Package: quadrupen (via r-universe)

October 17, 2026

Type Package

Title Sparse and Group Sparse Linear Models

Version 1.1-0

Date 2026-09-17

Description Fits the solution paths of classical sparse regression models with efficient active set algorithms by solving small sub-problems. Include LASSO, SCAD, MCP, (Sparse) Group-LASSO, Cooperative-LASSO, (Group) LAVA, (Generalized) Fused-Lasso and (Generalized) Elastic-Net. Also provides methods for model selection purpose (information criteria, cross-validation, stability selection).

URL https://github.com/jchiquet/quadrupen, https://jchiquet.github.io/quadrupen/

BugReports https://github.com/jchiquet/quadrupen/issues

License GPL (>= 3)

Encoding UTF-8

Language en-US

SystemRequirements GNU make

Depends R (>= 4.1.0)

Imports dplyr, ggplot2, grid, Matrix, methods, parallel, R6, Rcpp, rlang, scales, tidyr, lifecycle

Suggests elasticnet, glmnet, igraph, knitr, lars, MASS, ncvreg, grpreg, rmarkdown, testthat

LinkingTo Rcpp, RcppArmadillo

Collate 'init.R' 'quadrupen-package.R' 'QuadrupenFit-R6Class.R' 'quadrupen-S3Methods.R' 'Regularizers-R6Class.R' 'DataModel-R6Class.R' 'StabilityPath-R6Class.R' 'CrossValidation-R6Class.R' 'InformationCriteria-R6Class.R' 'bounded-reg.R' 'fused-lasso.R' 'sparse_lm.R' 'sparse_lm_aliases.R' 'group_sparse_lm.R' 'group_sparse_lm_aliases.R' 'group-lava.R' 'lava.R' 'utils.R' 'ridge.R' 'RcppExports.R'

Config/roxygen2/version 8.0.0

VignetteBuilder knitr

NeedsCompilation yes

Author Julien Chiquet [aut, cre] (ORCID:
<https://orcid.org/0000-0002-3629-3429>)

Maintainer Julien Chiquet <julien.chiquet@inrae.fr>

Config/pak/sysreqs make

Repository https://cran.r-universe.dev

Date/Publication 2026-09-17 18:00:02 UTC

RemoteUrl https://github.com/cran/quadrupen

RemoteRef HEAD

RemoteSha 5c3e405fe866e01ff0deddb283f60701cc7247c5

Contents

bounded_reg . . . 3
BoundedRegressionFit . . . 6
coef.QuadrupenFit . . . 7
criteria . . . 7
cross_validate . . . 9
CrossValidation . . . 11
DataModel . . . 13
deviance.QuadrupenFit . . . 15
fitted.QuadrupenFit . . . 16
fused_lasso . . . 16
FusedLassoFit . . . 18
group_lava . . . 19
group_sparse_lm . . . 23
GroupLavaFit . . . 29
InformationCriteria . . . 30
isQuadrupenFit . . . 31
lava . . . 32
LavaFit . . . 35
plot.QuadrupenFit . . . 37
predict.QuadrupenFit . . . 37
QuadrupenFit . . . 38
residuals.QuadrupenFit . . . 44
ridge . . . 45
RidgeRegressionFit . . . 47
selection . . . 48
sparse_lm . . . 49
SparseFit . . . 54
SparseGroupFit . . . 55
stability . . . 57
StabilityPath . . . 59

Index **63**

---

bounded_reg *Fit a linear model with infinity-norm plus ridge-like regularization*

Description

Adjust a linear model penalized by a (possibly weighted) ℓ_∞-norm (bounding the magnitude of the parameters) and a (possibly structured) ℓ_2-norm (ridge-like). The solution path is computed at a grid of values for the infinity-penalty, fixing the amount of ℓ_2 regularization. See details for the criterion optimized.

Usage

```
bounded_reg(
  x,
  y,
  lambda1 = NULL,
  lambda2 = 0.01,
  penscale = rep(1, ncol(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = ifelse(nrow(x) <= ncol(x), 0.01, 1e-04),
 maxfeat = ifelse(lambda2 < 0.01, min(nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
  control = list()
)

bounded.reg(
  x,
  y,
  lambda1 = NULL,
  lambda2 = 0.01,
  penscale = rep(1, ncol(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = ifelse(nrow(x) <= ncol(x), 0.01, 1e-04),
 maxfeat = ifelse(lambda2 < 0.01, min(nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
  control = list()
)
```

Arguments

x	matrix of features, possibly sparsely encoded (experimental). Do NOT include intercept. When normalized is TRUE, coefficients will then be rescaled to the original scale.

y	response vector.
lambda1	sequence of decreasing ℓ_1-penalty levels. If NULL (the default), a vector is generated with nlambda1 entries, starting from a guessed level lambda1.max where only the intercept is included, then shrunken to minratio*lambda1.max.
lambda2	real scalar; tunes the ℓ_2 penalty in the Elastic-net. Default is 0.01. Set to 0 to recover the Lasso.
penscale	vector with real positive values that weight the penalty of each feature. Default sets all weights to 1.
struct	matrix structuring the coefficients, possibly sparsely encoded. Must be at least positive semidefinite (this is checked internally). If NULL (the default), the identity matrix is used. See details below.
intercept	logical; indicates if an intercept should be included in the model. Default is TRUE.
normalize	logical; indicates if variables should be normalized to have unit L2 norm before fitting. Default is TRUE.
nlambda1	integer that indicates the number of values to put in the lambda1 vector. Ignored if lambda1 is provided.
minratio	minimal value of ℓ_1-part of the penalty that will be tried, as a fraction of the maximal lambda1 value. A too small value might lead to instability at the end of the solution path corresponding to small lambda1 combined with $\lambda_2 = 0$. The default value tries to avoid this, adapting to the '$n < p$' context. Ignored if lambda1 is provided.
maxfeat	integer; limits the number of features ever to enter the model; i.e., non-zero coefficients for the Elastic-net: the algorithm stops if this number is exceeded and lambda1 is cut at the corresponding level. Default is min(nrow(x),ncol(x)) for small lambda2 (<0.01) and min(4*nrow(x),ncol(x)) otherwise. Use with care, as it considerably changes the computation time.
control	list of argument controlling low level options of the algorithm –use with care and at your own risk– :

- verbose: integer; activate verbose mode –this one is not too risky!– set to 0 for no output; 1 for warnings only, and 2 for tracing the whole progression. Default is 1. Automatically set to 0 when the method is embedded within cross-validation or stability selection.
- timer: logical; use to record the timing of the algorithm. Default is FALSE.
- maxiter the maximal number of iteration used in the active set algorithm to solve the problem for a given value of lambda1 . Default is 50.
- maxadd the maximal number of variables (or groups, for group-sparse models) activated at once in the active set algorithm, chosen among the largest violations of the optimality conditions. Default is 10 (5 for group-sparse models).
- method a string for the underlying solver used. Either "quadra", "fista" or "pgd". Default is "quadra".
- factmat Boolean indicating if matrix factorization should be used to solve the sub-system. If TRUE (the default), a Cholesky decomposition is maintained along the path. If FALSE, the sub-system are solved with a conjugate gradient algorithm.

- `threshold` a threshold for convergence. The algorithm stops when the optimality conditions are fulfill up to this threshold. Default is `1e-6`.
- `monitor` indicates if a monitoring of the convergence should be recorded, by computing a lower bound between the current solution and the optimum: when `'0'` (the default), no monitoring is provided; when `'1'`, the bound derived in Grandvalet et al. is computed; when `'>1'`, the Fenchel duality gap is computed along the algorithm.

Details

The optimized criterion is the following:

$$\hat{\beta}_{\lambda_1,\lambda_1} = \arg\min_{\beta} \frac{1}{2}(y - X\beta)^T(y - X\beta) + \lambda_1\|D\beta\|_\infty + \frac{\lambda_2}{2}\beta^T S\beta,$$

where D is a diagonal matrix, whose diagonal terms are provided as a vector by the `penscale` argument. The ℓ_2 structuring matrix S is provided via the `struct` argument, a positive semidefinite matrix (possibly of class `Matrix`).

Note that the quadratic algorithm for the bounded regression may become unstable along the path because of singularity of the underlying problem, e.g. when there are too much correlation or when the size of the problem is close to or smaller than the sample size. In such cases, it might be a good idea to switch to the proximal solver, slower yet more robust. This is the strategy automatically adopted in code, that will send a warning in verbose mode while switching the method to `'fista'` and keep on optimizing on the remainder of the path.

Singularity of the system can also be avoided with a larger ℓ_2-regularization, via `lambda2`, or a "not-too-small" ℓ_∞ regularization, via a larger `'minratio'` argument.

Value

an object with class QuadrupenFit.

an object with class BoundedRegressionFit, inheriting from QuadrupenFit.

Examples

```
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

## Infinity norm without/with an additional l2 regularization term
## and with structuring prior
labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- "relevant"
```

```
plot(bounded_reg(x,y,lambda2=0) , label=labels) ## a mess
plot(bounded_reg(x,y,lambda2=10), label=labels) ## good guys are at the boundaries
plot(bounded_reg(x,y,lambda2=10,struct=solve(Sigma)), label=labels) ## even better
```

BoundedRegressionFit *Class "BoundedRegression"*

Description

Class of object returned by the fitting function bounded_reg(). Inherits fields and methods of QuadrupenFit.

Super class

QuadrupenFit -> BoundedRegressionFit

Active bindings

penalty character describing the regularizer/penalty

lambdainf vector of tuning parameters for the linf penalty

lambda2 vector of tuning parameters for the l2 penalty

Methods

Public methods:

- BoundedRegressionFit$new()
- BoundedRegressionFit$clone()

BoundedRegressionFit$new(): Initialize a BoundedRegressionFit model

Usage:

```
BoundedRegressionFit$new(data, intercept, regParam)
```

Arguments:

data a DataModel object

intercept a logical; should an intercept be included in the mode?

regParam a list with two elements, a vector and a scalar, for the regularization

BoundedRegressionFit$clone(): The objects of this class are cloneable with this method.

Usage:

```
BoundedRegressionFit$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

QuadrupenFit, bounded_reg()

coef.QuadrupenFit *Extract model coefficients*

Description

Extracts model coefficients from a QuadrupenFit object

Usage

```
## S3 method for class 'QuadrupenFit'
coef(object, selection = NULL, ...)
```

Arguments

object	a QuadrupenFit object
selection	either a character (model selection criteria) of a scalar (lambda value)
...	not used, only here for S3 compatibility

Value

a vector of coefficients

criteria *Penalized criteria based on estimation of degrees of freedom*

Description

Produce a plot or send back the values of some penalized criteria accompanied with the vector(s) of parameters selected accordingly. The default behavior plots the BIC and the AIC (with respective factor $\log(n)$ and 2) yet the user can specify any penalty.

Usage

```
criteria(
  object,
 penalty = setNames(c(2, log(object$nobs), log(object$nvar), log(object$nobs) + 2 *
    log(object$nvar)), c("AIC", "BIC", "mBIC", "eBIC")),
  sigma = NULL
)

## S3 method for class 'QuadrupenFit'
criteria(
  object,
 penalty = setNames(c(2, log(object$nobs), log(object$nvar), log(object$nobs) + 2 *
    log(object$nvar)), c("AIC", "BIC", "mBIC", "eBIC")),
  sigma = NULL
)
```

Arguments

`object`	output of a fitting procedure of the **quadrupen** package (e.g. `elastic_net()`).
`penalty`	a vector with as many penalties a desired. The default contains the penalty corresponding to the AIC and the BIC (2 and $\log(n)$). Setting the "names" attribute, as done in the default definition, leads to outputs which are easier to read.
`sigma`	scalar: an estimate of the residual variance. When available, it is plugged-in the criteria, which may be more relevant. If `NULL` (the default), it is estimated as usual (see details).

Value

an object with class InformationCriteria is sent back and stored as a field of the original QuadrupenFit object.

Methods (by class)

- `criteria(QuadrupenFit)`: S3 method for information criteria of a QuadrupenFit

Note

When `sigma` is provided, the criterion takes the form

$$\left\|\mathbf{y} - \mathbf{X}\hat{\beta}\right\|^2 + \text{penalty} \times \frac{\hat{\text{df}}}{n}\,\sigma^2.$$

When it is unknown, it writes

$$\log\left(\left\|\mathbf{y} - \mathbf{X}\hat{\beta}\right\|^2\right) + \text{penalty} \times \hat{\text{df}}.$$

Estimation of the degrees of freedom (for the elastic-net, the LASSO and also bounded regression) are computed by applying and adapting the results of Tibshirani and Taylor (see references below).

References

Ryan Tibshirani and Jonathan Taylor. Degrees of freedom in lasso problems, Annals of Statistics, 40(2) 2012.

Examples

```
## Not run:
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- bdiag(Soo,Sww,Soo,Sww,Soo)
```

```
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

## Plot penalized criteria for the Elastic-net path
criteria(elastic_net(x,y, lambda2=1))

#' Plot penalized criteria for the Bounded regression
criteria(bounded_reg(x,y, lambda2=1))

## End(Not run)
```

`cross_validate` *Cross-validation for Quadrupen object*

Description

Function that computes K-fold cross-validated error of a quadrupen fit, possibly on a grid of `lambda1`, `lambda2`.

Usage

```
cross_validate(
  object,
  K = 10,
  folds = split(sample(1:object$nobs), rep(1:K, length = object$nobs)),
  lambda2 = object$minor_tuning,
  verbose = TRUE,
  cores = 1
)

## S3 method for class 'QuadrupenFit'
cross_validate(
  object,
  K = 10,
  folds = split(sample(1:object$nobs), rep(1:K, length = object$nobs)),
  lambda2 = object$minor_tuning,
  verbose = TRUE,
  cores = parallel::detectCores() - 2
)
```

Arguments

`object`	an R6 object with class QuadrupenFit
`K`	integer indicating the number of folds. Default is 10.

folds	list of K vectors that describes the folds to use for the cross-validation. By default, the folds are randomly sampled with the specified K. The same folds are used for each values of lambda2.
lambda2	tunes the ℓ_2-penalty (ridge-like) of the fit. If none is provided, a vector of values is generated and a CV is performed on a grid of lambda2 and lambda1, using the same folds for each lambda2.
verbose	logical; indicates if the progression (the current lambda2 should be displayed. Default is TRUE.
cores	the number of cores to use. The default uses 1 core (safer in case your BLAS/LAPACK libraries are multithreaded)

Value

an object with class CrossValidation is sent back and stored as a field of the original QuadrupenFit object.

Methods (by class)

- cross_validate(QuadrupenFit): S3 method for cross-validation of a QuadrupenFit

Note

If the user runs the fitting method with option 'bulletproof' set to FALSE, the algorithm may stop at an early stage of the path. Early stops are handled internally, in order to provide results on the same grid of penalty tuned by λ_1. This is done by means of NA values, so as mean and standard error are consistently evaluated. If, while cross-validating, the procedure experiences too much early stops, a warning is sent to the user, in which case you should reconsider the grid of lambda1 used for the cross-validation. If bulletproof is TRUE (the default), there is nothing to worry about, except a possible slow down when any switching to the proximal algorithm is required.

Examples

```
## Not run:
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
cor  <- 0.75
Soo  <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variable
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- bdiag(Soo,Sww,Soo,Sww,Soo) + 0.1
diag(Sigma) <- 1
n <- 100
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

enet <- elastic_net(x, y, nlambda1=50)

## Use fewer lambda1 values by overwritting the default parameters
## and cross-validate over the sequences lambda1 and lambda2
cv.grid <- cross_validate(enet, lambda2=10^seq(2,-2,len=50))
```

```
## Rerun simple cross-validation with the appropriate lambda2
cv.10K <- crossval(x,y, lambda2=cv.grid$lambda2_min)
## Try leave one out also
cv.loo <- crossval(x,y, K=n, lambda2=cv.grid$lambda2_min)

plot(cv.grid)
plot(cv.10K)
plot(cv.loo)

## Performance for selection purpose
cat("\nFalse positives with the minimal 10-CV choice: ", sum(sign(beta) != sign(cv.10K$beta_min )))
cat("\nFalse positives with the minimal LOO-CV choice: ", sum(sign(beta) != sign(cv.loo$beta_min)))

## End(Not run)
```

`CrossValidation` *Class CrossValidation*

Description

Class of object returned by the `QuadrupenFit$cross_validate()` method or the `cross_validate()` function. Owns `print()` and `plot()` methods.

Active bindings

`data` a data frame containing the mean cross-validated error and its associated standard error for each values of `lambda1` and `lambda2`.

`folds` list of `K` vectors indicating the folds used for cross-validation.

`lambda1` vector of λ_1 (ℓ_1 or ℓ_∞ penalty levels) for which each cross-validation has been performed.

`lambda2` vector (or scalar) of ℓ_2-penalty levels for which each cross-validation has been performed.

`lambda1_min` level of λ_1 that minimizes the error estimated by cross-validation.

`lambda2_min` level of λ_2 that minimizes the error estimated by cross-validation.

`lambda1_1se` largest level of λ_1 such as the cross-validated error is within 1 standard error of the minimum.

`lambda2_1se` largest level of λ_2 such that the cross-validated error is within 1 standard error of the minimum (only relevant for ridge regression).

Methods

Public methods:

- `CrossValidation$new()`
- `CrossValidation$show()`
- `CrossValidation$print()`
- `CrossValidation$plotCV_1D()`

- `CrossValidation$plotCV_2D()`
- `CrossValidation$plot()`
- `CrossValidation$clone()`

`CrossValidation$new()`: Constructor for a CrossValidation object Should be called internally by an object `QuadrupenFit$cross_validate()`

Usage:

```
CrossValidation$new(cv_error, folds)
```

Arguments:

`cv_error` data frame storing output of a cv job

`folds` list of K folds used for cross-validation

`CrossValidation$show()`: User friendly print method

Usage:

```
CrossValidation$show()
```

`CrossValidation$print()`: User friendly print method

Usage:

```
CrossValidation$print()
```

`CrossValidation$plotCV_1D()`: Plot 1-dimensional cross-validation

Usage:

```
CrossValidation$plotCV_1D(
  log_scale = TRUE,
  title = "Cross-validation error",
  se = TRUE
)
```

Arguments:

`log_scale` logical, should a log-scale be used for the x-axis

`title` graph title

`se` logical, should confidence band be displayed (TRUE by default)

Returns: a ggplot object

`CrossValidation$plotCV_2D()`: Plot 2-dimensional cross-validation output (grid lambda1 x lambda2)

Usage:

```
CrossValidation$plotCV_2D(title = "Cross-validation error")
```

Arguments:

`title` graph title

Returns: a **ggplot2** object

`CrossValidation$plot()`: Plot cross-validation job by choosing the most appropriate output (1D- or 2D)

Usage:

```
CrossValidation$plot(log_scale = TRUE, title = "Cross-validation error")
```

Arguments:

`log_scale` logical, should a log-scale be used for the x-axis

`title` graph title

Returns: a ggplot object

`CrossValidation$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CrossValidation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`DataModel` *Data Class*

Description

Class for storing data and various fixed quantity

Public fields

`X` matrix of regressor

`y` vector of response

`C_inv` Inverse of the Cholesky decomposition of S

`S` SDP structuring matrix

`wy` vector of observation weights

Active bindings

`d` number of regressor

`n` sample size

`sparse_encoding` logical indicating if the matrix of regressor is sparsely encoded

`varnames` character, the names of the covariates/regressors

`normx` norm of each column of X

Methods

Public methods:

- DataModel$new()
- DataModel$CholStruct()
- DataModel$splitTrainTest()
- DataModel$splitFold()
- DataModel$splitSubSamples()
- DataModel$clone()

`DataModel$new()`: constructor for DataModel

Usage:

```
DataModel$new(
  covariates,
  outcome,
  cov_struct,
  obs_weights = rep(1, length(outcome)),
  check_args = TRUE
)
```

Arguments:

`covariates` matrix of covariates/regressors

`outcome` vector of outcome/response

`cov_struct` sdp matrix structuring the covariates/regressors

`obs_weights` vector of observations weights

`check_args` logical, should args be check at initialization?

`cov_weights` vector of covariates/regressors weights

`DataModel$CholStruct()`: Compute Cholesky factorization of the Structuring matrix

Usage:

```
DataModel$CholStruct()
```

`DataModel$splitTrainTest()`: a function splitting the data into train and test folds

Usage:

```
DataModel$splitTrainTest(
  nfolds = 10,
  folds = split(sample(1:self$n), rep(1:nfolds, length = self$n))
)
```

Arguments:

`nfolds` the number of folds

`folds` a list of vectors describing the folds (optional)

Returns: a list with train and test data and id.

`DataModel$splitFold()`: a function splitting the data into one train and one test set

Usage:

`DataModel$splitFold(omit)`

Arguments:

`omit` vector of the indices of the test observations

Returns: a list with train and test data and id.

`DataModel$splitSubSamples():` a function splitting data into subsamples

Usage:

```
DataModel$splitSubSamples(
  n_subsamples = 50,
  subsample_size = floor(self$n/2),
 subsamples = replicate(n_subsamples, sample(1:self$n, subsample_size), simplify =
    FALSE),
  weakness = 1
)
```

Arguments:

`n_subsamples` the number of subsamples

`subsample_size` the subsample size

`subsamples` list with vector of subsamples (optional)

`weakness` coefficient for randomly weighting the regressor, default to 1

Returns: a list of DataModel, resampling of the original

`DataModel$clone():` The objects of this class are cloneable with this method.

Usage:

```
DataModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`deviance.QuadrupenFit` *Extract model deviance*

Description

Extracts the deviance of a QuadrupenFit object

Usage

```
## S3 method for class 'QuadrupenFit'
deviance(object, ...)
```

Arguments

`object`	a QuadrupenFit object
`...`	not used, only here for S3 compatibility

Value

a scalar

fitted.QuadrupenFit *Extracts model fitted values*

Description

Extracts model fitted values

Usage

```
## S3 method for class 'QuadrupenFit'
fitted(object, ...)
```

Arguments

object	a QuadrupenFit object
...	not used, only here for S3 compatibility

Value

A matrix of fitted values extracted from object.

fused_lasso *A function for fitting generalized fused-Lasso problems*

Description

This function fits the standard version of the fused lasso. It can take a general matrix x and allows for possible weights on the lambda1 and lambda2 penalties.

Usage

```
fused_lasso(
  x,
  y,
  lambda1 = NULL,
  lambda2 = 1,
  pen_fused = c("L1", "L2", "Huber"),
  penscale = rep(1, ncol(x)),
  struct = NULL,
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 50, length(lambda1)),
  minratio = 0.01,
 maxfeat = ifelse(lambda2 < 1, min(nrow(x), ncol(x)), min(2 * nrow(x), ncol(x))),
  beta0 = rep(0, ncol(x)),
  control = list()
)
```

Arguments

x	matrix of features, possibly sparsely encoded (experimental). Do NOT include intercept. When normalized is TRUE, coefficients will then be rescaled to the original scale.
y	response vector.
lambda1	sequence of decreasing ℓ_1-penalty levels. If NULL (the default), a vector is generated with nlambda1 entries, starting from a guessed level lambda1.max where only the intercept is included, then shrunken to minratio*lambda1.max.
lambda2	real scalar; tunes the ℓ_2 penalty in the Elastic-net. Default is 0.01. Set to 0 to recover the Lasso.
pen_fused	penalty used for fusing the variables (either L1, L2 or Huber). Default is L1
penscale	vector with real positive values that weight the penalty of each feature. Default sets all weights to 1.
struct	Description of the graph that corresponds to the lambda2 penalty structure. If NULL (the default) a chain graph is assumed, like in the standard fused-lasso. If a matrix is given, interpreted as a symmetric adjacency matrix
intercept	logical; indicates if an intercept should be included in the model. Default is TRUE.
normalize	logical; indicates if variables should be normalized to have unit L2 norm before fitting. Default is TRUE.
nlambda1	integer that indicates the number of values to put in the lambda1 vector. Ignored if lambda1 is provided.
minratio	minimal value of ℓ_1-part of the penalty that will be tried, as a fraction of the maximal lambda1 value. A too small value might lead to instability at the end of the solution path corresponding to small lambda1 combined with $\lambda_2 = 0$. The default value tries to avoid this, adapting to the '$n < p$' context. Ignored if lambda1 is provided.
maxfeat	integer; limits the number of features ever to enter the model; i.e., non-zero coefficients for the Elastic-net: the algorithm stops if this number is exceeded and lambda1 is cut at the corresponding level. Default is min(nrow(x),ncol(x)) for small lambda2 (<0.01) and min(4*nrow(x),ncol(x)) otherwise. Use with care, as it considerably changes the computation time.
beta0	a starting point for the vector of parameter. By default, will initialized zero. May save time in some situation.
control	list of argument controlling low level options of the algorithm: • verbose: logical; verbose mode • timer: logical; use to record the timing of the algorithm. Default is FALSE. • maxiterin Maximum number of iterations in the inner loop to run. • maxiterout Maximum number of iterations in the outer loop to run. • maxactivation Maximum number of previously inactive variables to activate at the same time • accuracy Accuracy at which the algorithm will stop. • fusioncheck Should the fused sets be checked for separation? • verbose Should the function give some output what it is doing?

Details

The optimized criterion is the following:

$$\hat{\beta}_{\lambda_1,\lambda_2} = \arg\min_{\beta} \frac{1}{2}(y - X\beta)^T(y - X\beta) + \lambda_1\|w \circ \beta\|_1 + \frac{\lambda_2}{2}\sum_{i\sim j} w_{ij}|\beta_i - \beta_j|,$$

where D is a diagonal matrix, whose diagonal terms are provided as a vector by the `penscale` argument. The ℓ_1 fusion penalty is structured by a possibly weighted graph G provided via the `struct` argument, as a symmetric (undirected) adjacency matrix.

Value

an object with class FusedLassoFit, inheriting from QuadrupenFit.

Author(s)

Original code by Holger Hoefling, refactoring by Julien Chiquet

Examples

```
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

res <- fused_lasso(x, y, lambda2=5)
G <- igraph::make_ring(ncol(x)) |> igraph::as_adjacency_matrix(sparse = FALSE)
resG <- fused_lasso(x, y, lambda2=5, struct = G)
plot(res)
plot(resG)
```

FusedLassoFit *Class "FusedLassoFit"*

Description

Class of object returned by the fitting function `fused_lasso()`. Inherits fields and methods of QuadrupenFit.

Super class

`QuadrupenFit` -> `FusedLassoFit`

Active bindings

`penalty` character describing the regularizer/penalty

`lambda1` vector of tuning parameters for the l1 penalty

`lambda2` vector of tuning parameters for the fusion penalty

Methods

Public methods:

- `FusedLassoFit$new()`
- `FusedLassoFit$clone()`

`FusedLassoFit$new()`: Initialize a `FusedLassoFit` model

Usage:

```
FusedLassoFit$new(data, intercept, regParam)
```

Arguments:

`data` a `DataModel` object

`intercept` a logical; should an intercept be included in the mode?

`regParam` a list with two elements, a vector and a scalar, for the regularization

`FusedLassoFit$clone()`: The objects of this class are cloneable with this method.

Usage:

```
FusedLassoFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

QuadrupenFit, `bounded_reg()`

`group_lava` *Fit a linear model with group-lava regularization*

Description

Adjust a the group-lava regularized linear models, that is a lava transformation of the data plus a mixture of either a (possibly weighted) ℓ_1/ℓ_2- or ℓ_1/ℓ_∞-norm, and a (possibly structured) ℓ_2-norm (ridge-like). The solution path is computed at a grid of values for the ℓ_1/ℓ_q-penalty. See details for the criterion optimized.

Usage

```
group_lava(
  x,
  y,
  group,
  type = c("l2", "coop", "linf"),
  lambda1 = NULL,
  lambda2 = 1,
  weights = rep(1, nrow(x)),
  penscale = sqrt(tabulate(group)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
 maxfeat = ifelse(lambda2 < 0.01, min(nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
  beta0 = numeric(ncol(x)),
  control = list()
)
```

Arguments

x	matrix of features, possibly sparsely encoded (experimental). Do NOT include intercept. When normalized is TRUE, coefficients will then be rescaled to the original scale.
y	response vector.
group	vector of integers indicating group belonging. Must match the number of column in x. Must be SORTED integers starting from 1.
type	string indicating whether the ℓ_1/ℓ_2 or the ℓ_1/ℓ_∞ group-Lasso must be fitted. Could be "linf" or "l2", default is "l2"
lambda1	sequence of decreasing ℓ_1-penalty levels. If NULL (the default), a vector is generated with nlambda1 entries, starting from a guessed level lambda1.max where only the intercept is included, then shrunken to minratio*lambda1.max.
lambda2	real scalar; tunes the ℓ_2 penalty in the Elastic-net. Default is 0.01. Set to 0 to recover the Lasso.
weights	vector with real positive values that weight the observations (like in weighted least square). Default sets all weights to 1.
penscale	vector with real positive values that weight the penalty of each feature. Default sets all weights to 1.
struct	matrix structuring the coefficients, possibly sparsely encoded. Must be at least positive semidefinite (this is checked internally). If NULL (the default), the identity matrix is used. See details below.
intercept	logical; indicates if an intercept should be included in the model. Default is TRUE.

`normalize` logical; indicates if variables should be normalized to have unit L2 norm before fitting. Default is `TRUE`.

`refit` logical: indicates if the non null coefficients should be refit to avoid excessive bias. Default is FALSE. Can be changed later (both raw and refit coefficients are stored).

`nlambda1` integer that indicates the number of values to put in the `lambda1` vector. Ignored if `lambda1` is provided.

`minratio` minimal value of ℓ_1-part of the penalty that will be tried, as a fraction of the maximal `lambda1` value. A too small value might lead to instability at the end of the solution path corresponding to small `lambda1` combined with $\lambda_2 = 0$. The default value tries to avoid this, adapting to the '$n < p$' context. Ignored if `lambda1` is provided.

`maxfeat` integer; limits the number of features ever to enter the model; i.e., non-zero coefficients for the Elastic-net: the algorithm stops if this number is exceeded and `lambda1` is cut at the corresponding level. Default is `min(nrow(x),ncol(x))` for small `lambda2` (<0.01) and `min(4*nrow(x),ncol(x))` otherwise. Use with care, as it considerably changes the computation time.

`beta0` a starting point for the vector of parameter. By default, will initialized zero. May save time in some situation.

`control` list of argument controlling low level options of the algorithm –use with care and at your own risk– :

- `verbose`: integer; activate verbose mode –this one is not too risky!– set to `0` for no output; `1` for warnings only, and `2` for tracing the whole progression. Default is `1`. Automatically set to `0` when the method is embedded within cross-validation or stability selection.
- `timer`: logical; use to record the timing of the algorithm. Default is `FALSE`.
- `maxiter` the maximal number of iteration used in the active set algorithm to solve the problem for a given value of lambda1 . Default is 50.
- `maxadd` the maximal number of variables (or groups, for group-sparse models) activated at once in the active set algorithm, chosen among the largest violations of the optimality conditions. Default is 10 (5 for group-sparse models).
- `method` a string for the underlying solver used. Either `"quadra"`, `"fista"` or `"pgd"`. Default is `"quadra"`.
- `factmat` Boolean indicating if matrix factorization should be used to solve the sub-system. If `TRUE` (the default), a Cholesky decomposition is maintained along the path. If `FALSE`, the sub-system are solved with a conjugate gradient algorithm.
- `threshold` a threshold for convergence. The algorithm stops when the optimality conditions are fulfill up to this threshold. Default is `1e-6`.
- `monitor` indicates if a monitoring of the convergence should be recorded, by computing a lower bound between the current solution and the optimum: when `'0'` (the default), no monitoring is provided; when `'1'`, the bound derived in Grandvalet et al. is computed; when `'>1'`, the Fenchel duality gap is computed along the algorithm.

Details

The optimized criterion is the following:

$$\hat{\theta}_{\lambda_1,\lambda_2} = \arg\min_{\theta=\beta+\delta} \frac{1}{2n}(y - X(\beta+\delta))^T(y - X(\beta+\delta)) + \lambda_1\Omega_g(\beta) + \frac{\lambda_2}{2}\delta^T S\delta,$$

where Ω_g is the group-wise mixed norm: ℓ_1/ℓ_2 (Group-LAVA) or ℓ_1/ℓ_∞, controlled by the `type` argument. The ℓ_2 structuring matrix S is provided via `struct`.

Value

an object with class GroupLavaFit, inheriting from QuadrupenFit.

References

Chernozhukov, Victor, Christian Hansen, and Yuan Liao. "A lava attack on the recovery of sums of dense and sparse signals." The Annals of Statistics (2017): 39-76. doi:10.1214/16-AOS1434

Examples

```
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta  <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
delta <- runif(sum(c(25,10,25,10,25)),-.1,.1)
grp  <- rep(1:5, c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- "relevant"

## Not run:
## Standard Group-Lasso
plot(group_lava(x,y,grp), label=labels)
plot(group_lava(x,y,grp, lambda2=.5), label=labels)
plot(group_lava(x,y,grp, lambda2=10), label=labels)
plot(group_lava(x,y,grp, lambda2=10,struct=solve(Sigma)), label=labels)

## L1/LINF Group-Lasso
plot(group_lava(x, y, grp, type = "linf"), label=labels)
plot(group_lava(x, y, grp, type = "linf", lambda2=.5), label=labels)
plot(group_lava(x, y, grp, type = "linf", lambda2=10), label=labels)
plot(group_lava(x, y, grp, type = "linf", lambda2=10,struct=solve(Sigma)), label=labels)

## Cooperative-Lasso
plot(group_lava(x, y, grp, type = "coop"), label=labels)
```

```
plot(group_lava(x, y, grp, type = "coop", lambda2=.5), label=labels)
plot(group_lava(x, y, grp, type = "coop", lambda2=10), label=labels)
plot(group_lava(x, y, grp, type = "coop", lambda2=10,struct=solve(Sigma)), label=labels)

## End(Not run)
```

`group_sparse_lm` *Fit a linear model with (sparse) group regularisation*

Description

Adjust a linear model with (sparse) group regularization, that is, a mixture of an element-wise ℓ_1 norm and a group-wise mixed-norm (either ℓ_1/ℓ_2, ℓ_1/ℓ_∞ or cooperative). We also add a (possibly structured) ℓ_2-norm (ridge-like). The solution path is computed on an automatically tuned grid of values for the sparse group penalty. The mixture coefficient and the amount of ridge-like regularization are fixed by the user. See details for the criterion optimized.

Usage

```
group_sparse_lm(
  x,
  y,
  group,
  type = c("l2", "coop", "linf"),
  lambda1 = NULL,
  lambda2 = 0.01,
  alpha = 0,
  weights = rep(1, nrow(x)),
  penscale = sqrt(tabulate(group)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
 maxfeat = ifelse(lambda2 < 0.01, min(2 * nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
  beta0 = numeric(ncol(x)),
  control = list()
)

group_lasso(
  x,
  y,
  group,
  lambda1 = NULL,
  lambda2 = 0,
```

```
  weights = rep(1, nrow(x)),
  penscale = sqrt(tabulate(group)),
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
  maxfeat = ncol(x),
  beta0 = numeric(ncol(x)),
  control = list(method = "quadra")
)

group_l1linf(
  x,
  y,
  group,
  lambda1 = NULL,
  lambda2 = 0,
  weights = rep(1, nrow(x)),
  penscale = sqrt(tabulate(group)),
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
  maxfeat = ncol(x),
  beta0 = numeric(ncol(x)),
  control = list()
)

coop_lasso(
  x,
  y,
  group,
  lambda1 = NULL,
  lambda2 = 0,
  weights = rep(1, nrow(x)),
  penscale = sqrt(tabulate(group)),
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
  maxfeat = ncol(x),
  beta0 = numeric(ncol(x)),
  control = list()
)

sparse_group_lasso(
  x,
  y,
```

```
  group,
  lambda1 = NULL,
  lambda2 = 0,
  alpha = 0.5,
  weights = rep(1, nrow(x)),
  penscale = sqrt(tabulate(group)),
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
  maxfeat = ncol(x),
  beta0 = numeric(ncol(x)),
  control = list()
)

sparse_group_l1linf(
  x,
  y,
  group,
  lambda1 = NULL,
  lambda2 = 0,
  alpha = 0.5,
  weights = rep(1, nrow(x)),
  penscale = sqrt(tabulate(group)),
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
  maxfeat = ncol(x),
  beta0 = numeric(ncol(x)),
  control = list()
)

sparse_coop_lasso(
  x,
  y,
  group,
  lambda1 = NULL,
  lambda2 = 0,
  alpha = 0.5,
  weights = rep(1, nrow(x)),
  penscale = sqrt(tabulate(group)),
  intercept = TRUE,
  normalize = TRUE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
  maxfeat = ncol(x),
  beta0 = numeric(ncol(x)),
```

```
  control = list()
)
```

Arguments

x	matrix of features, possibly sparsely encoded (experimental). Do NOT include intercept. When normalized is TRUE, coefficients will then be rescaled to the original scale.
y	response vector.
group	vector of integers indicating group belonging. Must match the number of column in x. Must be SORTED integers starting from 1.
type	string indicating the sparse-group variant to be fitted. Could be "l2", "coop", or "linf". Default is "l2" (regular Group-Lasso)
lambda1	sequence of decreasing ℓ_1-penalty levels. If NULL (the default), a vector is generated with nlambda1 entries, starting from a guessed level lambda1.max where only the intercept is included, then shrunken to minratio*lambda1.max.
lambda2	real scalar; tunes the ℓ_2 penalty in the Elastic-net. Default is 0.01. Set to 0 to recover the Lasso.
alpha	real scalar in (0,1); tunes mixture between ℓ_1 group penalties. Default is 0.0 (standard group-lasso).
weights	vector with real positive values that weight the observations (like in weighted least square). Default sets all weights to 1.
penscale	vector with real positive values that weight the penalty of each feature. Default sets all weights to 1.
struct	matrix structuring the coefficients, possibly sparsely encoded. Must be at least positive semidefinite (this is checked internally). If NULL (the default), the identity matrix is used. See details below.
intercept	logical; indicates if an intercept should be included in the model. Default is TRUE.
normalize	logical; indicates if variables should be normalized to have unit L2 norm before fitting. Default is TRUE.
refit	logical: indicates if the non null coefficients should be refit to avoid excessive bias. Default is FALSE. Can be changed later (both raw and refit coefficients are stored).
nlambda1	integer that indicates the number of values to put in the lambda1 vector. Ignored if lambda1 is provided.
minratio	minimal value of ℓ_1-part of the penalty that will be tried, as a fraction of the maximal lambda1 value. A too small value might lead to instability at the end of the solution path corresponding to small lambda1 combined with $\lambda_2 = 0$. The default value tries to avoid this, adapting to the '$n < p$' context. Ignored if lambda1 is provided.
maxfeat	integer; limits the number of features ever to enter the model; i.e., non-zero coefficients for the Elastic-net: the algorithm stops if this number is exceeded and lambda1 is cut at the corresponding level. Default is min(nrow(x),ncol(x)) for small lambda2 (<0.01) and min(4*nrow(x),ncol(x)) otherwise. Use with care, as it considerably changes the computation time.

`beta0` a starting point for the vector of parameter. By default, will initialized zero. May save time in some situation.

`control` list of argument controlling low level options of the algorithm –use with care and at your own risk– :

- `verbose`: integer; activate verbose mode –this one is not too risky!– set to `0` for no output; `1` for warnings only, and `2` for tracing the whole progression. Default is `1`. Automatically set to `0` when the method is embedded within cross-validation or stability selection.
- `timer`: logical; use to record the timing of the algorithm. Default is `FALSE`.
- `maxiter` the maximal number of iteration used in the active set algorithm to solve the problem for a given value of lambda1 . Default is 50.
- `maxadd` the maximal number of variables (or groups, for group-sparse models) activated at once in the active set algorithm, chosen among the largest violations of the optimality conditions. Default is 10 (5 for group-sparse models).
- `method` a string for the underlying solver used. Either `"quadra"`, `"fista"` or `"pgd"`. Default is `"quadra"`.
- `factmat` Boolean indicating if matrix factorization should be used to solve the sub-system. If `TRUE` (the default), a Cholesky decomposition is maintained along the path. If `FALSE`, the sub-system are solved with a conjugate gradient algorithm.
- `threshold` a threshold for convergence. The algorithm stops when the optimality conditions are fulfill up to this threshold. Default is `1e-6`.
- `monitor` indicates if a monitoring of the convergence should be recorded, by computing a lower bound between the current solution and the optimum: when `'0'` (the default), no monitoring is provided; when `'1'`, the bound derived in Grandvalet et al. is computed; when `'>1'`, the Fenchel duality gap is computed along the algorithm.

Details

The optimized criterion is the following:

$$\hat{\beta}_{\lambda_1,\lambda_2} = \arg\min_{\beta} \frac{1}{2}(y - X\beta)^T(y - X\beta) + \lambda_1 \left[(1-\alpha)\,\Omega_g(\beta) + \alpha\|D\beta\|_1\right] + \frac{\lambda_2}{2}\beta^T S\beta,$$

where Ω_g is the group-wise mixed norm: ℓ_1/ℓ_2 (Group-Lasso), ℓ_1/ℓ_∞, or cooperative (Clime), controlled by the `type` argument; D is a diagonal matrix whose diagonal terms are given by `penscale`; α tunes the mixture between the group and element-wise penalties; and S is the ℓ_2 structuring matrix provided via `struct`, a positive semidefinite matrix (possibly of class `Matrix`).

Value

an object with class SparseGroupFit, inheriting from QuadrupenFit.

See Also

See also QuadrupenFit

Examples

```
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
grp  <- rep(1:5, c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

## Various sparse group linear models without/with an additional l2 regularization term
## and with structuring prior
labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- "relevant"

## Group-Lasso
plot(group_lasso(x, y, grp), label=labels)

## Sparse Group-Lasso
plot(sparse_group_lasso(x, y, grp, alpha = 0.75), label=labels)

## Not run:

## Sparse Group-Lasso + L2 regularization
plot(group_sparse_lm(x, y, grp, type = "l2", alpha = .75, lambda2=.5),
 label=labels)
plot(group_sparse_lm(x, y, grp, type = "l2", alpha = .75, lambda2=10),
 label=labels)
plot(group_sparse_lm(x, y, grp, type = "l2", alpha = .75, lambda2=10,
 struct=solve(Sigma)), label=labels)

## Group-Lasso L1/LINF
plot(group_l1linf(x, y, grp), label=labels)

## Sparse Group-Lasso L1/LINF
plot(sparse_group_l1linf(x, y, grp, alpha = 0.75), label=labels)

## Sparse L1/LINF Group-Lasso + L2 regularization
plot(group_sparse_lm(x, y, grp, type = "linf", alpha = .75, lambda2=.5),
  label=labels)
plot(group_sparse_lm(x, y, grp, type = "linf", alpha = .75, lambda2=10),
  label=labels)
plot(group_sparse_lm(x, y, grp, type = "linf", alpha = .75, lambda2=10,
  struct=solve(Sigma)), label=labels)

## Cooperative-Lasso
plot(coop_lasso(x, y, grp), label=labels)
```

```
## Sparse Cooperative-Lasso
plot(sparse_coop_lasso(x, y, grp, alpha = 0.75), label=labels)

## Sparse Cooperative-Lasso + L2 regularization
plot(group_sparse_lm(x, y, grp, type = "coop", alpha = .75, lambda2=.5),
 label=labels)
plot(group_sparse_lm(x, y, grp, type = "coop", alpha = .75, lambda2=10),
 label=labels)
plot(group_sparse_lm(x, y, grp, type = "coop", alpha = .75, lambda2=10,
 struct=solve(Sigma)), label=labels)

## End(Not run)
```

GroupLavaFit *Class "GroupLavaFit"*

Description

Class of object returned by the fitting function `group_lava()`. Inherits fields and methods of QuadrupenFit and LavaFit

Super classes

`QuadrupenFit` -> `LavaFit` -> `GroupLavaFit`

Active bindings

`lambda1` vector of tuning parameters for the l1 group penalty (sparse component)

`lambda2` vector of tuning parameters for the l2 penalty (dense component)

`penalty` character describing the regularizer/penalty

`group` vector of integers indicating group belonging

`type` string indicating whether the ℓ_1/ℓ_2 or the ℓ_1/ℓ_∞ group-Lasso must be fitted.

Methods

Public methods:

- `GroupLavaFit$new()`
- `GroupLavaFit$clone()`

`GroupLavaFit$new()`: Initialize a `GroupLavaFit` model

Usage:

```
GroupLavaFit$new(data, intercept, group, type, regParam)
```

Arguments:

`data` a `DataModel` object

`intercept` a logical; should an intercept be included in the mode?

`group` vector of integers indicating group belonging.

`type` string indicating whether the ℓ_1/ℓ_2 or the ℓ_1/ℓ_∞ group-Lasso must be fitted.

`regParam` a list with two elements, a vector and a scalar, for the regularization

`GroupLavaFit$clone()`: The objects of this class are cloneable with this method.

Usage:

```
GroupLavaFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

QuadrupenFit, `group_lava()`

`InformationCriteria` *Class InformationCriteria*

Description

Class of object returned by the `QuadrupenFit$criteria()` method or the `criteria()` function. Owns `print()` and `plot()` methods.

Active bindings

`data` a data frame containing the values of various information criteria (AIC, BIC, lmBIC, eBIC, GCV) along the value of `lambda1`.

`lambda` vector of λ_1 (ℓ_1 or ℓ_∞ penalty levels) for which each cross-validation has been performed.

`names` a vector of characters storing the names of the precomputed criteria

Methods

Public methods:

- `InformationCriteria$new()`
- `InformationCriteria$show()`
- `InformationCriteria$print()`
- `InformationCriteria$plot()`
- `InformationCriteria$clone()`

`InformationCriteria$new()`: Constructor for a InformationCriteria object Should be called internally by an object `QuadrupenFit$criteria()`

Usage:

```
InformationCriteria$new(value)
```

Arguments:

`value` data frame storing output of `QuadrupenFit$criteria()`

`InformationCriteria$show()`: User friendly print method

Usage:

```
InformationCriteria$show()
```

`InformationCriteria$print()`: User friendly print method

Usage:

```
InformationCriteria$print()
```

`InformationCriteria$plot()`: Plot the the desired criteria

Usage:

```
InformationCriteria$plot(
  criteria = self$names,
  log_scale = TRUE,
  xvar = c("lambda", "fraction", "df"),
  title = "Information Criteria"
)
```

Arguments:

`criteria` a vector of character with the criteria to plot. The default plot all the criteria available (stored in the field `names`)

`log_scale` logical; indicates if a log-scale should be used when `xvar="lambda"`. Default is `TRUE`.

`xvar` variable to plot on the X-axis: either `"df"` (the estimated degrees of freedom), `"lambda"` (λ_1 penalty level) or `"fraction"` (ℓ_1-norm of the coefficients). Default is set to `"lambda"`.

`title` graph title

Returns: a **ggplot2** object

`InformationCriteria$clone()`: The objects of this class are cloneable with this method.

Usage:

```
InformationCriteria$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`isQuadrupenFit` *Auxiliary functions to check the given class of an object*

Description

Auxiliary functions to check the given class of an object

Usage

```
isQuadrupenFit(Robject)
```

Arguments

Robject	an R object to evaluate

Value

logical

---

lava *Fit a linear model with lava regularization*

Description

Adjust a lava regularized linear model, that is a lava transformation of the data followed by a (possibly weighted) ℓ_1-norm. The solution path is computed at a grid of values for the ℓ_1-penalty. See details for the criterion optimized.

Usage

```
lava(
  x,
  y,
  lambda1 = NULL,
  lambda2 = 1,
  weights = rep(1, nrow(x)),
  penscale = rep(1, ncol(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = 0.01,
  maxfeat = min(nrow(x), ncol(x)),
  beta0 = numeric(ncol(x)),
  control = list()
)
```

Arguments

x	matrix of features, possibly sparsely encoded (experimental). Do NOT include intercept. When normalized is TRUE, coefficients will then be rescaled to the original scale.
y	response vector.
lambda1	sequence of decreasing ℓ_1-penalty levels. If NULL (the default), a vector is generated with nlambda1 entries, starting from a guessed level lambda1.max where only the intercept is included, then shrunken to minratio*lambda1.max.

`lambda2` real scalar; tunes the ℓ_2 penalty in the Elastic-net. Default is 0.01. Set to 0 to recover the Lasso.

`weights` vector with real positive values that weight the observations (like in weighted least square). Default sets all weights to 1.

`penscale` vector with real positive values that weight the penalty of each feature. Default sets all weights to 1.

`struct` matrix structuring the coefficients, possibly sparsely encoded. Must be at least positive semidefinite (this is checked internally). If `NULL` (the default), the identity matrix is used. See details below.

`intercept` logical; indicates if an intercept should be included in the model. Default is `TRUE`.

`normalize` logical; indicates if variables should be normalized to have unit L2 norm before fitting. Default is `TRUE`.

`refit` logical: indicates if the non null coefficients should be refit to avoid excessive bias. Default is FALSE. Can be changed later (both raw and refit coefficients are stored).

`nlambda1` integer that indicates the number of values to put in the `lambda1` vector. Ignored if `lambda1` is provided.

`minratio` minimal value of ℓ_1-part of the penalty that will be tried, as a fraction of the maximal `lambda1` value. A too small value might lead to instability at the end of the solution path corresponding to small `lambda1` combined with $\lambda_2 = 0$. The default value tries to avoid this, adapting to the '$n < p$' context. Ignored if `lambda1` is provided.

`maxfeat` integer; limits the number of features ever to enter the model; i.e., non-zero coefficients for the Elastic-net: the algorithm stops if this number is exceeded and `lambda1` is cut at the corresponding level. Default is `min(nrow(x),ncol(x))` for small `lambda2` (<0.01) and `min(4*nrow(x),ncol(x))` otherwise. Use with care, as it considerably changes the computation time.

`beta0` a starting point for the vector of parameter. By default, will initialized zero. May save time in some situation.

`control` list of argument controlling low level options of the algorithm –use with care and at your own risk– :

- `verbose`: integer; activate verbose mode –this one is not too risky!– set to `0` for no output; `1` for warnings only, and `2` for tracing the whole progression. Default is `1`. Automatically set to `0` when the method is embedded within cross-validation or stability selection.
- `timer`: logical; use to record the timing of the algorithm. Default is `FALSE`.
- `maxiter` the maximal number of iteration used in the active set algorithm to solve the problem for a given value of lambda1 . Default is 50.
- `maxadd` the maximal number of variables (or groups, for group-sparse models) activated at once in the active set algorithm, chosen among the largest violations of the optimality conditions. Default is 10 (5 for group-sparse models).
- `method` a string for the underlying solver used. Either `"quadra"`, `"fista"` or `"pgd"`. Default is `"quadra"`.

- `factmat` Boolean indicating if matrix factorization should be used to solve the sub-system. If `TRUE` (the default), a Cholesky decomposition is maintained along the path. If `FALSE`, the sub-system are solved with a conjugate gradient algorithm.
- `threshold` a threshold for convergence. The algorithm stops when the optimality conditions are fulfill up to this threshold. Default is `1e-6`.
- `monitor` indicates if a monitoring of the convergence should be recorded, by computing a lower bound between the current solution and the optimum: when `'0'` (the default), no monitoring is provided; when `'1'`, the bound derived in Grandvalet et al. is computed; when `'>1'`, the Fenchel duality gap is computed along the algorithm.

Details

The optimized criterion is the following:

$$\hat{\theta}_{\lambda_1,\lambda_2} = \arg\min_{\theta=\beta+\delta} \frac{1}{2n}(y - X(\beta+\delta))^T(y - X(\beta+\delta)) + \lambda_1\|\beta\|_1 + \frac{\lambda_2}{2}\delta^T S\delta,$$

.

Value

an object with class QuadrupenFit.

an object with class LavaFit, inheriting from QuadrupenFit.

References

Chernozhukov, Victor, Christian Hansen, and Yuan Liao. "A lava attack on the recovery of sums of dense and sparse signals." The Annals of Statistics (2017): 39-76. doi:10.1214/16-AOS1434

Examples

```
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta  <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
delta <- runif(sum(c(25,10,25,10,25)),-.1,.1)
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- "relevant"
## The solution path of the LAVA
out <- lava(x,y)
out$plot_path(component = "sparse", labels=labels)
```

```
out$plot_path(component = "dense", labels=labels)
```

LavaFit *Class "LavaFit"*

Description

Class of object returned by the fitting function `lava()`. Inherits fields and methods of QuadrupenFit

Super class

`QuadrupenFit` -> `LavaFit`

Active bindings

`penalty` character describing the regularizer/penalty

`lambda1` vector of tuning parameters for the l1 penalty (sparse component)

`lambda2` vector of tuning parameters for the l2 penalty (dense component)

`sparse_coef` sparse part of the decomposition of the coefficients

`dense_coef` dense part of the decomposition of the coefficients

`debias` logical, should we rely on the debias coefficient of the regularizer (if available) or not

Methods

Public methods:

- `LavaFit$new()`
- `LavaFit$fit()`
- `LavaFit$plot_path()`
- `LavaFit$clone()`

`LavaFit$new()`: Initialize a `LavaFit` model

Usage:

```
LavaFit$new(data, intercept, regParam)
```

Arguments:

`data` a `DataModel` object

`intercept` a logical; should an intercept be included in the mode?

`regParam` a list with two elements, a vector and a scalar, for the regularization

`LavaFit$fit()`: function performing the optimization

Usage:

```
LavaFit$fit(control)
```

Arguments:

`control` list controlling the optimization process Plot method for lava regularization path

`LavaFit$plot_path()`: Produce a plot of the solution path of a LavaFit object.

Usage:

```
LavaFit$plot_path(
  xvar = c("lambda", "fraction", "df"),
  log_scale = TRUE,
  component = "both",
  title = paste("Lava path:", component, "component(s)"),
  standardize = TRUE,
  labels = NULL
)
```

Arguments:

`xvar` variable to plot on the X-axis: either `"lambda"` (ℓ_1 penalty level, or ℓ_2 for ridge and ℓ_∞) or `"fraction"` (ℓ_1-norm of the coefficients) or `df` for estimated degrees of freedom. Default is set to `"lambda"`.

`log_scale` logical; indicates if a log-scale should be used when `xvar="lambda"`. Default is `TRUE`.

`component` a character indicating the component to plot: both (sum of sparse and dense), sparse or dense. Default to both.

`title` the title. Default is set to the model name followed by what is on the Y-axis.

`standardize` logical; standardize the coefficients before plotting (with the norm of the predictor). Default is `TRUE`.

`labels` vector indicating the names associated to the plotted variables. When specified, a legend is drawn in order to identify each variable. Only relevant when the number of predictor is small. Remind that the intercept does not count. Default is `NULL`.

Returns: a **ggplot2** object .

`LavaFit$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LavaFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

QuadrupenFit, `lava()`

plot.QuadrupenFit *Plot method for quadrupen objects*

Description

S3 plot methods for QuadrupenFit, CrossValidation and StabilityPath objects, delegating to their respective R6 `$plot()` method.

Usage

```
## S3 method for class 'QuadrupenFit'
plot(x, ...)

## S3 method for class 'CrossValidation'
plot(x, ...)

## S3 method for class 'StabilityPath'
plot(x, ...)
```

Arguments

x	a QuadrupenFit, CrossValidation or StabilityPath object.
...	additional arguments passed to the underlying R6 `$plot()` method. For QuadrupenFit: `type` (`"path"`, `"criteria"`, `"crossval"`, `"stability"`), `log_scale`, `labels`. For CrossValidation: `log_scale`, `title`. For StabilityPath: `xvar`, `title`, `labels`, `sel_mode`, `cutoff`, `PFER`, `nvarsel`.

Value

a **ggplot2** object.

Functions

- `plot(CrossValidation)`: Plot method for a CrossValidation object
- `plot(StabilityPath)`: Plot method for a StabilityPath object

predict.QuadrupenFit *Perform model prediction*

Description

Predict response for new sample based on the current model

Usage

```
## S3 method for class 'QuadrupenFit'
predict(object, newx = NULL, selection = NULL, ...)
```

Arguments

object	an R object to evaluate
newx	matrix of new values for the regressor with which to predict. If omitted, the fitted values are used.
selection	either a character (model selection criteria) of a scalar (lambda value)
...	not used, only here for S3 compatibility

Value

a vector of predicted value

QuadrupenFit *Class "QuadrupenFit"*

Description

Class of object returned by any fitting function of the **quadrupen** package (`elastic_net` or `bounded_reg`).

This class comes with the usual `predict()`, `fitted()`, `coef()`, `residuals()`, `show()`, `print()` and `deviance()` S3 methods.

Specific R6 methods are available for model extraction `QuadrupenFit$get_model()`, cross validation `QuadrupenFit$cross_validate()`, stability selection `QuadrupenFit$stability_path()`, criteria derivation QuadrupenFit$criteria() and plotting `QuadrupenFit$plot()`. They come with equivalent S3 methods : `cross_validate()`, `stability()` and `plot()`.

The `"path"`plot is available as soon as a fit has been performed. For the others, the appropriate post-treatments must have been made via the methods `QuadrupenFit$criteria()`, `QuadrupenFit$cross_validate()` or `QuadrupenFit$stability()`

All plots functions are given with the default arguments, except for `labels` and `log_scale`. If you need more control, please use the dedicated methods: `QuadrupenFit$plot_path()`, `InformationCriteria$plot()`, `CrossValidation$plot()`, `StabilityPath$plot()` or the corresponding S3 methods.

Plot method for regularization path

Active bindings

`nvar` number of coefficient (without intercept)

`nobs` sample size

`dataModel` an object with class `DataModel` storing the data

`major_tuning` vector of "leading" tuning parameters (either l1, linf or l2)

`minor_tuning` vector of "minor" tuning parameters (either l1 or l2)

`is_l2_regularized` Boolean indicating if l2 regularization is applied

`optim_monitoring` list monitoring the optimization

`optim_config` list with low level options used for optimization.

`fitted` Matrix of fitted values, each column corresponding to a value of `lambda1`.

`coefficients` Matrix (class `"dgCMatrix"`) of coefficients with respect to the original input. The number of rows corresponds the length of `lambda1`.

`intercept` A vector containing the successive values of the (unpenalized) intercept. Equals to zero if `intercept` has been set to `FALSE`.

`debias` logical, should we rely on the debias coefficient of the regularizer (if available) or not

`residuals` Matrix of residuals, each column corresponding to a value of `lambda1`.

`deviance` the model deviance

`degrees_freedom` Estimated degree of freedoms for the successive `lambda1`.

`r_squared` vector giving the coefficient of determination as a function of lambda1.

`information_criteria` object with class `InformationCriteria` storing various information criteria (AIC, BIC, GCV, etc) for the current fit.

`cross_validation` object with class `CrossValidation` storing output of CV job. Only available once method cross_validate has been called.

`stability_path` object with class `StabilityPath` storing output of stability selection. Only available once method $stability has been called.

Methods

Public methods:

- `QuadrupenFit$new()`
- `QuadrupenFit$show()`
- `QuadrupenFit$print()`
- `QuadrupenFit$fit()`
- `QuadrupenFit$get_model()`
- `QuadrupenFit$predict()`
- `QuadrupenFit$cross_validate()`
- `QuadrupenFit$stability()`
- `QuadrupenFit$criteria()`
- `QuadrupenFit$plot()`
- `QuadrupenFit$plot_path()`
- `QuadrupenFit$clone()`

`QuadrupenFit$new()`: Initialize a QuadrupenFit model

Usage:

```
QuadrupenFit$new(data, intercept, regParam)
```

Arguments:

`data` a DataModel object

`intercept` a logical; should an intercept be included in the mode?

regParam a list with two elements, a vector and a scalar, for the regularization

`QuadrupenFit$show()`: User friendly print method

Usage:

```
QuadrupenFit$show()
```

`QuadrupenFit$print()`: User friendly print method

Usage:

```
QuadrupenFit$print()
```

`QuadrupenFit$fit()`: function performing the optimization

Usage:

```
QuadrupenFit$fit(control)
```

Arguments:

control list controlling the optimization process

`QuadrupenFit$get_model()`: Model extraction

Usage:

```
QuadrupenFit$get_model(selection, type = c("coefficients", "penalty", "index"))
```

Arguments:

selection either a character (model selection criteria) of a scalar (lambda value)

type character for the desired output

Returns: either a vector of coefficients, a scalar or the model index

`QuadrupenFit$predict()`: Predict response for new sample based on the current model

Usage:

```
QuadrupenFit$predict(newx = NULL, selection = NULL)
```

Arguments:

newx matrix of new values for the regressor with which to predict. If omitted, the fitted values are used.

selection either a character (model selection criteria) of a scalar (lambda value)

Returns: a vector of predicted value Cross-validation for Quadrupen object

`QuadrupenFit$cross_validate()`: Function that computes K-fold cross-validated error of a quadrupen fit, possibly on a grid of lambda1, lambda2.

Usage:

```
QuadrupenFit$cross_validate(
  K = 10,
  folds = split(sample(1:self$nobs), rep(1:K, length = self$nobs)),
  lambda2 = self$minor_tuning,
  verbose = TRUE,
  cores = 1
)
```

Arguments:

`K` integer indicating the number of folds. Default is 10.

`folds` list of `K` vectors that describes the folds to use for the cross-validation. By default, the folds are randomly sampled with the specified K. The same folds are used for each values of `lambda2`.

`lambda2` tunes the ℓ_2-penalty (ridge-like) of the fit. If none is provided, a vector of values is generated and a CV is performed on a grid of `lambda2` and `lambda1`, using the same folds for each `lambda2`.

`verbose` logical; indicates if the progression (the current `lambda2` should be displayed. Default is `TRUE`.

`cores` the number of cores to use. The default uses 1 core (safer in case your BLAS/LAPACK libraries are multithreaded)

Returns: an object with class CrossValidation is sent back and stored as a field of the original QuadrupenFit object.

`QuadrupenFit$stability()`**:** Compute the stability path of a (possibly randomized) fitting procedure as introduced by Meinshausen and Buhlmann (2010).

Usage:

```
QuadrupenFit$stability(
  n_subsamples = 50,
  subsample_size = floor(self$nobs/2),
 subsamples = replicate(n_subsamples, sample(1:self$nobs, subsample_size), simplify =
    FALSE),
  weakness = 1,
  verbose = TRUE,
  cores = 1
)
```

Arguments:

`n_subsamples` integer indicating the number of subsamplings used to estimate the selection probabilities. Default is 100.

`subsample_size` integer indicating the size of each subsamples. Default is `floor(n/2)`.

`subsamples` list with `subsamples` entries with vectors describing the folds to use for the stability procedure. By default, the folds are randomly sampled with the specified `n_subsamples` and `subsample_size` argument.

`weakness` Coefficient used for randomizing the weights of each features. Default is 1‘ for no randomization. See details below.

`verbose` logical; indicates if the progression should be displayed. Default is `TRUE`.

`cores` the number of cores to use. The default uses 1 core (safer in case your BLAS/LAPACK libraries are multithreaded)

Returns: an object with class StabilityPath is sent back and stored as a field of the original QuadrupenFit object.

`QuadrupenFit$criteria()`**:** Produce a plot or send back the values of some penalized criteria accompanied with the vector(s) of parameters selected accordingly. The default behavior plots the BIC and the AIC (with respective factor $\log(n)$ and 2) yet the user can specify any penalty.

Usage:

```
QuadrupenFit$criteria(
  penalty = setNames(c(2, log(self$nobs), log(self$nvar), log(self$nobs) + 2 *
    log(self$nvar)), c("AIC", "BIC", "mBIC", "eBIC")),
  sigma = NULL
)
```

Arguments:

`penalty` a vector with as many penalties a desired. The default contains the penalty corresponding to the AIC and the BIC (2 and $\log(n)$). Setting the "names" attribute, as done in the default definition, leads to outputs which are easier to read.

`sigma` scalar: an estimate of the residual variance. When available, it is plugged-in the criteria, which may be more relevant. If `NULL` (the default), it is estimated as usual (see details).

Returns: an object with class InformationCriteria is sent back and stored as a field of the original QuadrupenFit object.

`QuadrupenFit$plot()`**:** Plot method for QuadrupenFit

Usage:

```
QuadrupenFit$plot(
  type = c("path", "criteria", "crossval", "stability"),
  log_scale = TRUE,
  labels = NULL
)
```

Arguments:

`type` the type of plot, either `"path"` for regularization path; `"criteria"` for BIC-like information criteria ; `"crossval"` for cross-validation plot ; and `"stability"` for stability path.

`log_scale` logical; indicates if a log-scale should be used when `xvar="lambda"`. Default is `TRUE`.

`labels` vector indicating the names associated to the plotted variables. When specified, a legend is drawn in order to identify each variable. Only relevant when the number of predictor is small. Remind that the intercept does not count. Default is `NULL`.

`QuadrupenFit$plot_path()`**:** Produce a plot of the solution path of a QuadrupenFit object.

Usage:

```
QuadrupenFit$plot_path(
  xvar = c("lambda", "fraction", "df"),
  log_scale = TRUE,
  title = paste("Path for", self$penalty),
  standardize = TRUE,
  labels = NULL
)
```

Arguments:

`xvar` variable to plot on the X-axis: either `"lambda"` (ℓ_1 penalty level, or ℓ_2 for ridge and ℓ_∞) or `"fraction"` (ℓ_1-norm of the coefficients) or `df` for estimated degrees of freedom. Default is set to `"lambda"`.

`log_scale` logical; indicates if a log-scale should be used when `xvar="lambda"`. Default is `TRUE`.

`title` the title. Default is set to the model name followed by what is on the Y-axis.

`standardize` logical; standardize the coefficients before plotting (with the norm of the predictor). Default is `TRUE`.

`labels` vector indicating the names associated to the plotted variables. When specified, a legend is drawn in order to identify each variable. Only relevant when the number of predictor is small. Remind that the intercept does not count. Default is `NULL`.

Returns: a **ggplot2** object .

Examples:

```
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

## Plot the Lasso path
plot(lasso(x,y), title="Lasso solution path")
## Plot the Elastic-net path
plot(elastic_net(x,y), title = "Elastic-net solution path")
## Plot the Elastic-net path (fraction on X-axis, unstandardized coefficient)
plot(elastic_net(x,y, lambda2=10), standardize=FALSE, xvar="fraction")
## Plot the Bounded regression path (fraction on X-axis)
plot(bounded_reg(x,y, lambda2=10), xvar="fraction")
```

`QuadrupenFit$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
QuadrupenFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

See also `InformationCriteria`, `CrossValidation` and `StabilityPath`

`cross_validate()` Stability selection for Quadrupen object

`stability()` Penalized criteria based on estimation of degrees of freedom

`criteria()`

Examples

```
## ------------------------------------------------
## Method `QuadrupenFit$plot_path()`
## ------------------------------------------------

## Not run:
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

## Plot the Lasso path
plot(lasso(x,y), title="Lasso solution path")
## Plot the Elastic-net path
plot(elastic_net(x,y), title = "Elastic-net solution path")
## Plot the Elastic-net path (fraction on X-axis, unstandardized coefficient)
plot(elastic_net(x,y, lambda2=10), standardize=FALSE, xvar="fraction")
## Plot the Bounded regression path (fraction on X-axis)
plot(bounded_reg(x,y, lambda2=10), xvar="fraction")

## End(Not run)
```

`residuals.QuadrupenFit` *Extract model residuals*

Description

Extracts model residuals from a QuadrupenFit object

Usage

```
## S3 method for class 'QuadrupenFit'
residuals(object, newx = NULL, newy = NULL, ...)
```

Arguments

`object`	a QuadrupenFit object
`newx`	matrix of new covariates for out-of-sample residuals. Must be provided together with `newy`. If `NULL` (default), training residuals are returned.

newy	vector of new responses for out-of-sample residuals. Must be provided together with newx. If NULL (default), training residuals are returned.
...	not used, only here for S3 compatibility

Value

Matrix of residuals, each column corresponding to a value of lambda1.

---

ridge *Fit a linear model with a structured ridge regularization*

Description

Adjust a linear model with ridge regularization (possibly structured ℓ_2-norm). The solution path is computed at a grid of values for the ℓ_2-penalty. See details for the criterion optimized.

Usage

```
ridge(
  x,
  y,
  lambda = NULL,
  weights = rep(1, nrow(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
  penscale = rep(1, ncol(x)),
  intercept = TRUE,
  normalize = TRUE,
  nlambda = 100,
  minratio = 1e-05,
  lambda_max = 100,
  control = list()
)
```

Arguments

x	matrix of features, possibly sparsely encoded (experimental). Do NOT include intercept. When normalized is TRUE, coefficients will then be rescaled to the original scale.
y	response vector.
lambda	sequence of decreasing ℓ_2-penalty levels. If NULL (the default), a vector is generated with nlambda entries, starting from a guessed level lambda_max where only the intercept is included, then shrunken to minratio*lambda_max.
weights	vector with real positive values that weight the observations (like in weighted least square). Default sets all weights to 1.
struct	matrix structuring the coefficients, possibly sparsely encoded. Must be at least positive semidefinite (this is checked internally). If NULL (the default), the identity matrix is used. See details below.

penscale — vector with real positive values that weight the penalty of each feature. Default sets all weights to 1.

intercept — logical; indicates if an intercept should be included in the model. Default is TRUE.

normalize — logical; indicates if variables should be normalized to have unit L2 norm before fitting. Default is TRUE.

nlambda — integer that indicates the number of values to put in the lambda vector. Ignored if lambda is provided.

minratio — minimal value of ℓ_1-part of the penalty that will be tried, as a fraction of the maximal lambda1 value. A too small value might lead to instability at the end of the solution path corresponding to small lambda1 combined with $\lambda_2 = 0$. The default value tries to avoid this, adapting to the '$n < p$' context. Ignored if lambda1 is provided.

lambda_max — the largest value of lambda considered

control — list of argument controlling low level options of the algorithm –use with care and at your own risk– :

- verbose: integer; activate verbose mode –this one is not too risky!– set to 0 for no output; 1 for warnings only, and 2 for tracing the whole progression. Default is 1. Automatically set to 0 when the method is embedded within cross-validation or stability selection.
- timer: logical; use to record the timing of the algorithm. Default is FALSE.
- maxiter the maximal number of iteration used in the active set algorithm to solve the problem for a given value of lambda1 . Default is 50.
- maxadd the maximal number of variables (or groups, for group-sparse models) activated at once in the active set algorithm, chosen among the largest violations of the optimality conditions. Default is 10 (5 for group-sparse models).
- method a string for the underlying solver used. Either "quadra", "fista" or "pgd". Default is "quadra".
- factmat Boolean indicating if matrix factorization should be used to solve the sub-system. If TRUE (the default), a Cholesky decomposition is maintained along the path. If FALSE, the sub-system are solved with a conjugate gradient algorithm.
- threshold a threshold for convergence. The algorithm stops when the optimality conditions are fulfill up to this threshold. Default is 1e-6.
- monitor indicates if a monitoring of the convergence should be recorded, by computing a lower bound between the current solution and the optimum: when '0' (the default), no monitoring is provided; when '1', the bound derived in Grandvalet et al. is computed; when '>1', the Fenchel duality gap is computed along the algorithm.

Details

The optimized criterion is the following:

$$\hat{\beta}_{\lambda_2} = \arg\min_{\beta} \frac{1}{2}(y - X\beta)^T(y - X\beta) + \frac{\lambda_2}{2}\beta^T S\beta,$$

where the ℓ_2 structuring positive semidefinite matrix S is provided via the `struct` argument (possibly of class `Matrix`).

Value

an object with class RidgeRegressionFit, inheriting from QuadrupenFit.

See Also

See also QuadrupenFit

Examples

```
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- "relevant"
plot(ridge(x,y) , label=labels) ## a mess
plot(ridge(x,y, struct=solve(Sigma)), label=labels) ## even better
```

RidgeRegressionFit *Class "RidgeRegressionFit"*

Description

Class of object returned by the fitting function `ridge()`. Inherits fields and methods of QuadrupenFit.

Super class

`QuadrupenFit` -> `RidgeRegressionFit`

Active bindings

`penalty` character describing the regularizer/penalty

`lambda2` vector of tuning parameters for the l2 penalty

Methods

Public methods:

- RidgeRegressionFit$new()
- RidgeRegressionFit$clone()

`RidgeRegressionFit$new()`: Initialize a RidgeRegressionFit model

Usage:

```
RidgeRegressionFit$new(data, intercept, regParam)
```

Arguments:

`data` a DataModel object

`intercept` a logical; should an intercept be included in the mode?

`regParam` a list with two elements, a vector and a scalar, for the regularization

`RidgeRegressionFit$clone()`: The objects of this class are cloneable with this method.

Usage:

```
RidgeRegressionFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

QuadrupenFit, `ridge()`

`selection` *Variable selection from a stability path*

Description

S3 generic for variable selection based on a StabilityPath object, as introduced by Meinshausen and Buhlmann (2010).

Usage

```
selection(object, ...)

## S3 method for class 'StabilityPath'
selection(
  object,
  sel_mode = c("rank", "PFER"),
  cutoff = 0.75,
  PFER = 2,
  nvarsel = NULL,
  ...
)
```

Arguments

`object`	a StabilityPath object.
`...`	not used, only here for S3 compatibility.
`sel_mode`	a character string, either `"rank"` or `"PFER"`. Default is `"rank"`.
`cutoff`	probability threshold for `sel_mode = "PFER"`. Default is `0.75`.
`PFER`	per-family error rate to control for `sel_mode = "PFER"`. Default is `2`.
`nvarsel`	number of variables to select for `sel_mode = "rank"`. Default is `floor(nobs / log(nvar))`.

Value

an integer vector of selected variable indices.

Methods (by class)

- `selection(StabilityPath)`: S3 method for variable selection from a StabilityPath

References

N. Meinshausen and P. Buhlmann (2010). Stability Selection, JRSS(B).

See Also

`stability()`

`sparse_lm` *Fit a linear model with sparse regularization*

Description

Adjust a linear model with sparse regularization. We also add a (possibly structured) ℓ_2-norm (ridge-like). The solution path is computed at a grid of values for the ℓ_1-penalty, fixing the amount of ℓ_2 regularization. See details for the criterion optimized.

Usage

```
sparse_lm(
  x,
  y,
  type = c("l1", "mcp", "scad"),
  lambda1 = NULL,
  lambda2 = 0.01,
  eta = 3.7,
  weights = rep(1, nrow(x)),
  penscale = rep(1, ncol(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
```

```
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = ifelse(nrow(x) <= ncol(x), 0.01, 1e-04),
 maxfeat = ifelse(lambda2 < 0.01, min(nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
  beta0 = numeric(ncol(x)),
  control = list()
)

elastic.net(
  x,
  y,
  lambda1 = NULL,
  lambda2 = 0.5,
  weights = rep(1, nrow(x)),
  penscale = rep(1, ncol(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = ifelse(nrow(x) <= ncol(x), 0.01, 1e-04),
 maxfeat = ifelse(lambda2 < 0.01, min(nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
  beta0 = numeric(ncol(x)),
  control = list(method = "quadra")
)

elastic_net(
  x,
  y,
  lambda1 = NULL,
  lambda2 = 0.5,
  weights = rep(1, nrow(x)),
  penscale = rep(1, ncol(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = ifelse(nrow(x) <= ncol(x), 0.01, 1e-04),
 maxfeat = ifelse(lambda2 < 0.01, min(nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
  beta0 = numeric(ncol(x)),
  control = list(method = "quadra")
)

lasso(
  x,
```

```
  y,
  lambda1 = NULL,
  weights = rep(1, nrow(x)),
  penscale = rep(1, ncol(x)),
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = ifelse(nrow(x) <= ncol(x), 0.01, 1e-04),
  maxfeat = min(nrow(x), ncol(x)),
  beta0 = numeric(ncol(x)),
  control = list(method = "quadra")
)

mcp(
  x,
  y,
  lambda1 = NULL,
  lambda2 = 0,
  eta = 3,
  weights = rep(1, nrow(x)),
  penscale = rep(1, ncol(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = ifelse(nrow(x) <= ncol(x), 0.01, 1e-04),
 maxfeat = ifelse(lambda2 < 0.01, min(nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
  beta0 = numeric(ncol(x)),
  control = list(method = "quadra")
)

scad(
  x,
  y,
  lambda1 = NULL,
  lambda2 = 0,
  eta = 3.7,
  weights = rep(1, nrow(x)),
  penscale = rep(1, ncol(x)),
  struct = Matrix::Diagonal(ncol(x), 1),
  intercept = TRUE,
  normalize = TRUE,
  refit = FALSE,
  nlambda1 = ifelse(is.null(lambda1), 100, length(lambda1)),
  minratio = ifelse(nrow(x) <= ncol(x), 0.01, 1e-04),
 maxfeat = ifelse(lambda2 < 0.01, min(nrow(x), ncol(x)), min(4 * nrow(x), ncol(x))),
```

```
  beta0 = numeric(ncol(x)),
  control = list(method = "quadra")
)
```

Arguments

x	matrix of features, possibly sparsely encoded (experimental). Do NOT include intercept. When normalized is TRUE, coefficients will then be rescaled to the original scale.
y	response vector.
type	string indicating the sparse variant to be fitted. Could be "l1", "mcp" or "scad". Default is "l1". be careful as scad and mcp are still experimental and have not been fully tested yet
lambda1	sequence of decreasing ℓ_1-penalty levels. If NULL (the default), a vector is generated with nlambda1 entries, starting from a guessed level lambda1.max where only the intercept is included, then shrunken to minratio*lambda1.max.
lambda2	real scalar; tunes the ℓ_2 penalty in the Elastic-net. Default is 0.01. Set to 0 to recover the Lasso.
eta	real positive scalar for tuning SCAD or MCP penalties. Default is 3.7. Ignored when type == "l1".
weights	vector with real positive values that weight the observations (like in weighted least square). Default sets all weights to 1.
penscale	vector with real positive values that weight the penalty of each feature. Default sets all weights to 1.
struct	matrix structuring the coefficients, possibly sparsely encoded. Must be at least positive semidefinite (this is checked internally). If NULL (the default), the identity matrix is used. See details below.
intercept	logical; indicates if an intercept should be included in the model. Default is TRUE.
normalize	logical; indicates if variables should be normalized to have unit L2 norm before fitting. Default is TRUE.
refit	logical: indicates if the non null coefficients should be refit to avoid excessive bias. Default is FALSE. Can be changed later (both raw and refit coefficients are stored).
nlambda1	integer that indicates the number of values to put in the lambda1 vector. Ignored if lambda1 is provided.
minratio	minimal value of ℓ_1-part of the penalty that will be tried, as a fraction of the maximal lambda1 value. A too small value might lead to instability at the end of the solution path corresponding to small lambda1 combined with $\lambda_2 = 0$. The default value tries to avoid this, adapting to the '$n < p$' context. Ignored if lambda1 is provided.
maxfeat	integer; limits the number of features ever to enter the model; i.e., non-zero coefficients for the Elastic-net: the algorithm stops if this number is exceeded and lambda1 is cut at the corresponding level. Default is min(nrow(x),ncol(x)) for small lambda2 (<0.01) and min(4*nrow(x),ncol(x)) otherwise. Use with care, as it considerably changes the computation time.

`beta0` a starting point for the vector of parameter. By default, will initialized zero. May save time in some situation.

`control` list of argument controlling low level options of the algorithm –use with care and at your own risk– :

- `verbose`: integer; activate verbose mode –this one is not too risky!– set to `0` for no output; `1` for warnings only, and `2` for tracing the whole progression. Default is `1`. Automatically set to `0` when the method is embedded within cross-validation or stability selection.
- `timer`: logical; use to record the timing of the algorithm. Default is `FALSE`.
- `maxiter` the maximal number of iteration used in the active set algorithm to solve the problem for a given value of lambda1 . Default is 50.
- `maxadd` the maximal number of variables (or groups, for group-sparse models) activated at once in the active set algorithm, chosen among the largest violations of the optimality conditions. Default is 10 (5 for group-sparse models).
- `method` a string for the underlying solver used. Either `"quadra"`, `"fista"` or `"pgd"`. Default is `"quadra"`.
- `factmat` Boolean indicating if matrix factorization should be used to solve the sub-system. If `TRUE` (the default), a Cholesky decomposition is maintained along the path. If `FALSE`, the sub-system are solved with a conjugate gradient algorithm.
- `threshold` a threshold for convergence. The algorithm stops when the optimality conditions are fulfill up to this threshold. Default is `1e-6`.
- `monitor` indicates if a monitoring of the convergence should be recorded, by computing a lower bound between the current solution and the optimum: when `'0'` (the default), no monitoring is provided; when `'1'`, the bound derived in Grandvalet et al. is computed; when `'>1'`, the Fenchel duality gap is computed along the algorithm.

Details

The optimized criterion is the following:

$$\hat{\beta}_{\lambda_1,\lambda_2} = \arg\min_{\beta} \frac{1}{2}(y - X\beta)^T(y - X\beta) + \lambda_1 pen_\eta(D\beta) + \frac{\lambda_2}{2}\beta^T S\beta,$$

where D is a diagonal matrix, whose diagonal terms are provided as a vector by the `penscale` argument. The ℓ_2 structuring matrix S is provided via the `struct` argument, a positive semidefinite matrix (possibly of class `Matrix`).

Value

an object with class SparseFit, inheriting from QuadrupenFit.

See Also

See also SparseFit

Examples

```
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
cor <- 0.75
Soo <- toeplitz(cor^(0:(25-1))) ## Toeplitz correlation for irrelevant variables
Sww  <- matrix(cor,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo)
diag(Sigma) <- 1
n <- 50
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)
labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- "relevant"

## Lasso
plot(lasso(x, y), label=labels)

## SCAD
plot(scad(x, y), label=labels)

## MCP
plot(mcp(x, y), label=labels)

## Elastic-net
plot(elastic_net(x,y,lambda2=1), label=labels)

## Structured Elastic-net (l2-structuring prior)
plot(elastic_net(x,y,lambda2=3,struct=solve(Sigma)), label=labels)

## SCAD + L2
plot(scad(x,y, eta = 3.7, lambda2=1), label=labels)

## MCP + L2
plot(mcp(x, y, eta = 3, lambda2=1), label=labels)
```

SparseFit *Class "SparseFit"*

Description

Class of object returned by the fitting function `elastic_net()`. Inherits fields and methods of QuadrupenFit

Super class

`QuadrupenFit` -> `SparseFit`

Active bindings

`lambda1` vector of tuning parameters for the l1 penalty

`lambda2` vector of tuning parameters for the l2 penalty

`penalty` character describing the regularizer/penalty

`type` string the type of group-wise regularization applied

`unbiasing_tuning` unbiasing coefficient of the MCP or SCAD penalties

Methods

Public methods:

- `SparseFit$new()`
- `SparseFit$clone()`

`SparseFit$new()`: Initialize a `SparseFit` model

Usage:

```
SparseFit$new(data, intercept, type, regParam)
```

Arguments:

`data` a `DataModel` object

`intercept` a logical; should an intercept be included in the mode?

`type` string the type of group-wise regularization applied

`regParam` a list with two elements, a vector and a scalar, for the regularization

`SparseFit$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SparseFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

QuadrupenFit, `elastic_net()`

`SparseGroupFit` *Class "SparseGroupFit"*

Description

Class of object returned by the fitting function `group_sparse_lm()`. Inherits fields and methods of QuadrupenFit

Super class

`QuadrupenFit` -> `SparseGroupFit`

Active bindings

`lambda1` vector of tuning parameters for the l1 group penalty

`lambda2` vector of tuning parameters for the l2 penalty

`alpha` mixing parameter of the sparse group-penalty

`penalty` character describing the regularizer/penalty

`group` vector of integers indicating group belonging

`type` string the type of group-wise regularization applied

`mixture_tuning` mixture coefficient of the sparse group penalty

`is_group_sparse` boolean indicating if sparse group or group penalty is applied

Methods

Public methods:

- `SparseGroupFit$new()`
- `SparseGroupFit$clone()`

`SparseGroupFit$new()`: Initialize a `SparseGroupFit` model

Usage:

```
SparseGroupFit$new(data, intercept, group, type, regParam)
```

Arguments:

`data` a `DataModel` object

`intercept` a logical; should an intercept be included in the mode?

`group` vector of integers indicating group belonging.

`type` string indicating whether the ℓ_1/ℓ_2 or the ℓ_1/ℓ_∞ group-Lasso must be fitted.

`regParam` a list with two elements, a vector and a scalar, for the regularization

`SparseGroupFit$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SparseGroupFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

QuadrupenFit, `group_sparse_lm()`

`stability` *Stability selection for Quadrupen object*

Description

Compute the stability path of a (possibly randomized) fitting procedure as introduced by Meinshausen and Buhlmann (2010).

Usage

```
stability(
  object,
  n_subsamples = 50,
  subsample_size = floor(object$nobs/2),
 subsamples = replicate(n_subsamples, sample(1:object$nobs, subsample_size), simplify =
    FALSE),
  weakness = 1,
  verbose = TRUE,
  cores = 1
)

## S3 method for class 'QuadrupenFit'
stability(
  object,
  n_subsamples = 50,
  subsample_size = floor(object$nobs/2),
 subsamples = replicate(n_subsamples, sample(1:object$nobs, subsample_size), simplify =
    FALSE),
  weakness = 1,
  verbose = TRUE,
  cores = parallel::detectCores() - 2
)
```

Arguments

`object`	an R6 object with class QuadrupenFit
`n_subsamples`	integer indicating the number of subsamplings used to estimate the selection probabilities. Default is 100.
`subsample_size`	integer indicating the size of each subsamples. Default is `floor(n/2)`.
`subsamples`	list with `subsamples` entries with vectors describing the folds to use for the stability procedure. By default, the folds are randomly sampled with the specified `n_subsamples` and `subsample_size` argument.
`weakness`	Coefficient used for randomizing the weights of each features. Default is 1‘ for no randomization. See details below.
`verbose`	logical; indicates if the progression should be displayed. Default is `TRUE`.
`cores`	the number of cores to use. The default uses 1 core (safer in case your BLAS/LAPACK libraries are multithreaded)

Value

an object with class StabilityPath is sent back and stored as a field of the original QuadrupenFit object.

Methods (by class)

- stability(QuadrupenFit): S3 method for stability selection of a QuadrupenFit

Note

When weakness < 1, the penscale argument that weights the penalty tuned by λ_1 is perturbed (divided) for each subsample by a random variable uniformly distributed on $[\alpha, 1]$, where α is the weakness parameter.

If the user runs the fitting method with option 'bulletproof' set to FALSE, the algorithm may stop at an early stage of the path. Early stops of the underlying fitting function are handled internally, in the following way: we chose to simply skip the results associated with such runs, in order not to bias the stability selection procedure. If it occurs too often, a warning is sent to the user, in which case you should reconsider the grid of lambda1 for stability selection. If bulletproof is TRUE (the default), there is nothing to worry about, except a possible slow down when any switching to the proximal algorithm is required.

References

N. Meinshausen and P. Buhlmann (2010). Stability Selection, JRSS(B).

Examples

```
## Not run:
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
Soo  <- matrix(0.75,25,25) ## bloc correlation between zero variables
Sww  <- matrix(0.75,10,10) ## bloc correlation between active variables
Sigma <- bdiag(Soo,Sww,Soo,Sww,Soo) + 0.2
diag(Sigma) <- 1
n <- 100
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

## Build a vector of label for true nonzeros
labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- c("relevant")
labels <- factor(labels, ordered=TRUE, levels=c("relevant","irrelevant"))

enet <- elastic_net(x, y, lambda2 = 10, struct = solve(Sigma), minratio = 1e-2)
stab <- stability(enet, n_subsamples = 200)

## Build the plot an recover the selected variable
plot(stab, labels=labels)
stabpath <- plot(stab, xvar="fraction", labels=labels, sel_mode="PFER", cutoff=0.75, PFER=1)
```

```
cat("\nFalse positives for the randomized Elastic-net with stability selection: ",
     sum(labels[stab$selection()] != "relevant"))
cat("\nDONE.\n")

## End(Not run)
```

`StabilityPath` *Class StabilityPath*

Description

Class of object returned by the `QuadrupenFit$cross_validate()` method or the `cross_validate()` function. Owns `print()` and `plot()` methods.

Public fields

`probabilities` a `Matrix` object containing the estimated probabilities of selection along the path of solutions.

`regParam` a list with the levels of the regularizing parameters used

`subsamples` a list that contains the folds used for each subsample.

Active bindings

`nvar` number of variables (without intercept)

`nobs` number of observation/sample size

`nonzero` variables with a non-null probability of selection along the stability path

`nonzeroprob` subset of the probabilities stability path on the nonzero variables

Methods

Public methods:

- `StabilityPath$new()`
- `StabilityPath$show()`
- `StabilityPath$print()`
- `StabilityPath$selection()`
- `StabilityPath$plot()`
- `StabilityPath$clone()`

`StabilityPath$new()`: Constructor for a StabilityPath object Should be called internally by an object `QuadrupenFit$stability()`

Usage:

```
StabilityPath$new(probabilities, regParam, subsamples)
```

Arguments:

probabilities a Matrix object containing the estimated probabilities of selection along the path of solutions.

regParam a list with the levels of the regularizing parameters used

subsamples a list that contains the folds used for each subsample.

`StabilityPath$show()`: User friendly print method

Usage:

```
StabilityPath$show()
```

`StabilityPath$print()`: User friendly print method

Usage:

```
StabilityPath$print()
```

`StabilityPath$selection()`: Perform variable selection based on the stability path

Usage:

```
StabilityPath$selection(
  sel_mode = c("rank", "PFER"),
  cutoff = 0.75,
  PFER = 2,
  nvarsel = floor(self$nobs/log(self$nvar))
)
```

Arguments:

sel_mode a character string, either "rank" or "PFER". In the first case, the selection is based on the rank of total probabilities by variables along the path: the first nvarsel variables are selected (see below). In the second case, the PFER control is used as described in Meinshausen and Buhlmannn's paper. Default is "rank".

cutoff value of the cutoff probability (only relevant when sel_mode equals "PFER").

PFER value of the per-family error rate to control (only relevant when sel_mode equals "PFER").

nvarsel number of variables selected (only relevant when sel_mode equals "rank". Default is floor(n/log(p)).

`StabilityPath$plot()`: Produce a plot of the stability path obtained by stability selection.

Usage:

```
StabilityPath$plot(
  xvar = "lambda",
  title = "Stability path",
  labels = rep("unknown status", self$nvar),
  sel_mode = c("rank", "PFER"),
  cutoff = 0.75,
  PFER = 2,
  nvarsel = min(self$nvar, floor(self$nobs/log(self$nvar)))
)
```

Arguments:

xvar variable to plot on the X-axis: either "lambda" (first penalty level) or "fraction" (fraction of the penalty level applied tune by λ_1). Default is "lambda".

`title` title title. If none given, a somewhat appropriate title is automatically generated.

`labels` an optional vector of labels for each variable in the path (e.g., 'relevant'/'irrelevant'). See examples.

`sel_mode` a character string, either `"rank"` or `"PFER"`. In the first case, the selection is based on the rank of total probabilities by variables along the path: the first `nvarsel` variables are selected (see below). In the second case, the PFER control is used as described in Meinshausen and Buhlmannn's paper. Default is `"rank"`.

`cutoff` value of the cutoff probability (only relevant when `sel_mode` equals `"PFER"`).

`PFER` value of the per-family error rate to control (only relevant when `sel_mode` equals `"PFER"`).

`nvarsel` number of variables selected (only relevant when `sel_mode` equals `"rank"`. Default is `floor(n/log(p))`.

`plot` logical; indicates if the graph should be plotted. Default is `TRUE`. If `FALSE`, only the **ggplot2** object is sent back.

Returns: a list with a **ggplot2** object which can be plotted via the `print` method, and a vector of selected variables corresponding to method of choice (`"rank"` or `"PFER"`).

Examples:

```
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
Soo  <- matrix(0.75,25,25) ## bloc correlation between zero variables
Sww  <- matrix(0.75,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo) + 0.2
diag(Sigma) <- 1
n <- 100
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

## Build a vector of label for true nonzeros
labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- c("relevant")
labels <- factor(labels, ordered=TRUE, levels=c("relevant","irrelevant"))

enet <- elastic_net(x, y, lambda2 = 10, struct = solve(Sigma), minratio = 1e-2)
stab <- stability(enet, n_subsamples = 200)

## Build the plot an recover the selected variable
plot(stab, labels=labels)

cat("\nFalse positives for the randomized Elastic-net with stability selection: ",
     sum(labels[stab$selection()] != "relevant"))
cat("\nDONE.\n")
```

`StabilityPath$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
StabilityPath$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `StabilityPath$plot()`
## ------------------------------------------------

## Not run:
## Simulating multivariate Gaussian with blockwise correlation
## and piecewise constant vector of parameters
beta <- rep(c(0,1,0,-1,0), c(25,10,25,10,25))
Soo  <- matrix(0.75,25,25) ## bloc correlation between zero variables
Sww  <- matrix(0.75,10,10) ## bloc correlation between active variables
Sigma <- Matrix::bdiag(Soo,Sww,Soo,Sww,Soo) + 0.2
diag(Sigma) <- 1
n <- 100
x <- as.matrix(matrix(rnorm(95*n),n,95) %*% chol(Sigma))
y <- 10 + x %*% beta + rnorm(n,0,10)

## Build a vector of label for true nonzeros
labels <- rep("irrelevant", length(beta))
labels[beta != 0] <- c("relevant")
labels <- factor(labels, ordered=TRUE, levels=c("relevant","irrelevant"))

enet <- elastic_net(x, y, lambda2 = 10, struct = solve(Sigma), minratio = 1e-2)
stab <- stability(enet, n_subsamples = 200)

## Build the plot an recover the selected variable
plot(stab, labels=labels)

cat("\nFalse positives for the randomized Elastic-net with stability selection: ",
     sum(labels[stab$selection()] != "relevant"))
cat("\nDONE.\n")

## End(Not run)
```

Index

bounded.reg *(bounded_reg)*, 3
bounded_reg, 3
bounded_reg(), *6*, *19*
BoundedRegressionFit, *5*, 6, *6*

coef(), *38*
coef.QuadrupenFit, 7
coop_lasso *(group_sparse_lm)*, 23
criteria, 7
criteria(), *30*, *43*
cross_validate, 9
cross_validate(), *11*, *38*, *43*, *59*
CrossValidation, *10*, 11, *12*, *37*, *39*, *41*, *43*
CrossValidation$plot(), *38*

DataModel, *6*, 13, *19*, *29*, *35*, *38, 39*, *48*, *55, 56*
deviance(), *38*
deviance.QuadrupenFit, 15

elastic.net *(sparse_lm)*, 49
elastic_net *(sparse_lm)*, 49
elastic_net(), *8*, *54, 55*

fitted(), *38*
fitted.QuadrupenFit, 16
fused_lasso, 16
fused_lasso(), *18*
FusedLassoFit, 18, *18, 19*

group_l1linf *(group_sparse_lm)*, 23
group_lasso *(group_sparse_lm)*, 23
group_lava, 19
group_lava(), *29, 30*
group_sparse_lm, 23
group_sparse_lm(), *55, 56*
GroupLavaFit, *22*, 29, *29*

InformationCriteria, *8*, 30, *30*, *39*, *42, 43*
InformationCriteria$plot(), *38*
isQuadrupenFit, 31

lasso *(sparse_lm)*, 49
lava, 32
lava(), *35, 36*
LavaFit, *29*, *34*, 35, *35, 36*

mcp *(sparse_lm)*, 49

plot(), *11*, *30*, *38*, *59*
plot.CrossValidation *(plot.QuadrupenFit)*, 37
plot.QuadrupenFit, 37
plot.StabilityPath *(plot.QuadrupenFit)*, 37
predict(), *38*
predict.QuadrupenFit, 37
print(), *11*, *30*, *38*, *59*

QuadrupenFit, *5–10*, *15, 16*, *18, 19*, *22*, *27*, *29, 30*, *34–37*, 38, *39*, *41, 42*, *44*, *47, 48*, *53–58*
QuadrupenFit$criteria(), *30*, *38*
QuadrupenFit$cross_validate(), *11, 12*, *38*, *59*
QuadrupenFit$get_model(), *38*
QuadrupenFit$plot(), *38*
QuadrupenFit$plot_path(), *38*
QuadrupenFit$stability(), *38*, *59*
QuadrupenFit$stability_path(), *38*

residuals(), *38*
residuals.QuadrupenFit, 44
ridge, 45
ridge(), *47, 48*
RidgeRegressionFit, 47, *47, 48*

scad *(sparse_lm)*, 49
selection, 48
show(), *38*
sparse_coop_lasso *(group_sparse_lm)*, 23
sparse_group_l1linf *(group_sparse_lm)*, 23

sparse_group_lasso *(*group_sparse_lm*)*, 23
sparse_lm, 49
SparseFit, *53*, 54, *55*
SparseGroupFit, *27*, 55, *56*
stability, 57
stability(), *38*, *43*, *49*
StabilityPath, *37*, *39*, *41*, *43*, *48*, *49*, *58*, 59, *59*
StabilityPath$plot(), *38*