Package ‘sglOptim’

May 8, 2019

Type Package
Title Generic Sparse Group Lasso Solver
Version 1.3.8
Date 2019-05-07
Description Fast generic solver for sparse group lasso optimization problems. The loss (objective) function must be defined in a C++ module. The optimization problem is solved using a coordinate gradient descent algorithm. Convergence of the algorithm is established (see reference) and the algorithm is applicable to a broad class of loss functions. Use of parallel computing for cross validation and subsampling is supported through the ‘foreach’ and ‘doParallel’ packages. Development version is on GitHub, please report package issues on GitHub.

URL https://dx.doi.org/10.1016/j.csda.2013.06.004,
https://github.com/nielsrhusan/sglOptim

BugReports https://github.com/nielsrhusan/sglOptim/issues

License GPL (>= 2)
LazyLoad yes
Imports methods, stats, tools, utils
Depends R (>= 3.2.4), Matrix, foreach, doParallel
LinkingTo Rcpp, RcppProgress, RcppArmadillo, BH
Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 6.1.1
NeedsCompilation yes

Author Martin Vincent [aut],
Niels Richard Hansen [ctb, cre]

Maintainer Niels Richard Hansen <niels.r.hansen@math.ku.dk>

Repository CRAN

Date/Publication 2019-05-07 22:10:03 UTC
R topics documented:

add_data .................................................. 3
add_data.sgl .............................................. 3
best_model .................................................. 4
best_model.sgl ............................................. 4
coeff.sgl .................................................. 5
compute_error ........................................... 6
create.sgl ................................................... 6
element_class ............................................. 7
Err .......................................................... 8
Err.sgl ...................................................... 9
features ..................................................... 10
features.sgl ............................................... 10
features_stat ............................................ 11
features_stat.sgl ........................................ 11
get_coef .................................................... 12
models ....................................................... 12
models.sgl ................................................ 13
nmod ........................................................ 13
nmod.sgl .................................................... 14
parameters ............................................... 15
parameters.sgl .......................................... 15
parameters_stat ......................................... 16
parameters_stat.sgl .................................... 16
prepare.args ............................................ 17
prepare.args.sgl ........................................ 18
prepare_data ............................................. 19
print_with_metric_prefix ............................... 19
rearrange .................................................. 20
rearrange.sgl .............................................. 21
sgl.algorithm.config .................................. 21
sgl.c.config .............................................. 23
sgl.standard.config .................................... 23
sglOptim ................................................... 24
sgl_cv ...................................................... 25
sgl_fit ...................................................... 26
sgl_lambda_sequence .................................. 28
sgl_predict ............................................... 29
sgl_print ................................................... 30
sgl_subsampling ....................................... 31
sgl_test .................................................... 33
sparseMatrix_from_C_format .......................... 34
sparseMatrix_to_C_format ............................. 34
subsample .................................................. 35
subsample.sgl ............................................ 35
test.data ................................................... 36
test_rtools ............................................... 36
add_data

Add data to a sgldata data object

Description

Add data to a sgldata data object

Usage

add_data(data, x, name, ...)

Arguments

data: sgldata object
x: data object to add,
name: name of data object
... additional parameters

Author(s)

Martin Vincent

---

add_data.sgldata

Add data to a sgldata data object

Description

Add data to a sgldata data object

Usage

## S3 method for class 'sgldata'
add_data(data, x, name, default = NULL,
  type = element_class(x), sparse = is(x, "sparseMatrix"), ...)

Arguments

data: sgldata object
x: matrix or vector,
name: name of the data object
default: default value, returned if x is null
type: data type, 'numeric' or 'integer'sparse: if TRUE y will be treated as sparse, if FALSE y will be treated as dens.
... not used
Author(s)
Martin Vincent

See Also
Other sgldata: `create.sgldata`, `prepare.args.sgldata`, `prepare.args.prepare_data`, `rearrange.sgldata`, `subsample.sgldata`

---

### Description

Returns the index of the best model

### Usage

```
best_model(object, ...)
```

### Arguments

- `object`: an sgl object
- `...`: additional parameters (optional)

### Value

index of the best model.

---

Author(s)
Martin Vincent

---

### Description

Returns the index of the best model, in terms of lowest error rate

### Usage

```
## S3 method for class 'sgl'
best_model(object, pkg, ...)
```
extracting the nonzero coefficients

This function returns the nonzero coefficients (that is the nonzero entries of the \textit{beta} matrices)

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'sgl'
coef(object, index = 1:nmod(object), parameter = "beta",
    ...)
\end{verbatim}

\textbf{Arguments}

- \textit{object} \hspace{1cm} a \texttt{sgl} object
- \textit{index} \hspace{1cm} indices of the models
- \textit{parameter} \hspace{1cm} name of the parameter (default is 'beta')
- \textit{...} \hspace{1cm} ignored

\textbf{Value}

- a list of with nonzero coefficients of the models

\textbf{Author(s)}

Martin Vincent
**compute_error**  
*Helper function for computing error rates*

**Description**

This function can be used to compute mean error rates. It is consist with the use cases of the `Err` genetic function. The loss function should be of the form `function(x, y)` and must return a single numeric number, with `x` a list of true responses and `y` a list of responses (one list element for each sample).

**Usage**

```r
compute_error(x, response_name, true_response, loss,
              transposed_response = FALSE)
```

**Arguments**

- `x`  
  sgl object containing responses
- `response_name`  
  the name of the response.
- `true_response`  
  the true response
- `loss`  
  the loss function.
- `transposed_response`  
  have the response list been transposed with `transpose_response_elements`

**Value**

a vector or matrix with the computed error rates

**Author(s)**

Martin Vincent

---

**create.sgldata**  
*Create a sgldata object*

**Description**

Creates a sgldata object from a design matrix and an optional response vector or matrix.

**Usage**

```r
create.sgldata(x, y, response_dimension = .get_response_dimension(y),
                response_names = .get_response_names(y), sparseX = is(x,
                "sparseMatrix"), sparseY = is(y, "sparseMatrix"),
                typeX = element_class(x), typeY = element_class(y))
```
element_class

Arguments

x the design matrix, a matrix of size \( N \times p \) (will be parsed to the loss module as \( X \)).
y the responses, NULL, a vector or a matrix (will be parsed to the loss module as matrix \( Y \)).
response_dimension number of models, that is the dimension of the returned response.
response_names names of models, that is the names of the elements of the returned response.
sparseX if TRUE \( x \) will be treated as sparse, if FALSE \( x \) will be treated as dens.
sparseY if TRUE \( y \) will be treated as sparse, if FALSE \( y \) will be treated as dens.
typeX type of the elements of \( x \).
typeY type of the elements of \( y \).

Author(s)

Martin Vincent

See Also

Other sglgdata: add_data.sglgdata, prepare.args.sglgdata, prepare.args, prepare_data, rearrange.sglgdata, subsample.sglgdata

Retur the element class of an object.

Description

Return the element class of an object. The object must be a matrix, vector or NULL. The element class of NULL is NULL.

Usage

element_class(x)

Arguments

x a matrix, vector or NULL

Author(s)

Martin Vincent
Err

Generic function for computing error rates

Description

Compute and returns an error rate for each model contained in \( x \). See details for generic use cases.

Usage

\( \text{Err}(\text{object, data, response, ...}) \)

Arguments

- \text{object}: an object
- \text{data}: a data object
- \text{response}: a response object
- ...: additional parameters (optional)

Details

The following generic use case should be supported (see for example \texttt{msgl} package for an implementation):

1. With fit a sgl fit object with models estimated using \( x \) data, the code
   \( \text{Err}(\text{fit, x}) \)
   should return a vector with the \textit{training errors} of the models.

2. With \( x.\text{new} \) a new data set with known responses \( \text{response.}\text{new} \), the code
   \( \text{Err}(\text{fit, x.new, response.new}) \)
   should return a vector with the errors of the models when applied to the new data set.

3. With fit.\text{cv} a sgl cross validation object, the code
   \( \text{Err}(\text{fit.cv}) \)
   should return a vector with estimates of the \textit{expected generalization errors} of the models (i.e. the cross validation errors).

4. If subsampling is supported then, with fit.\text{sub} a sgl subsampling object, the code
   \( \text{Err}(\text{fit.sub}) \)
   should return a matrix with the test errors (each column corresponding to a model, i.e. rows corresponds to tests).

Value

- a vector of length \( n\text{mod}(\text{object}) \) or a matrix with \( n\text{mod}(\text{object}) \) columns containing error rates for the models
Description

Compute and return the root-mean-square error for each model. This method is only intended for testing.

The root-mean-square error (RMSE) is

\[
\frac{1}{K} \sum_{i=1}^{K} \sqrt{\frac{1}{N} \sum_{j=1}^{N} Y_{ji} - (X_\hat{\beta})_{ji}}
\]

RMSE is the default error.

Usage

```r
## S3 method for class 'sgl'
Err(object, data = NULL, response = object$Y.true, ...)
```

Arguments

- `object`: a lsgl object.
- `data`: a design matrix (the \(X\) matrix).
- `response`: a matrix of the true responses (the \(Y\) matrix).
- `...`: ignored.

Value

- a vector of errors.

Author(s)

Martin Vincent
**features**

*Extracts nonzero features*

**Description**

Generic function for extracting nonzero features.

**Usage**

`features(object, NNN)`

**Arguments**

- **object**: an object
- **NNN**: additional parameters (optional)

**Value**

A list of length `nmod(x)` containing the nonzero features of the models.

**Author(s)**

Martin Vincent

---

**features.sgl**

*Extracting nonzero features*

**Description**

Extract the nonzero features of the fitted models.

**Usage**

```r
# S3 method for class 'sgl'
features(object, ...)  
```

**Arguments**

- **object**: a sgl object
- **...**: ignored

**Value**

A list of vectors containing the nonzero features (that is nonzero columns of the `beta` matrices).

**Author(s)**

Martin Vincent
features_stat

Extract feature statistics

Description

Generic function for extracting feature statistics.

Usage

features_stat(object, ...)

Arguments

object an object
... additional parameters (optional)

Value

an object containing the computed statistics.

Author(s)

Martin Vincent

features_stat.sgl

Extract feature statistics

Description

Extracts the number of nonzero features (or group) in each model.

Usage

## S3 method for class 'sgl'
features_stat(object, ...)

Arguments

object an object
... ignored

Value

a vector of length nmod(x) or a matrix containing the number of nonzero features (or group) of the models.
Author(s)

Martin Vincent

---

**get_coef**

*Get the nonzero coefficients*

**Description**

Extracting nonzero coefficients from list (of lists) of matrices

**Usage**

```r
get_coef(object, index = 1:length(object))
```

**Arguments**

- `object` a list of lists of matrices or a list of matrices
- `index` indices to be extracted from

**Value**

a list (of lists) with the nonzero coefficients

Author(s)

Martin Vincent

---

**models**

*Extract fitted models*

**Description**

Generic function for extracting the fitted models. Returns the fitted models.

**Usage**

```r
models(object, index, ...)
```

**Arguments**

- `object` an object
- `index` a vector of indices of the models to be returned
- `...` additional parameters (optional)
models.sgl

Value

a list of length length(index) containing the models

Author(s)

Martin Vincent

models.sgl Extract the estimated models

Description

This function returns the estimated models (that is the beta matrices)

Usage

## S3 method for class 'sgl'
models(object, index = 1:nmod(object), ...)

Arguments

  object    a sgl object
  index     indices of the models to be returned
  ...       ignored

Value

a list of sparse matrices

Author(s)

Martin Vincent

nmod Number of models used for fitting

Description

Generic function for counting the number of models used for fitting the object. Returns the number of models used for fitting. However, note that the objects returned by msgl.cv and msgl.subsampling does not contain any models even though nmod returns a nonzero number.

Usage

nmod(object, ...)

Arguments

object an object
... additional parameters (optional)

Value

the number of models used when fitting the object x.

Author(s)

Martin Vincent

---

**nmod.sgl**

*Returns the number of models in a sgl object*

Description

Returns the number of models used for fitting.

Usage

```r
## S3 method for class 'sgl'
nmod(object, ...)
```

Arguments

object a sgl object
... ignored

Details

Note that cv and subsampling objects does not conating any models even though nmod returns a positiv number.

Value

the number of models in object

Author(s)

Martin Vincent
### parameters

**Extracts nonzero parameters**

- **Description**
  
  Generic function for extracting nonzero parameters for each model.

- **Usage**
  
  ```r
  parameters(object, ...) 
  ```

- **Arguments**
  
  - `object`: an object
  - `...`: additional parameters (optional)

- **Value**
  
  a list of length `nmod(x)` containing the nonzero parameters of the models.

- **Author(s)**
  
  Martin Vincent

### parameters.sgl

**Extracting nonzero parameters**

- **Description**
  
  Extract the nonzero parameters in each model. Only the parameters of nonzero features (columns of the $\beta$ matrices) are returned.

- **Usage**
  
  ```r
  ## S3 method for class 'sgl'
  parameters(object, ...) 
  ```

- **Arguments**
  
  - `object`: a sgl object
  - `...`: ignored

- **Value**
  
  a list of vectors containing the nonzero parameters (that is nonzero entries of the $\beta$ matrices)
Author(s)

Martin Vincent

parameters_stat

Extract parameter statistics

Description

Generic function for extracting parameter statistics.

Usage

parameters_stat(object, ...)

Arguments

object an object
... additional parameters (optional)

Value

an object containing the computed statistics.

Author(s)

Martin Vincent

parameters_stat.sgl

Extracting parameter statistics

Description

Extracts the number of nonzero parameters in each model.

Usage

# S3 method for class 'sgl'
parameters_stat(object, ...)

Arguments

object an object
... ignored
prepare.args

Value

a vector of length \( n \) or a matrix containing the number of nonzero parameters of the models.

Author(s)

Martin Vincent

prepare.args

Generic function for preparing the sgl call arguments

Description

Compute and prepare the sgl call arguments for the objective function

\[
\text{loss}(\text{data})(\beta) + \lambda \left( (1 - \alpha) \sum_{J=1}^{m} \gamma_J \| \beta^{(J)} \|_2 + \alpha \sum_{i=1}^{n} \xi_i |\beta_i| \right)
\]

where \( \text{loss} \) is a loss/objective function. The \( n \) parameters are organized in the parameter matrix \( \beta \) with dimension \( q \times p \). The vector \( \beta^{(J)} \) denotes the \( J \) parameter group, the dimension of \( \beta^{(J)} \) is denote by \( d_J \). The dimensions \( d_J \) must be multiple of \( q \), and \( \beta = (\beta^{(1)} \ldots \beta^{(m)}) \). The group weights \( \gamma \in [0, \infty)^m \) and the parameter weights \( \xi \in [0, \infty)^{qp} \).

Usage

\[
\text{prepare.args}(\text{data}, \ldots)
\]

Arguments

- \( \text{data} \) a data object
- \( \ldots \) additional parameters

Value

- \( \text{block.dim} \) a vector of length \( m \), containing the dimensions \( d_J \) of the groups (i.e. the number of parameters in the groups)
- \( \text{groupWeights} \) a vector of length \( m \), containing the group weights
- \( \text{parameterWeights} \) a matrix of dimension \( q \times p \), containing the parameter weights
- \( \alpha \) the \( \alpha \) value
- \( \text{data} \) the data parsed to the loss module
- \( \text{group.order} \) original order of the columns of \( \beta \). Before sgl routines return \( \beta \) will be reorganized according to this order.

Author(s)

Martin Vincent
See Also

prepare.args.sgldata

Other sgldata: add_data.sgldata, create.sgldata, prepare.args.sgldata, prepare_data, rearrange.sgldata, subsample.sgldata

prepare.args.sgldata  Prepare sgl function arguments

Description

Prepare sgl function arguments using sgldata.

Usage

```r
## S3 method for class 'sgldata'
prepare.args(data, parameterGrouping = NULL,
             groupWeights = NULL, parameterWeights = NULL,
             parameterNames = NULL, alpha, test_data = NULL, ...)
```

Arguments

data  a sgldata object

parameterGrouping  grouping of parameters, a vector of length \( p \). Each element of the vector specifying the group of the parameters in the corresponding column of \( \beta \).

groupWeights  the group weights, a vector of length `length(unique(parameterGrouping))` (the number of groups).

parameterWeights  a matrix of size \( q \times p \), that is the same dimension as \( \beta \).

parameterNames  dim-names of parameters, if NULL `dimnames(parameterWeights)` will be used.

alpha  the \( \alpha \) value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.

test_data  optional test data to be prepared (a sgldata object)

...  not used

Author(s)

Martin Vincent

See Also

Other sgldata: add_data.sgldata, create.sgldata, prepare.args, prepare_data, rearrange.sgldata, subsample.sgldata
prepare_data

Prepare a sgldata data object

Description

Creates a sgldata data object from a matrix or vector

Usage

```r
prepare_data(x, default = NULL, type = "numeric", sparse = is(x, "sparseMatrix"))
```

Arguments

- `x` the matrix,
- `default` default value, returned if `x` is null
- `type` data type, 'numeric' or 'integer'
- `sparse` if TRUE `y` will be treated as sparse, if FALSE `y` will be treated as dens.

Author(s)

Martin Vincent

See Also

Other sgldata: `add_data.sgldata`, `create.sgldata`, `prepare.args.sgldata`, `prepare.args`, `rearrange.sgldata`, `subsample.sgldata`

print_with_metric_prefix

Print a numeric with metric prefix

Description

Print a numeric with metric prefix

Usage

```r
print_with_metric_prefix(x, digits = 3)
```

Arguments

- `x` numeric to be printed
- `digits` number of significant digits
**Value**

- a string

**Author(s)**

- Martin Vincent

---

### Generic rearrange function

**Description**

Rearrange the order of the covariates in the data object.

**Usage**

```r
rearrange(data, covariate.order, ...)
```

**Arguments**

- `data`: a data object
- `covariate.order`: the new order of the covariates
- `...`: additional parameters

**Value**

- a rearranged data object of same class as `data`

**Author(s)**

- Martin Vincent

**See Also**

- `rearrange.sgldata`
rearrange.sgldata  Rearrange sgldata

Description

Rearrange the order of the covariates in a sgldata object.

Usage

```r
## S3 method for class 'sgldata'
rearrange(data, covariate.order, ...)
```

Arguments

- `data`: a sgldata object
- `covariate.order`: the new order of the covariates
- `...`: not used

Value

A sgldata object with the covariates reordered

Author(s)

Martin Vincent

See Also

Other sgldata: `add_data.sgldata`, `create.sgldata`, `prepare.args.sgldata`, `prepare.args`, `prepare_data`, `subsample.sgldata`

sgl.algorithm.config  Create a new algorithm configuration

Description

With the exception of `verbose` it is not recommended to change any of the default values.
Usage

```r
sgl.algorithm.config(tolerance_penalized_main_equation_loop = 1e-10,
          tolerance_penalized_inner_loop_alpha = 1e-04,
          tolerance_penalized_inner_loop_beta = 1,
          tolerance_penalized_medium_loop_alpha = 0.01,
          tolerance_penalized_outer_loop_alpha = 0.01,
          tolerance_penalized_outer_loop_beta = 0,
          tolerance_penalized_outer_loop_gamma = 1e-05,
          use_bound_optimization = TRUE,
          use_stepsize_optimization_in_penalized_loop = TRUE,
          stepsize_opt_penalized_initial_t = 1, stepsize_opt_penalized_a = 0.1,
          stepsize_opt_penalized_b = 0.1, max_iterations_outer = 10000,
          inner_loop_convergence_limit = 10000, verbose = TRUE)
```

Arguments

tolerance_penalized_main_equation_loop
tolerance threshold.
tolerance_penalized_inner_loop_alpha
tolerance threshold.
tolerance_penalized_inner_loop_beta
tolerance threshold.
tolerance_penalized_medium_loop_alpha
tolerance threshold.
tolerance_penalized_outer_loop_alpha
tolerance threshold.
tolerance_penalized_outer_loop_beta
tolerance threshold.
tolerance_penalized_outer_loop_gamma
tolerance threshold.
use_bound_optimization
if TRUE hessian bound check will be used.
use_stepsize_optimization_in_penalized_loop
if TRUE step-size optimization will be used.
stepsize_opt_penalized_initial_t
initial step-size.
stepsize_opt_penalized_a
step-size optimization parameter.
stepsize_opt_penalized_b
step-size optimization parameter.
max_iterations_outer
max iteration of outer loop.
inner_loop_convergence_limit
inner loop convergence limit.
verbose
If TRUE some information, regarding the status of the algorithm, will be printed in the R terminal.
**sgl.c.config**

**Value**
A configuration.

**Author(s)**
Martin Vincent

**Examples**

```r
config.no_progressbar <- sgl.algorithm.config(verbos = FALSE)
```

---

**sgl.c.config** *Fetch information about the C side configuration of the package*

**Description**
Fetch information about the C side configuration of the package

**Usage**

```r
sgl.c.config()
```

**Value**
list

**Author(s)**
Martin Vincent

---

**sgl.standard.config** *Standard algorithm configuration*

**Description**

```r
gsl.standard.config <- sgl.algorithm.config()
```

**Usage**

```r
gsl.standard.config
```

**Format**
An object of class list of length 15.

**Author(s)**
Martin Vincent
sglOptim: Generic Sparse Group Lasso Solver

Description

Fast generic solver for sparse group lasso optimization problems. The loss (objective) function must be defined in a C++ module. The optimization problem is solved using a coordinate gradient descent algorithm. Convergence of the algorithm is established (see reference) and the algorithm is applicable to a broad class of loss functions. Use of parallel computing for cross validation and subsampling is supported through the ‘foreach’ and ‘doParallel’ packages. Development version is on GitHub, please report package issues on GitHub.

Details

Computes a sequence of minimizers (one for each lambda given in the lambda argument) of

\[ \text{loss}(\beta) + \lambda \left( (1 - \alpha) \sum_{J=1}^{m} \gamma_J \| \beta^{(J)} \|_2 + \alpha \sum_{i=1}^{n} \xi_i | \beta_i | \right) \]

where loss is the loss/objective function specified by module_name. The parameters are organized in the parameter matrix \( \beta \) with dimension \( q \times p \). The vector \( \beta^{(J)} \) denotes the \( J \) parameter group. The group weights \( \gamma \in [0, \infty)^m \) and the parameter weights \( \xi = (\xi^{(1)}, \ldots, \xi^{(m)}) \in [0, \infty)^n \) with \( \xi^{(1)} \in [0, \infty)^{n_1}, \ldots, \xi^{(m)} \in [0, \infty)^{n_m} \).

The package includes generic functions for:

- Fitting models using sparse group lasso, that is computing the minimizers of the above equation.
- Cross validation using parallel computing.
- Generic subsampling using parallel computing.
- Applying the fitted models on new data and predicting responses.
- Computing lambda sequences.
- Navigating the models and computing error rates.

Author(s)

Martin Vincent
Generic sparse group lasso cross validation using multiple possessors

Usage

```r
sgl_cv(module_nameL PACKAGEL dataL parameterGrouping = NULLL groupWeights = NULLL parameterWeights = NULLL alphaL lambdaL d = 100L compute_lambda = length(lambda) == 1L fold = 2L sampleGroups = NULLL cv.indices = list()L responses = NULLL max.threads = NULLL use_parallel = FALSEL algorithm.config = sgl.standard.config)
```

Arguments

- **module_name**: reference to objective specific C++ routines.
- **PACKAGE**: name of the calling package.
- **data**: a list of data objects – will be parsed to the specified module.
- **parameterGrouping**: grouping of parameters, a vector of length \( p \). Each element of the vector specifying the group of the parameters in the corresponding column of \( \beta \).
- **groupWeights**: the group weights, a vector of length \( \text{length(unique(parameterGrouping))} \) (the number of groups).
- **parameterWeights**: a matrix of size \( q \times p \).
- **alpha**: the \( \alpha \) value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.
- **lambda**: lambda.min relative to lambda.max (if \( \text{compute_lambda = TRUE} \)) or the lambda sequence for the regularization path, a vector or a list of vectors (of the same length) with the lambda sequence for the subsamples.
- **d**: length of lambda sequence (ignored if \( \text{compute_lambda = FALSE} \)).
- **compute_lambda**: should the lambda sequence be computed
- **fold**: the fold of the cross validation, an integer larger than 1 and less than \( N + 1 \). Ignored if \( \text{cv.indices != NULL} \). If \( \text{fold} \leq \text{max(table(classes))} \) then the data will be split into \( \text{fold} \) disjoint subsets keeping the ratio of classes approximately equal. Otherwise the data will be split into \( \text{fold} \) disjoint subsets without keeping the ratio fixed.
- **sampleGroups**: grouping of samples, the algorithm computing the cv.indices will try to equally divide the groups among the subsamples.
- **cv.indices**: a list of indices of a cross validation splitting. If cv.indices = NULL then a random splitting will be generated using the fold argument.
responses a vector of responses to simplify and return (if NULL (default) no formatting will be done)
max.threads Deprecated (will be removed in 2018), instead use use_parallel = TRUE and registre parallel backend (see package 'doParallel'). The maximal number of threads to be used.
use_parallel If TRUE the foreach loop will use %dopar%. The user must registre the parallel backend.
algorithm.config
   the algorithm configuration to be used.

Value
Y.true the response, that is the y object in data as created by create.sgl.data.
responses content will depend on the C++ response class
cv.indices the cross validation splitting used
features number of features used in the models
parameters number of parameters used in the models
lambda the lambda sequence used.

Author(s)
Martin Vincent

Usage
sgl_fit(module_nameL packageL dataL parameterGrouping = NULL,
   groupWeights = NULL, parameterWeights = NULL, alpha, lambda,
   d = 100, compute_lambda = length(lambda) == 1,
   return_indices = NULL, algorithm.config = sgl.standard.config)
**Arguments**

- `module_name` reference to objective specific C++ routines.
- `PACKAGE` name of the calling package.
- `data` a list of data objects – will be parsed to the specified module.
- `parameterGrouping` grouping of parameters, a vector of length \( p \). Each element of the vector specifying the group of the parameters in the corresponding column of \( \beta \).
- `groupWeights` the group weights, a vector of length `length(unique(parameterGrouping))` (the number of groups).
- `parameterWeights` a matrix of size \( q \times p \).
- `alpha` the \( \alpha \) value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.
- `lambda` lambda.min relative to lambda.max (if `compute_lambda = TRUE`) or the lambda sequence for the regularization path, a vector or a list of vectors (of the same length) with the lambda sequence for the subsamples.
- `d` length of lambda sequence (ignored if `compute_lambda = FALSE`)
- `compute_lambda` should the lambda sequence be computed
- `return_indices` the indices of lambda values for which to return fitted parameters.
- `algorithm.config` the algorithm configuration to be used.

**Value**

- `Y.true` the response, that is the \( y \) object in data as created by `create.sgldata`.
- `beta` the fitted parameters – a list of length `length(return)` with each entry a matrix of size \( q \times (p + 1) \) holding the fitted parameters.
- `loss` the values of the loss function.
- `objective` the values of the objective function (i.e. loss + penalty).
- `lambda` the lambda values used.

**Author(s)**

Martin Vincent
sgl_lambda_sequence  Computing a Lambda Sequence

Description

Computes a decreasing lambda sequence of length d. The sequence ranges from a data determined maximal lambda $\lambda_{\text{max}}$ to the user supplied lambda $\min$.

Usage

\[
\text{sgl\_lambda\_sequence}(\text{module\_name}, \text{PACKAGE}, \text{data}, \text{parameterGrouping} = \text{NULL}, \\
groupWeights = \text{NULL}, \text{parameterWeights} = \text{NULL}, \alpha, d = 100, \\
\lambda_{\min}, \text{algorithm\_config} = \text{sgl\_standard\_config}, \\
\lambda_{\min\_rel} = \text{FALSE})
\]

Arguments

- **module_name**: reference to objective specific C++ routines.
- **PACKAGE**: name of the calling package.
- **data**: list of data objects – will be parsed to the specified module.
- **parameterGrouping**: grouping of parameters, a vector of length $p$. Each element of the vector specifying the group of the parameters in the corresponding column of $\beta$.
- **groupWeights**: group weights, a vector of length $\text{length(unique(parameterGrouping))}$ (the number of groups).
- **parameterWeights**: parameters weights, a matrix of size $q \times p$.
- **alpha**: the $\alpha$ value, 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.
- **d**: the length of the lambda sequence.
- **lambda.min**: the smallest lambda value in the computed sequence.
- **algorithm.config**: the algorithm configuration.
- **lambda.min.rel**: is lambda.min relative to lambda.max? (i.e. actual lambda min used is lambda.min*lambda.max, with lambda.max the computed maximal lambda value)

Value

- a vector of length $d$ containing the compute lambda sequence.

Author(s)

Martin Vincent
sgl_predict

Predict

Description

Predict and return responses as defined in the module.

Usage

sgl_predict(module_name, PACKAGE, object, data, responses = NULL,
            auto_response_names = TRUE, ...)

Arguments

module_name reference to objective specific C++ routines.
PACKAGE name of the calling package.
object a sgl object containing a list of estimated models.
data a list of data objects – will be parsed to the specified module.
responses a vector of responses to simplify and return (if NULL (default) no formatting will be done)
auto_response_names
            set response names
... not used.

Details

If no formatting is done (i.e. if responses = NULL) then the responses field contains a list of lists structured in the following way:

• sample 1
  – model (lambda) index 1
    * response elements
  – model (lambda) index 2
    * response elements
  – ...
• sample 2
  – model (lambda) index 1
    * response elements
  – model (lambda) index 2
    * response elements
  – ...
• ...
...
If `responses = \"rname\"` with `rname` the name of the response then a list at `responses$rname` will be returned. The content of the list will depend on the type of the response.

- **scalar**: a matrix of size $n \times d$ with the responses (where $n$ is the number of samples and $d$ the length of the lambda sequence).
- **vector**: a list of length $d$ with each element a matrix of dimension $n \times q$ containing the responses for the corresponding model (where $q$ is the dimension of the response).
- **matrix**: a list with format samples $\rightarrow$ models $\rightarrow$ the response matrix.

**Value**

- `responses` list of lists structured as described in details. Content of the response elements will depend on the C++ response class
- `lambda` the lambda sequence used.

**Author(s)**

Martin Vincent

---

**sgl_print**

*Print information about sgl object*

**Description**

Prints information about sgl object

**Usage**

```r
sgl_print(x)
```

**Arguments**

- `x` a object of sgl family class

**Author(s)**

Martin Vincent
**sgl_subsampling**

*Generic sparse group lasso subsampling procedure*

**Description**

Subsampling procedure with support parallel computations.

**Usage**

```r
sgl_subsampling(module_nameL packageL dataL parameterGrouping = NULL, 
groupWeights = NULL, parameterWeights = NULL, alpha, lambda, 
d = 100, compute_lambda = length(lambda) == 1, training = NULL, 
test = NULL, responses = NULL, auto_response_names = TRUE, 
collapse = FALSE, max_threads = NULL, use_parallel = FALSE, 
algorithm.config = sgl.standard.config)
```

**Arguments**

- **module_name**: reference to objective specific C++ routines.
- **PACKAGE**: name of the calling package.
- **data**: a list of data objects – will be parsed to the specified module.
- **parameterGrouping**: grouping of parameters, a vector of length \(p\). Each element of the vector specifying the group of the parameters in the corresponding column of \(\beta\).
- **groupWeights**: the group weights, a vector of length \(\text{length(unique(parameterGrouping))}\) (the number of groups).
- **parameterWeights**: a matrix of size \(q \times p\).
- **alpha**: the \(\alpha\) value 0 for group lasso, 1 for lasso, between 0 and 1 gives a sparse group lasso penalty.
- **lambda**: lambda.min relative to lambda.max (if \(\text{compute_lambda = TRUE}\)) or the lambda sequence for the regularization path, a vector or a list of vectors (of the same length) with the lambda sequence for the subsamples.
- **d**: length of lambda sequence (ignored if \(\text{compute_lambda = FALSE}\)).
- **compute_lambda**: should the lambda sequence be computed
- **training**: a list of training samples, each item of the list corresponding to a subsample. Each item in the list must be a vector with the indices of the training samples for the corresponding subsample. The length of the list must equal the length of the test list.
- **test**: a list of test samples, each item of the list corresponding to a subsample. Each item in the list must be vector with the indices of the test samples for the corresponding subsample. The length of the list must equal the length of the training list.
sgl_subsampling

responses a vector of responses to simplify and return (if NULL (default) no formatting will be done)

auto_response_names set response names

collapse if TRUE the results will be collapsed and ordered into one result, resembling the output of sgl_predict (this is only valid if the test samples are not overlapping)

max.threads Deprecated (will be removed in 2018), instead use use_parallel = TRUE and register parallel backend (see package 'doParallel'). The maximal number of threads to be used.

use_parallel If TRUE the foreach loop will use %dopar%. The user must register the parallel backend.

algorithm.config the algorithm configuration to be used.

Details

If no formatting is done (i.e. if responses = NULL) then the responses field contains a list of lists structured in the following way:

subsamples 1:

- sample test[[1]][1]
  - model (lambda) index 1
    * response elements
  - model (lambda) index 2
    * response elements
  - ...

- sample test[[1]][2]
  - model (lambda) index 1
    * response elements
  - model (lambda) index 2
    * response elements
  - ...

- ...

subsamples 2: ...

If responses = "rname" with rname the name of the response then a list at responses$rname will be returned. The content of the list will depend on the type of the response.

- vector A list with format subsamples -> models -> matrix of dimension $n_i \times q$ containing the responses for the corresponding model and subsample (where $q$ is the dimension of the response).

- matrix A list with format subsamples -> samples -> models -> the response matrix.
**Value**

- **y_true** the response, that is the y object in data as created by `create.sgldata`.
- **responses** content will depend on the C++ response class
- **features** number of features used in the models
- **parameters** number of parameters used in the models
- **lambda** the lambda sequences used (a vector or list of length `length(training)`).

**Author(s)**

Martin Vincent

---

**sgl_test**

*Test a sgl-Objective*

---

**Description**

This function will run tests on the gradient and hessian functions implemented in a C++ objective module. Detected problems will be printed to the console.

**Usage**

```r
sgl_test(module_name, PACKAGE, data, parameterGrouping, groupWeights, parameterWeights, algorithm.config = sgl.standard.config)
```

**Arguments**

- **module_name** reference to objective specific C++ routines.
- **PACKAGE** name of the calling package.
- **data** a list of data objects – will be parsed to the specified module.
- **parameterGrouping** grouping of parameters, a vector of length `p`. Each element of the vector specifying the group of the parameters in the corresponding column of `β`.
- **groupWeights** the group weights, a vector of length `length(unique(parameterGrouping))` (the number of groups).
- **parameterWeights** a matrix of size `q × p`.
- **algorithm.config** the algorithm configuration to be used.

**Value**

The number of found problems

**Author(s)**

Martin Vincent
sparseMatrix_from_C_format

Convert to sparse matrix

Description
Convert sparse matrix returned from .Call to sparseMatrix.

Usage
sparseMatrix_from_C_format(x)

Arguments
x .Call returned list

Author(s)
Martin Vincent

sparseMatrix_to_C_format

Prepare sparse matrix for .Call

Description
Prepare sparse matrix for .Call

Usage
sparseMatrix_to_C_format(x)

Arguments
x a sparse matrix

Author(s)
Martin Vincent
**subsample**

| Subsample |
|-----------|---|

**Description**

Pick out a subsample of an object

**Usage**

```r
subsample(data, indices, ...)  
```

**Arguments**

- `data`: a data object
- `indices`: a vector of indices to pick out
- `...`: not used

**Value**

A data object of the same class as `data`

**Author(s)**

Martin Vincent

---

**subsample.sgldata**

<table>
<thead>
<tr>
<th>Subsample sgldata</th>
</tr>
</thead>
</table>

**Description**

Pick out a subsample of a sgldata object

**Usage**

```r
## S3 method for class 'sgldata'  
subsample(data, indices, ...)  
```

**Arguments**

- `data`: a sgldata object
- `indices`: a vector of indices to pick out
- `...`: not used
Value

a sgldata

Author(s)

Martin Vincent

See Also

Other sgldata: `add_data.sgldata`, `create.sgldata`, `prepare.args.sgldata`, `prepare.args`, `prepare_data`, `rearrange.sgldata`

test.data

Simulated data set

Description

This data set is for testing only.

test_rtools

Test internal rtools

Description

This function runs some internal tests and is not intended for users of the package.

Usage

`test_rtools()`

Author(s)

Martin Vincent
**transpose_response_elements**

*Transpose response elements*

---

**Description**

Transpose response elements in a response list and sub lists

**Usage**

```
transpose_response_elements(x)
```

**Arguments**

- `x`  response list or matrix

**Value**

response list with all matrices transposed

**Author(s)**

Martin Vincent
Index

*Topic **datasets**
  sgl.standard.config, 23

*Topic **data**
  test.data, 36

add_data, 3
add_data.sgldata, 3, 7, 18, 19, 21, 36

best_model, 4
best_model.sgl, 4

coeff.sgl, 5
compute_error, 6
create.sgldata, 4, 6, 18, 19, 21, 36

element_class, 7
Err, 8
Err.sgl, 9

features, 10
features.sgl, 10
features.stat, 11
features.stat.sgl, 11

get_coef, 12

models, 12
models.sgl, 13

nmod, 13
nmod.sgl, 14

parameters, 15
parameters.sgl, 15
parameters.stat, 16
parameters.stat.sgl, 16
prepare.args, 4, 7, 17, 18, 19, 21, 36
prepare.args.sgldata, 4, 7, 18, 19, 21, 36
prepare_data, 4, 7, 18, 19, 21, 36
print_with_metric_prefix, 19

rearrange, 20
rearrange.sgldata, 4, 7, 18, 19, 21, 36

ggl.algorithm.config, 21
ggl.c.config, 23
ggl.standard.config, 23
ggl.cv, 25
ggl_fit, 26
ggl_lambda_sequence, 28
ggl_predict, 29
ggl_print, 30
ggl_subsampling, 31
ggl_test, 33
gglOptim, 24
gglOptim-package (sglOptim), 24
sparseMatrix_from_C_format, 34
sparseMatrix_to_C_format, 34
subsample, 35
subsample.sgldata, 4, 7, 18, 19, 21, 35

test.data, 36
test_rtools, 36
transition_response_elements, 37

38