Package ‘HDPenReg’

October 5, 2020

Version 0.94.7
Date 2020-10-03
Encoding UTF-8
Title High-Dimensional Penalized Regression
Copyright Inria - Université de Lille
BugReports https://github.com/modal-inria/HDPenReg/issues
Depends R (>= 3.0.2), rtkore (>= 1.5.5)
Imports methods, Matrix
Description Algorithms for lasso and fused-lasso problems: implementation of
    the lars algorithm for lasso and fusion penalization and EM-based
    algorithms for (logistic) lasso and fused-lasso penalization.
License GPL (>= 2)
LinkingTo rtkore, Rcpp
SystemRequirements GNU make
RoxygenNote 7.1.1
NeedsCompilation yes
Author Quentin Grimonprez [aut, cre],
    Serge Iovleff [aut]
Maintainer Quentin Grimonprez <quentingrim@yahoo.fr>
Repository CRAN
Date/Publication 2020-10-05 07:50:02 UTC

R topics documented:

HDPenReg-package .................................................. 2
coef.LarsPath ...................................................... 3
coeff ............................................................... 4
computeCoefficients .............................................. 4
EMcvfusedlasso .................................................. 5
EMcvlasso ......................................................... 7
HDPenReg-package

Algorithms for lasso and fused-lasso problems.

Description

This package contains algorithms for lasso and fused-lasso problems. It contains an implementation of the lars algorithm [1], for the lasso and fusion penalization and EM-based algorithms for (logistic) lasso and fused-lasso.

Details

Package: HDPenReg
Type: Package
Version: 0.94.5
Date: 2019-03-29
License: GPL (>=2)

The main function is HDlars.

Author(s)

Maintainer: Quentin Grimonprez <quentin.grimonprez@inria.fr>

See Also

HDLars HDcvlars

Examples

## Not run:
# see vignette
vignette("HDPenReg")
## End(Not run)

### coef.LarsPath

**Description**

Compute coefficients at a given level of penalty

**Usage**

```r
# S3 method for class 'LarsPath'
coef(object, index = NULL, mode = c("lambda", "step", "fraction", "norm"), ...)
```

**Arguments**

- `object`: a LarsPath object
- `index`: If `mode = "norm"`, index represents the l1-norm of the coefficients with which we want to predict. If `mode = "fraction"`, index represents the ratio (l1-norm of the coefficients with which we want to predict)/(l1-norm maximal of the LarsPath object). If `mode = "lambda"`, index represents the value of the penalty parameter. If `mode = "step"`, index represents the number of the step at which we want coefficients.
- `mode`: "fraction" or "norm" or "lambda" or "step".
- `...`: other arguments. Not used

**Value**

A vector containing the estimated coefficient for index

**Author(s)**

Quentin Grimonprez

**See Also**

HDLars LarsPath

**Examples**

```r
dataset <- simul(50, 10000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- HDLars(dataset$data[1:40, ], dataset$response[1:40])
coeff <- coef(result, 0.3, "fraction")
```
computeCoefficients

coeff                  get coefficients at a given step.

Description

Get the vector of coefficients at a given step

Usage

coeff(x, step)

Arguments

x            A LarsPath object.
step        The step at which you want to get the coefficients.

Value

a vector of size p containing the value of coefficients at the desired step.

See Also

HDlars HDfusion LarsPath

Examples

dataset <- simul(50, 1000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- HDfusion(dataset$data, dataset$response)
coefficient <- coeff(result, result@nbStep) # get the coefficients

computeCoefficients   Compute coefficients

Description

Compute coefficients at a given level of penalty

Usage

computeCoefficients(x, lambda, mode = "fraction")
**Arguments**

- `x` a LarsParth object
- `lambda` If mode = "norm", lambda represents the l1-norm of the coefficients with which we want to predict. If mode = "fraction", lambda represents the ratio (l1-norm of the coefficients with which we want to predict)/(l1-norm maximal of the LarsPath object).
- `mode` "fraction" or "norm" or "lambda".

**Value**

A list containing

- `variable` Index of non-zeros coefficients.
- `coefficient` non-zeros coefficients.

**Author(s)**

Quentin Grimonprez

**Examples**

```r
dataset <- simul(50, 10000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- HDlars(dataset$data[1:40, ], dataset$response[1:40])
coeff <- computeCoefficients(result, 0.3, "fraction")
```

**Description**

cross validation function for **EMfusedlasso**.

**Usage**

```r
EMcvfusedlasso(
  x,
  y,
  lambda1,
  lambda2,
  nbFolds = 10,
  maxSteps = 1000,
  burn = 50,
  intercept = TRUE,
  model = c("linear", "logistic"),
  eps = 1e-05,
  eps0 = 1e-08,
  epsCG = 1e-08
)
```
Arguments

- **X** the matrix (of size n*p) of the covariates.
- **y** a vector of length n with the response.
- **lambda1** Values of lambda1 at which prediction error should be computed. Can be a single value.
- **lambda2** Values of lambda2 at which prediction error should be computed. Can be a single value.
- **nbFolds** the number of folds for the cross-validation.
- **maxSteps** Maximal number of steps for EM algorithm.
- **burn** Number of steps for the burn period.
- **intercept** If TRUE, there is an intercept in the model.
- **model** "linear" or "logistic".
- **eps** Tolerance of the algorithm.
- **eps0** Zero tolerance. Coefficients under this value are set to zero.
- **epsCG** Epsilon for the convergence of the conjugate gradient.

Value

A list containing

- **cv** Mean prediction error for each value of index.
- **cvError** Standard error of cv.
- **minCv** Minimal cv criterion.
- **lambda1** Values of lambda1 at which prediction error should be computed.
- **lambda2** Values of lambda2 at which prediction error should be computed.
- **lambda.optimal** Value of (lambda1,lambda2) for which the cv criterion is minimal.

Author(s)

Quentin Grimonprez, Serge Iovleff

Examples

dataset <- simul(50, 100, 0.4, 1, 10, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- EMcvfusedlasso(X = dataset$data, y = dataset$response, lambda1 = 3:1,
lambda2 = 3:1, nbFolds = 5, intercept = FALSE)
EMcvlasso
cross validation for EMlasso

Description
cross validation function for EMlasso.

Usage
EMcvlasso(
  X,  
  y,  
  lambda = NULL,  
  nbFolds = 10,  
  maxSteps = 1000,  
  intercept = TRUE,  
  model = c("linear", "logistic"),  
  burn = 30,  
  threshold = 1e-08,  
  eps = 1e-05,  
  epsCG = 1e-08  
)

Arguments

X                  the matrix (of size n*p) of the covariates.
y                  a vector of length n with the response.
lambda             Values at which prediction error should be computed.
nbFolds            the number of folds for the cross-validation.
maxSteps           Maximal number of steps for EM algorithm.
intercept          If TRUE, there is an intercept in the model.
model              "linear" or "logistic".
burn               Number of steps for the burn period.
threshold          Zero tolerance. Coefficients under this value are set to zero.
eps                Tolerance of the EM algorithm.
epsCG              Epsilon for the convergence of the conjugate gradient.

Value

A list containing

  cv    Mean prediction error for each value of index.
  cvError Standard error of lambda.
  minCv Minimal lambda criterion.
lambda  Values of lambda at which prediction error should be computed.
lambda.optimal Value of lambda for which the cv criterion is minimal.
Author(s)
Quentin Grimonprez, Serge Iovleff

Examples

dataset <- simul(50, 100, 0.4, 1, 10, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- EMcvlasso(X = dataset$data, y = dataset$response,
lambda = 5:1, nbFolds = 5, intercept = FALSE)

EMfusedlasso

EM algorithm for fused-lasso penalty

Description
EM algorithm for fused-lasso penalty

Usage

EMfusedlasso(
  X,
  y,
  lambda1,
  lambda2,
  maxSteps = 1000,
  burn = 50,
  intercept = TRUE,
  model = c("linear", "logistic"),
  eps = 1e-05,
  eps0 = 1e-08,
  epsCG = 1e-08
)

Arguments

X  the matrix (of size n*p) of the covariates.
y  a vector of length n with the response.
lambda1  a positive real. Parameter associated with the lasso penalty.
lambda2  a positive real. Parameter associated with the fusion penalty.
maxSteps  Maximal number of steps for EM algorithm.
burn  Number of steps before regrouping some variables in segment.
intercept  If TRUE, there is an intercept in the model.
model  "linear" or "logistic"
eps  tolerance for convergence of the EM algorithm.
eps0  Zero tolerance. Coefficients under this value are set to zero.
epsCG  tolerance for convergence of the conjugate gradient.
**Value**

A list containing:

- **step** Vector containing the number of steps of the algorithm for every lambda.
- **variable** List of vector of size "step+1". The i+1-th item contains the index of non-zero coefficients at the i-th step.
- **coefficient** List of vector of size "step+1". The i+1-th item contains the non-zero coefficients at the i-th step.
- **lambda** Vector of length "step+1", containing the lambda at each step.
- **mu** Intercept.

**Author(s)**

Quentin Grimonprez, Serge Iovleff

**See Also**

- `EMcvfusedlasso`

**Examples**

```r
dataset <- simul(50, 100, 0.4, 1, 10, matrix(c(0.1, 0.9, 0.02, 0.02), nrow = 2))
result <- EMfusedlasso(dataset$data, dataset$response, 1, 1)
```

---

**EMlasso**

*EM algorithm for lasso penalty*

**Description**

EM algorithm for lasso penalty

**Usage**

```r
EMlasso(
  X,
  y,
  lambda,
  maxSteps = 1000,
  intercept = TRUE,
  model = c("linear", "logistic"),
  burn = 50,
  threshold = 1e-08,
  eps = 1e-05,
  epsCG = 1e-08
)
```
EMlasso

Arguments

- **X** the matrix (of size n*p) of the covariates.
- **y** a vector of length n with the response.
- **lambda** a sequence of l1 penalty regularization term. If no sequence is provided, the function computes his own sequence.
- **maxSteps** Maximal number of steps for EM algorithm.
- **intercept** If TRUE, there is an intercept in the model.
- **model** "linear" or "logistic"
- **burn** Number of steps before thresholding some variables to zero.
- **threshold** Zero tolerance. Coefficients under this value are set to zero.
- **eps** Epsilon for the convergence of the EM algorithm.
- **epsCG** Epsilon for the convergence of the conjugate gradient.

Value

A list containing:

- **step** Vector containing the number of steps of the algorithm for every lambda.
- **variable** List of vector of the same length as lambda. The i-th item contains the index of non-zero coefficients for the i-th lambda value.
- **coefficient** List of vector of the same length as lambda. The i-th item contains the non-zero coefficients for the i-th lambda value.
- **lambda** Vector containing the lambda values.
- **mu** Intercept.

Author(s)

Quentin Grimonprez, Serge Iovleff

See Also

EMcvlasso

Examples

dataset <- simul(50, 100, 0.4, 1, 10, matrix(c(0.1, 0.9, 0.02, 0.02), nrow = 2))
result <- EMlasso(dataset$data, dataset$response)
# Obtain estimated coefficient in matrix format
coefficient <- listToMatrix(result)
**HDcvlars**

**Description**

cross validation function for lars algorithm

**Usage**

```r
HDcvlars(
  X,
  y,
  nbFolds = 10,
  index = seq(0, 1, by = 0.01),
  mode = c("fraction", "lambda"),
  maxSteps = 3 * min(dim(X)),
  partition = NULL,
  intercept = TRUE,
  eps = .Machine$double.eps^0.5
)
```

**Arguments**

- `X` the matrix (of size n*p) of the covariates.
- `y` a vector of length n with the response.
- `nbFolds` the number of folds for the cross-validation.
- `index` Values at which prediction error should be computed. When mode = "fraction", this is the fraction of the saturated |beta|. The default value is seq(0,1,by=0.01). When mode="lambda", this is values of lambda.
- `mode` Either "fraction" or "lambda". Type of values containing in partition.
- `maxSteps` Maximal number of steps for lars algorithm.
- `partition` partition in nbFolds folds of y. Must be a vector of same size than y containing the index of folds.
- `intercept` If TRUE, there is an intercept in the model.
- `eps` Tolerance of the algorithm.

**Value**

A list containing

- `cv` Mean prediction error for each value of index.
- `cvError` Standard error of cv.
- `minCv` Minimal cv criterion.
- `minIndex` Value of index for which the cv criterion is minimal.
**index** Values at which prediction error should be computed. This is the fraction of the saturated $|\beta|$. The default value is seq(0,1,by=0.01).

**maxSteps** Maximum number of steps of the lars algorithm.

**Author(s)**
Quentin Grimonprez

**Examples**

```r
dataset <- simul(50, 10000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- HDcvlars(dataset$data, dataset$response, 5)
```

**Description**

It performs the lars algorithm for solving a special case of lasso problem. It is a linear regression problem with a $L_1$-penalty on the difference of two successive coefficients.

**Usage**

```r
HDfusion(
  X,
  y,
  maxSteps = 3 * min(dim(X)),
  intercept = TRUE,
  eps = .Machine$double.eps^0.5
)
```

**Arguments**

- **X**: the matrix (of size n*p) of the covariates.
- **y**: a vector of length n with the response.
- **maxSteps**: Maximal number of steps for lars algorithm.
- **intercept**: If TRUE, there is an intercept in the model.
- **eps**: Tolerance of the algorithm.

**Value**

An object of type **LarsPath, LarsPath-class**.

**Author(s)**
Quentin Grimonprez
References


See Also

LarsPath HDlars

Examples

```r
set.seed(10)
dataset <- simul(50, 10000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- HDfusion(dataset$data, dataset$response)
```

HDlars

Lars algorithm

Description

It performs the lars algorithm for solving lasso problem. It is a linear regression problem with a l1-penalty on the estimated coefficient.

Usage

```r
HDlars(
  X,
  y,
  maxSteps = 3 * min(dim(X)),
  intercept = TRUE,
  eps = .Machine$double.eps^0.5
)
```

Arguments

- `X`: the matrix (of size n*p) of the covariates.
- `y`: a vector of length n with the response.
- `maxSteps`: Maximal number of steps for lars algorithm.
- `intercept`: If TRUE, add an intercept to the model.
- `eps`: Tolerance of the algorithm.

Details

The l1 penalty performs variable selection via shrinkage of the estimated coefficient. It depends on a penalty parameter called lambda controlling the amount of regularization. The objective function of lasso is:

\[ ||y - X\beta||_2 + \lambda||\beta||_1 \]
Value
An object of type \texttt{LarsPath}.

Author(s)
Quentin Grimonprez

References

See Also
\texttt{LarsPath HDcvlars listToMatrix}

Examples
```r
dataset <- simul(50, 10000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- HDlars(dataset$data, dataset$response)
# Obtain estimated coefficient in matrix format
coefficient <- listToMatrix(result)
```

---

\textbf{LarsPath-class} \hspace{1cm} \textit{Constructor of LarsPath class}

Description
This class stores the results of lars and fusion algorithms.

Details
\begin{description}
\item[\texttt{nbStep}] Number of steps of the algorithm.
\item[\texttt{variable}] List of vector of size "step+1". The i+1-th item contains the index of non-zero coefficients at the i-th step.
\item[\texttt{coefficient}] List of vector of size "step+1". The i+1-th item contains the non-zero coefficients at the i-th step.
\item[\texttt{ll1norm}] Vector of length "step+1", containing the L1-norm of the coefficients at each step.
\item[\texttt{lambda}] Vector of length "step+1", containing the lambda at each step.
\item[\texttt{dropIndex}] Vector of length "step" containing the index of the dropped variable at the i-th step. 0 means no variable has been dropped at this step.
\item[\texttt{addIndex}] Vector of length "step" containing the index of the added variable at the i-th step. 0 means no variable has been added at this step.
\item[\texttt{mu}] Intercept.
\item[\texttt{meanX}] Mean of columns of X.
\end{description}
**ignored**  A vector containing index of ignored variables during the algorithm.

**p**  Total number of covariates.

**fusion**  If TRUE, results from HDfusion function.

**error**  Error message from lars.

### See Also

HDLars

---

**listToMatrix**  
*List to sparse matrix conversion*

### Description

Create a matrix with all estimated coefficients from the output of HDlars or EMlasso functions.

### Usage

```r
listToMatrix(x, row = c("covariates", "lambda"))
```

### Arguments

- **x**  A LarsPath or EMlasso object
- **row**  if covariates, covariates are in row

### Value

A sparse matrix containing the values of estimated coefficients for all penalty parameter and all covariates

### See Also

HDLars EMlasso
plot-methods

plot methods for LarsPath object

Description

plot the path of the lars algorithm.

Usage

## S4 method for signature 'LarsPath'
plot(
  x,
  sep.line = FALSE,
  abscissa = c("l1norm", "lambda"),
  log.scale = FALSE,
  ...
)

Arguments

x LarsPath object
sep.line If TRUE, print vertical dashed line when a variable is added or dropped in the path
abscissa either "l1norm" or "lambda". If "lambda", regularization parameter is used as abscissa, else l1 norm of the solution is used.
log.scale If TRUE, use logarithm scale on abscissa
...
Other plot arguments

See Also

HDLars LarsPath

plot.HDcvlars

plot cross validation mean square error

Description

plot cross validation mean square error

Usage

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

plotCoefficient

Arguments

  x  
  Output from HDcvlars function.
  ...
    graphical parameters

Author(s)

  Quentin Grimonprez

Examples

  dataset <- simul(50, 10000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
  result <- HDcvlars(dataset$data, dataset$response, 5)
  plot(result)

  plotCoefficient(x, step, ylab = "coefficients", xlab = "variables", ...)

Description

  Plot of the coefficients of a step

Usage

  plotCoefficient(x, step, ylab = "coefficients", xlab = "variables", ...)

Arguments

  x  
  A LarsPath object.
  step  
  The step at which you want to plot the coefficients.
  ylab  
  Name of the y axis.
  xlab  
  Name of the x axis.
  ...  
  Other plot arguments.

See Also

  HDlars LarsPath

Examples

  dataset <- simul(50, 1000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
  result <- HDfusion(dataset$data, dataset$response)
  plotCoefficient(result, result@nbStep) # plot coefficients at the last step
### Usage

```r
## S3 method for class 'LarsPath'
predict(object, Xnew, lambda, mode = c("fraction", "lambda", "norm"), ...)
```

### Arguments

- `object`: a LarsPath object
- `Xnew`: a matrix (of size n*object@p) of covariates.
- `lambda`: If mode = "norm", lambda represents the l1-norm of the coefficients with which we want to predict. If mode = "fraction", lambda represents the ratio (l1-norm of the coefficients with which we want to predict)/(l1-norm maximal of the LarsPath object).
- `mode`: "fraction", "lambda" or "norm".
- `...`: other arguments. Not used.

### Value

The predicted response

### Author(s)

Quentin Grimonprez

### Examples

```r
dataset <- simul(50, 10000, 0.4, 10, 50, matrix(c(0.1, 0.8, 0.02, 0.02), nrow = 2))
result <- HDLars(dataset$data[1:40, ], dataset$response[1:40])
y <- predict(result, dataset$data[41:50, ], 0.3, "fraction")
```
Simulate copy number data for a case-control study.

Usage

```r
simul(n, nbSNP, probCas, nbSeg, meanSegmentSize, prob, alpha = 15)
```

Arguments

- `n`: Number of individuals.
- `nbSNP`: Size of the DNA sequence.
- `probCas`: Probability to be a case individual.
- `nbSeg`: Number of causal segments.
- `meanSegmentSize`: The mean size of an abnormal segment.
- `prob`: A 2*2 matrix containing probabilities:
  - `prob[1,1]`: probability to have an anomaly to a SNP given the person does not have the disease and the SNP is causal.
  - `prob[1,2]`: probability to have an anomaly to a SNP given the person does not have the disease and the SNP is not causal.
  - `prob[2,1]`: probability to have an anomaly to a SNP given the person has the disease and the SNP is causal.
  - `prob[2,2]`: probability to have an anomaly to a SNP given the person has the disease and the SNP is not causal.
- `alpha`: Parameter of the beta(alpha, alpha).

Value

- A list containing:
  - `data`: A matrix of size n*nbSeg, containing values of the copy-number signal.
  - `response`: A vector of size n containing the case/control status.
  - `causalSNP`: A vector of size nbSeg containing the center of causal segments.

Author(s)

Quentin Grimonprez, Serge Iovleff

Examples

```r
data <- simul(50, 10000, 0.4, 10, 150, matrix(c(0.1, 0.8, 0.001, 0.001), nrow = 2))
```
Index

*package
 HDPenReg-package, 2

coef.LarsPath, 3
coeff, 4
computeCoefficients, 4

EMcvfusedlasso, 5, 9
EMcvlasso, 7, 10
EMfusedlasso, 5, 8
EMlasso, 7, 9, 15

HDcvlars, 2, 11, 14
HDFusion, 4, 12
HDlars, 2–4, 13, 15–17
HDPenReg (HDPenReg-package), 2
HDPenReg-package, 2
HDPenReg-package, (HDPenReg-package), 2

LarsPath, 3, 4, 12, 14–17
LarsPath (LarsPath-class), 14
LarsPath-class, 14
listToMatrix, 14, 15

plot.LarsPath-method (plot-methods), 16
plot-methods, 16
plot.HDcvlars, 16
plotCoefficient, 17
predict.LarsPath, 18

simul, 19