Guide to gurls_train and gurls_test

October 15, 2014

A very common machine learning task consists in training a model on a given dataset and then to test it. Here the two steps are made simple:

```
model = gurls_train(Xtrain, ytrain);
ypredicted = gurls_test(model, Xtest);
```

In particular the functions `gurls_train, gurls_test` solve different kinds of supervised learning problems: regression, classification, multiclass classification. The functions automatically recognize the kind of problem, find the most suitable model and learning algorithm for the given dataset and select the appropriate metaparameters, if not specified in the options.

1 gurls_train

The function train has different signatures

```
model = gurls_train(Xtrain, ytrain);
model = gurls_train(Xtrain, ytrain, options);
model = gurls_train(Xtrain, ytrain, 'optname1', optval1, 'optname2', optvalue2, ...);
```

`Xtrain` can be a $n \times d$ matrix of the form `[x1; x2;... xn]` where `xi` are the n inputs of d dimensions. Otherwise it can be a precomputed Gram matrix (thus a $n \times n$ positive semidefinite matrix).

`ytrain` is a $n \times T$ matrix where the i-th row is the vector of labels associated to the i-th sample point `xi` in `Xtrain`. `ytrain` is a matrix of real numbers in the regression tasks, while a matrix of $-1, +1$ in the classification or multiclassification ones, where the `ytrain(i,j)` is 1 if `xi` belongs to the class j otherwise it is -1. This notation is general and covers the case of overlapping classes. Note that for the multiclass case another allowed format for `ytrain` is a $n \times 1$ vectors of numbers in $\{1, 2, .., T\}$, where `ytrain(i)=j` if `xi` belongs to the class j.

`options` is a list of options and is defined as follows

```
options = struct('optname1', optvalue1, 'optname2', optvale2, ...)
```

if `options` is defined in this way the second and the third signatures are equivalent.

`'optname1', optval1,...` these are options with the associated values that are passed to the train function. Note that all the options specified here will be passed to the underlying modules of gurls. The following is a list of options with the possible values.

- `'datatype'`: (automatically deduced, if not specified) can be `'vector'` when the input type is a matrix with the format specified for `Xtrain`, or `'kernel'` when `Xtrain` is a Gram matrix.

- `'problem'`: (automatically deduced, if not specified) can be `'regression'` or `'classification'`, see the definition of `'ytrain'`
- `'algorithm'`: it specifies the algorithm to use for solving the problem, it can be
  - `'lrls'`: (Only when the datatype is vector) it is linear regularized least squares,
  - `'krls'`: (default) it is kernel regularized least squares,
  - `'krlsrf'`: it is kernel regularized least squares with random features,
  - `'gpr'`: it is gaussian processes regression.
- `'filter'`: (automatically deduced, if not specified) it specifies the filter used by the algorithm, it can be
  - `'tikh'`: (default) tikhonov filter,
  - `'land'`: landweber iterative filter. To use this, specify the parameter `regrange` with the range of metaparameters to test (e.g. `1:100`),
  - `'nu'`: nu-method iterative filter. To use this, specify the parameter `regrange` with the range of metaparameters to test (e.g. `1:100`),
  - `'conjgrad'`: conjugate gradient filter,
  - `'randtikh'`: randomized tikhonov filter.
- `'kernelfun'`: it specifies the kernel function used by the algorithm. It applies for all the algorithms except for lrls that is linear by default. Possible values are
  - `'linear'`: linear kernel,
  - `'rbf'`: (default) gaussian kernel,
  - `'quasiperiodic'`: quasiperiodic kernel, (the options `paramsel.alpha` and `period` must be specified).
  - `'chisquared'`: chi-square kernel,
  - `'datatype'`: used when `'Xtrain'` is a precomputed Gram matrix.
- `'partuning'`: specifies the cross validation approach to be used for automatically selecting the metaparameters. It can be `'loo'` for leave one out, or `'ho'` for hold-out. The default is `ho`
- `'perfm'`: measure of the performances of the model. It can be
  - `'rmse'`: root mean square error (default when the problem is of regression type),
  - `'macroavg'`: macro average (default when the problem is of classification type),
  - `'precrec'`: precision recall,
  - `'gpregr'`: specific performance measure for the gaussian process regression (mandatory when the algorithm is gpr).
- `'pars'`: (automatically deduced, if not specified) it tells the system which metaparameters must be selected.
  - `'none'`: both the regularization and the kernel metaparameters are specified in the options,
  - `'reg'`: the regularization metaparameter has to be found by cross validation,
  - `'ker'`: the kernel metaparameter has to be found by cross validation,

  - 'all': both the regularization and the kernel metaparameter has to be found by cross validation.

- 'regrange': the range where the automatic cross validation will search for the best regularization parameter. It must be specified for the nu and land filters, otherwise it is automatically computed by the system.
- 'kerrange': the range where the automatic cross validation will search for the best kernel parameter. It is automatically computed by the system, if not specified.
- 'regpar': specific value for the regularization metaparameter.
- 'kerpar': specific value for the kernel metaparameter.

2 gurls_test

```
ypredicted = gurls_test(model, Xtest)
[ypredicted, accuracy] = gurls_test(model, Xtest, ytest)
[ypredicted, accuracy] = gurls_test(model, Xtest, ytest, perfmeas)
```

The first signature computes the predicted labels `ypredicted` associated to the test dataset. The second computes the predicted labels and their accuracy with respect to the test label `ytest`. In this case the accuracy is computed by the performance measure used in the training phase. In order to compute it with a different performance measure, use the third signature. Note that the format of `Xtest` and of `ytest` must be the same of `Xtrain` and `ytrain`.

3 GurlsOptions

The `GurlsOptions` is an internal class intended for collecting all the variables needed by the function `gurls`. It is the class of `model` that is the model learned by `gurls_train` and is the class of `opt` that is the object automatically produced by `gurls_defopt`. Let `namevar` refers to a variable of interest, use

- `isprop(opt, 'namevar')` to check if it is in `opt`
- `obj.namevar` to read its value
- `obj.namevar = value` to write its value (when the variable is in `obj`)
- `opt.newprop('namevar', value)` to add it to `opt`. If the variable is already present, it will be overwritten.

Moreover you can

- add multiple variables by using `opt.newprops(struct('name1',value1,'name2', value2,...))`. The variables in opt with the same name of the ones in oldOpt will be overwritten.
- copy the content of a existent opt in a new one by using`opt.newprops(oldOpt)`. The variables in opt with the same name of the ones in oldOpt will be overwritten.
- create subvariables with `opt.newprop('name1.name2',value)`. It will be accessed by `opt.name1.name2`. Note that now `opt.name1` contains `struct('name2', value)`. Thus the code

```
opt.newprop('name1.name2',value1);
opt.newprop('name1.name3',value2);
```

is equivalent to write

```
opt.newprop('name1', struct());
opt.name1.name2 = value1;
opt.name1.name3 = value2;
```

Note that the object `opt` is of class `GurlsOptions`, that is a subclass of `handle`, thus it is passed by reference and not by value as shown in the following example

```
opt = defopt('foobar');
opt.newprop('v1', 5);
fun(opt);
disp(opt.v1); % it will display 6

function fun(opt)
  if isprop(opt, 'v1')
    opt.v1 = opt.v1 + 1;
  end
end
```