GURLS

Effective machine learning made easy

Alessandro Rudi
Carlo Ciliberto and Lorenzo Rosasco

MLCC 2015
Machine Learning in a “Nutshell”

A share of big data + “Favorite” pre-processing + Statistical modeling = Data representation
Machine Learning in a “Nutshell”

A share of big data

“Favorite” pre-processing

Data representation

Statistical modeling

=
Machine Learning in a “Nutshell”

Several Algorithms Available

Data representation + Statistical modeling = Favorite

Several Algorithms Available

A share of big data

Several Algorithms Available

A share of big data

Several Algorithms Available

A share of big data
Machine Learning in a “Nutshell”

Several Algorithms Available

How to select a good hyperparameters range?

How to select a good hyperparameters range?

Statistical modeling

Data

How to select a good hyperparameters range?
Machine Learning in a “Nutshell”

Several Algorithms Available

How to select a good hyperparameters range?

Statistical modeling

Lots of Boilerplate Code
GURLS Automates this boring part!
Multiple “Depths”

Casual User

Advanced User

Machine Learner
GURLS as a Casual User

A share of big data

“Favorite” pre-processing

Statistical modeling

Data representation

\[
\text{model} = \text{gurls\_train}(X_{tr}, y_{tr}) \\
\text{ypred} = \text{gurls\_test}(\text{model}, X_{ts})
\]
What if we wanted to change... e.g. the kernel?

A share of big data

“Favorite” pre-processing

Statistical modeling

Data representation

model = gurls_train(Xtr, ytr, 'kernel','rbf','kerpar',0.1)

ypred = gurls_test(model,Xts)
model = gurls_train(Xtr, ytr, 'par1', val1, 'par2', val2, ...)

ypred = gurls_test(model, Xts)
Features of the Library

**Algorithms**
- KRLS with
  - Tikhonov
  - Landweber
  - Nu-method
  - Truncated-SVD
  - Conjugate Gradient
- Kernel Logistic Regression
- Gaussian Processes
- Random Features

**Kernels**
- Linear
- Polynomial
- RBF
- Chisquared
- Quasi-periodic

**Model Selection**
- Performance Measures:
  - RMSE
  - Macroavg
  - Precision/Recall
- Automatic Parameter Tuning:
  - Automatic split & randomization
  - Hold out
  - Leave-one out
  - k-fold
- Automatic Range for Hyperparameters:

**MATLAB & C++ Interfaces**
Example

**Linear RLS**

**KRLS - RBF Kernel**

**Landweber Filter**
Example

Linear RLS

KRLS - RBF Kernel
Landweber Filter
Example

Linear RLS

KRLS - RBF Kernel
Landweber Filter

gurls_train(Xtr, ytr,
   'algorithm','lrls')

gurls_train(Xtr, ytr,
   'algorithm','krls',
   'kernelfun','rbf',
   'filter','land')
UNDER THE HOOD

The Pipeline
Under the hood - The Pipeline

highly modular, i.e. reuse of code components
Under the hood - The Pipeline

highly modular, i.e. reuse of code components
Example: from gurls_train to the pipeline

gurls_train(Xtr, ytr, 'algorithm', 'lrls')
Example: from gurls_train to the pipeline

gurls_train(Xtr, ytr,'algorithm','lrls')

With the pipeline...

```python
opt = gurls_defopt('');
opt.seq = {'split:ho', 'paramsel:hoprimal', 'rls:primal'};
opt.process{1} = [2, 2, 2];

model = gurls(Xtr, ytr, opt, 1);
```
Examples with what we have seen

```python
gurls_train(Xtr, ytr, 'algorithm', 'krls', 'kernelfun', 'rbf', 'filter', 'land')
```
Examples with what we have seen

gurls_train(Xtr, ytr, 'algorithm', 'krls', 'kernelfun', 'rbf', 'filter', 'land')

With the pipeline...

opt = gurls_defopt('');
opt.newprop('paramsell.guesses', 100);
opt.paramsell.optimizer = str2func('rls_landweberdual');
opt.kernel.func = str2func('kernel_rbf');
opt.process{1} = [2, 2, 2, 2];

model = gurls(Xtr, ytr, opt, 1);
Structure of a stage

gurls_train(Xtr, ytr, 'algorithm', 'lrls')

function [cfr] = rls_primal (X, y, opt)
% rls_primal(X,y,opt)
% computes a classifier for the primal formulation of RLS.
% The regularization parameter is set to the one found in opt.paramsel.
% In case of multiclass problems, the regularizers need to be combined with the opt.
% singlelambda function.
%
% INPUTS:
% -OPT: struct of options with the following fields:
%   fields that need to be set through previous gurls tasks:
%     - paramsel.lambda (set by the paramsel_* routines)
%   fields with default values set through the defopt function:
%     - singlelambda
%
% For more information on standard OPT fields
% see also defopt
%
% OUTPUT: struct with the following fields:
All stages have same interface

<stage>_<funcname>(X, y, opt)
All blocks have same interface

```
<stage>_<funcname>(X, y, opt)
```

```
function [kernel] = kernel_rbf(X, y, opt)
% kernel_rbf(opt)
% Computes the kernel matrix for a Gaussian kernel.
% INPUTS:
% -OPT: struct with the following fields:
%   - kernel_type (set by the kernel_* routines)
% OUTPUT: struct with the following fields:
%   - kernel
% Code adapted from PASCAL VOC 2007 competition.
```

```
function [cfr] = rls_primal (X, y, opt)
% rls_primal(X,y,opt)
% computes a classifier for the primal formulation of RLS.
% The regularization parameter is set to the one found in opt.paramsel.
% In case of multiclass problems, the regularizers need to be computed
% using a single lambda function.
% INPUTS:
% -OPT: struct of options with the following fields:
%   - fields that need to be set through previous gurls tasks:
%     - paramsel.lambdas (set by the paramsel_* routines)
%     - fields with default values set through the defopt function:
%       - singlelambda
% For more information on standard OPT fields
% see also defopt
% OUTPUT: struct with the following fields:
%   - W: matrix of coefficient vectors of rls estimator for each class
%   - C: empty matrix
```

```
function [p] = perf_precrec(X,y, opt)
% perf_precrec(opt)
% Computes the average precision per class.
% INPUTS:
% -y: labels matrix
% -fields that need to be set through previous gurls tasks:
%   - pred (set by the pred_* routines)
% OUTPUT: struct with the following fields:
% -acc: array of prediction accuracy fields
% -forho: ""
% -forplot: ""
% if isstruct(opt.pred)
Results - Classification Benchmarks
Results - Comparison with the state of the art

**PubFig83**

[http://www.eecs.harvard.edu/~zak/pubfig83/](http://www.eecs.harvard.edu/~zak/pubfig83/)

<table>
<thead>
<tr>
<th>Package</th>
<th>Kernel</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GURLS</td>
<td>Linear</td>
<td>87%</td>
<td>0h13m</td>
</tr>
<tr>
<td>LIBSVM</td>
<td>Linear</td>
<td>76%</td>
<td>5h20m</td>
</tr>
<tr>
<td>GURLS</td>
<td>RBF with selection</td>
<td>88%</td>
<td>5h51m</td>
</tr>
<tr>
<td>GURLS</td>
<td>RBF = 25th PCT of DST</td>
<td>87%</td>
<td>0h14m</td>
</tr>
<tr>
<td>LIBSVM</td>
<td>RBF = 25th PCT of DST</td>
<td>76%</td>
<td>4h18m</td>
</tr>
</tbody>
</table>
Results - Object Recognition in Robotics

The graph shows the mean accuracy of different object recognition methods as a function of the number of training examples per class. The methods compared include HMAX-SVM, SC-RLS, BOW-RLS, HMAX-KNN, SC-KNN, BOW-KNN, and SIFT-KNN. The performance is evaluated across various numbers of training examples per class, with the accuracy increasing with more training data.

There is also a bar chart comparing the frame per second (FPS) for different coding methods: HMAX, SC, BOW, and SIFT. The BOW method appears to offer the highest FPS, suitable for real-time applications.
Download or Pull from Github: [https://github.com/LCSL/GURLS](https://github.com/LCSL/GURLS)

Reference Paper:

**GURLS: a Least Squares Library for Supervised Learning.**  
A. Tacchetti, P. K. Mallapragada, M. Santoro and L. Rosasco  
*Journal of Machine Learning Research*
This afternoon at **2PM:**
a Lab to get acquainted with GURLS

+ 

A small **machine learning challenge**
(just for fun)