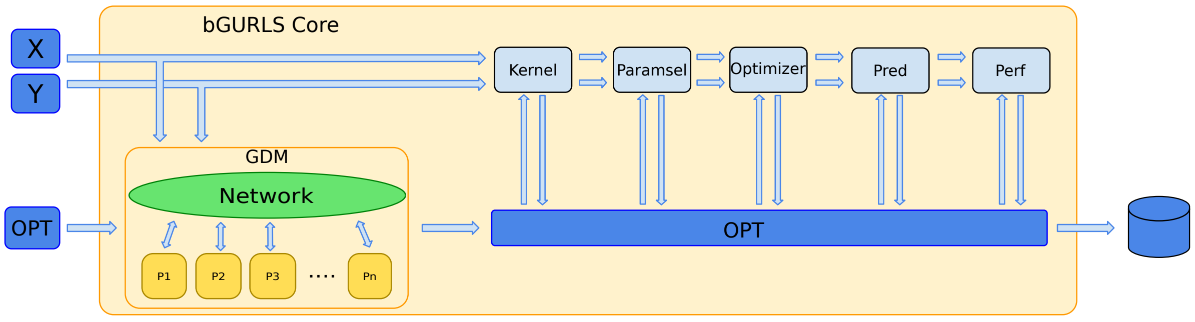

Figure 1: BGURLS design.

The options' structure is built through the `bigdefopt` function with default fields and values. Most of the main fields in the options' structure are the same as in GURLS, however `bgurls` requires the options' structure to have the additional field `files`, which must be a structure with fields:

- `Xva_filename`: the prefix of the files that constitute the bigarray of the input data used for validation
- `yva_filename`: the prefix of the files that constitute the bigarray of the labels vector used for validation
- `pred_filename`: the prefix of the files that constitute the bigarray of the predicted labels for the test set
- `XtX_filename`: the name of the files where pre-computed matrix $X'X$ is stored
- `Xty_filename`: the name of the files where pre-computed matrix $Xt'y$ is stored
- `XvatXva_filename`: the name of the files where pre-computed matrix $X'_{va}X_{va}$ is stored
- `Xvatyva_filename`: the name of the files where pre-computed matrix $X'_{va}y_{va}$ is stored

2.0.1 BGURLS example

Let us consider the demo `bigdemoA.m` in the demo directory to better understand the usage of BGURLS. The demo computes a linear classifier with the regularization parameter chosen via hold-out validation, and then evaluate the prediction accuracy on a test set. The data set used in the demo is the `bio` data set used in [1], which is saved in the demo directory as a .zip file, 'bio_unique.zip', containing two files:

- 'X.csv': containing the input $n \times d$ data matrix, where n is the number of samples (24,942) and d is the number of variables (68)
- 'Y.csv': containing the input $n \times 1$ label vector

Note that the bio data is not properly a big data set, as it could reside in memory, however it is large enough to make it reasonable to use BGURLS.

In the following we examine the salient part of the demo in details. First unzip the data file

```
unzip('bio_unique.zip','bio_unique')
```

and set the name of the data files

```
filenameX = 'bio_unique/X.csv'; %nxd input data matrix
filenameY = 'bio_unique/y.csv'; %nx1 or 1xn labels vector
```

Now set the size of the blocks for the bigarrays (matrices of size `blocksize`× `d` must fit into memory):

```
blocksize = 1000;
```

the fraction of total samples to be used for testing:

```
test_hoproportion = .2;
```

the fraction of training samples to be used for validation:

```
va_hoproportion = .2;
```

and the directory where all processed data is going to be stored:

```
dpath = 'bio_data_processed';
```

Now set the prefix of the files that will constitute the bigarrays

```
mkdir(dpath)
files.Xtrain_filename = fullfile(dpath, 'bigarrays/Xtrain');
files.ytrain_filename = fullfile(dpath, 'bigarrays/ytrain');
files.Xtest_filename = fullfile(dpath, 'bigarrays/Xtest');
files.ytest_filename = fullfile(dpath, 'bigarrays/ytes');
files.Xva_filename = fullfile(dpath, 'bigarrays/Xva');
files.yva_filename = fullfile(dpath, 'bigarrays/yva');
```

and the name of the files where pre-computed matrices will be stored

```
files.XtX_filename = fullfile(dpath, 'XtX.mat');
files.Xty_filename = fullfile(dpath, 'Xty.mat');
files.XvatXva_filename = fullfile(dpath,'XvatXva.mat');
files.Xvatyva_filename = fullfile(dpath, 'Xvatyva.mat');
```

We are now ready to prepare the data for BGURLS. The following line of command reads files `filenameX` and `filenameY` blockwise – thus avoiding to load all file at the same time– and stores them in the bigarray format, after having split the data into train, validation and test set

```
bigTrainTestPrepare(filenameX, filenameY,files,blocksize,...
va_hoproportion,test_hoproportion)
```

Bigarrays are now stored in the file names specified in the structure `files`. We can now pre-compute matrices that will be recursively used in the training phase, and store them in the file names specified in the structure `files`

```
bigMatricesBuild(files)
```

The data set is now prepared for running the learning pipeline with the `bgurls` command. This phase behaves almost completely as in GURLS. The only differences are that:

- we need not to load the data into memory, but simply "load" the bigarray, that is load the information necessary to access the data blockwise.
- we have to specify in the opt structure the path where the already computed matrix multiplications, and bigarrays for validation data are stored.

Let us first define the option structure as in GURLS

```
name = fullfile(wpath,'gurls');
opt = bigdefopt(name);
opt.seq = {'paramsel:dhoprimal','rls:dprimal','pred:primal','perf:macroavg'};
opt.process{1} = [2,2,0,0];
opt.process{2} = [3,3,2,2];
```

Note that no task is defined for the `split` category, as data has already been split in the preprocessing phase and bigarrays for validation were built. In the following fragment of code we add to the options' structure the information relative to the already computed matrix multiplications and to the validation bigarrays

```
opt.files = files;
opt.files = rmfield(opt.files,{'Xtrain_filename';'ytrain_filename';...
'Xtest_filename';'ytest_filename'}); %not used by bgurls
opt.files.pred_filename = fullfile(dpath, 'bigarrays/pred');
```

Note that we have also defined where the predicted labels shall be stored as bigarray.

Now we have to "load" bigarrays for training

```
X = bigarray.Obj(files.Xtrain_filename);
y = bigarray.Obj(files.ytrain_filename);
X.Transpose(true);
y.Transpose(true);
```

and run `bgurls` on the training set

```
bgurls(X,y,opt,1)
```

In order to run the testing process, we first have to "load" bigarrays variables for test data

```
X = bigarray.Obj(files.Xtest_filename);
y = bigarray.Obj(files.ytest_filename);
X.Transpose(true);
y.Transpose(true);
```

and then we can finally run `bgurls` on the test set

```
bgurls(X,y,opt,2);
```

Now you should have a mat file named `gurls.mat` in your path. This file contains all the information about your experiment. If you want to see the mean accuracy, for example, load the file in your workspace and type

```
>> mean(opt.perf.acc)
```

If you are interested in visualizing or printing stats and facts about your experiment, check the documentation about the summarizing functions in the gurls package.

2.0.2 Dealing with other data formats

Other two demos can be found in the 'demo' directory. The three demos differ in the format of the input data, as we tried to provide examples for the most common data formats. The data set used in `bigdemoB` is again the bio data set, though in a slightly different format as it is already split into train and test data. The `bigTrainPrepare` and `bigTestPrepare` take care of preparing the train and test set separately.

The data set used in `bigdemoC` is the ImageNet data set, which is automatically downloaded from `http://bratwurst.mit.edu/sbow.tar`, when running the demo. This data set is stored in 1000 .mat files where the i-th file contains the variable `x` which is a $d \times n_i$ input data matrix for the n_i samples of class i. The `bigTrainTestPrepare_manyfiles` takes care of preparing the bigarrays for the ImageNet data format. Note that, while the bio data is not properly a big data set, the ImageNet occupies about 1G of RAM and can thus be called a big data set.

In order to run BGURLS on other data formats, one can simply use `bigdemoA` after having substituted the line

```
bigTrainTestPrepare(filenameX, filenameY,files,blocksize,...
    va_hoproportion,test_hoproportion)
```

with a suitable fragment of code. The remainder of the data preparation, that is the computation and storage of the relevant matrices, can be left unchanged.

3 Available methods

In this section we summarize all the available tasks that have been implemented in the 2 modules, GURLS and BGURLS.

task category	**description**	**available tasks**
bigparamsel	performs selection of the regularization parameter λ	hoprimal calibratesgd
bigrls	solves RLS optimization problem	primal pegasos
bigpred	predicts the labels	primal
bigperf	assess prediction performance	macroavg

Table 1: List of BGURLS tasks organized by category.

Bibliography

[1] F. Lauer and Y. Guermeur. Msvmpack: A multi-class support vector machine package. *The Journal of Machine Learning Research*, 12:2293–2296, 2011.