# Package ‘hdi’

October 13, 2022

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>High-Dimensional Inference</td>
</tr>
<tr>
<td>Version</td>
<td>0.1-9</td>
</tr>
<tr>
<td>Date</td>
<td>2021-05-27</td>
</tr>
<tr>
<td>Author</td>
<td>Lukas Meier [aut, cre], Ruben Dezeure [aut], Nicolai Meinshausen [aut], Martin Maechler [aut], Peter Buehlmann [aut]</td>
</tr>
<tr>
<td>Maintainer</td>
<td>Lukas Meier <a href="mailto:meier@stat.math.ethz.ch">meier@stat.math.ethz.ch</a></td>
</tr>
<tr>
<td>Description</td>
<td>Implementation of multiple approaches to perform inference in high-dimensional models.</td>
</tr>
<tr>
<td>Depends</td>
<td>scalreg</td>
</tr>
<tr>
<td>DependsNote</td>
<td>scalreg does not correctly import lars etc, so we need to depend on it</td>
</tr>
<tr>
<td>Imports</td>
<td>grDevices, graphics, stats, parallel, MASS, glmnet, linprog</td>
</tr>
<tr>
<td>Suggests</td>
<td>Matrix</td>
</tr>
<tr>
<td>SuggestsNote</td>
<td>for tests only</td>
</tr>
<tr>
<td>Encoding</td>
<td>UTF-8</td>
</tr>
<tr>
<td>License</td>
<td>GPL</td>
</tr>
<tr>
<td>NeedsCompilation</td>
<td>no</td>
</tr>
<tr>
<td>Repository</td>
<td>CRAN</td>
</tr>
<tr>
<td>Date/Publication</td>
<td>2021-05-27 13:10:02 UTC</td>
</tr>
</tbody>
</table>

## R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdi-package</td>
<td>2</td>
</tr>
<tr>
<td>boot.lasso.proj</td>
<td>4</td>
</tr>
<tr>
<td>clusterGroupBound</td>
<td>6</td>
</tr>
<tr>
<td>fdr.adjust</td>
<td>9</td>
</tr>
<tr>
<td>glm.pval</td>
<td>10</td>
</tr>
<tr>
<td>groupBound</td>
<td>11</td>
</tr>
<tr>
<td>hdi</td>
<td>13</td>
</tr>
<tr>
<td>lasso.cv</td>
<td>15</td>
</tr>
</tbody>
</table>
Description

Implementation of multiple approaches to perform inference in high-dimensional models.

Details

The DESCRIPTION file:

Package: hdi
Type: Package
Title: High-Dimensional Inference
Version: 0.1-9
Date: 2021-05-27
Author: Lukas Meier [aut, cre], Ruben Dezeure [aut], Nicolai Meinshausen [aut], Martin Maechler [aut], Peter Buehlmann [aut]
Maintainer: Lukas Meier <meier@stat.math.ethz.ch>
Description: Implementation of multiple approaches to perform inference in high-dimensional models.
Depends: scalreg
DependsNote: scalreg does not correctly import lars etc, so we need to depend on it
Imports: grDevices, graphics, stats, parallel, MASS, glmnet, linprog
Suggests: Matrix
SuggestsNote: for tests only
Encoding: UTF-8
License: GPL

Index of help topics:

- boot.lasso.proj P-values based on the bootstrapped lasso projection method
- clusterGroupBound Hierarchical structure group tests in linear model
- fdr.adjust Function to calculate FDR adjusted p-values
glm.pval Function to calculate p-values for a generalized linear model.
groupBound Lower bound on the l1-norm of groups of regression variables
hdri Function to perform inference in high-dimensional (generalized) linear models
hdri-package
lasso.cv Select Predictors via (10-fold) Cross-Validation of the Lasso
lasso.firstq Determine the first q Predictors in the Lasso Path
lasso.proj P-values based on lasso projection method
lm.ci Function to calculate confidence intervals for ordinary multiple linear regression.
lm.pval Function to calculate p-values for ordinary multiple linear regression.
multi.split Calculate P-values Based on Multi-Splitting Approach
plot.clusterGroupBound Plot output of hierarchical testing of groups of variables
rXb Generate Data Design Matrix X and Coefficient Vector beta
riboflavin Riboflavin data set
ridge.proj P-values based on ridge projection method
stability Function to perform stability selection

Author(s)

Lukas Meier, Ruben Dezeure, Nicolai Meinshausen, Martin Mächler, Peter Bühlmann, Maintainer: Lukas Meier <meier@stat.math.ethz.ch>

References


boot.lasso.proj  

**P-values based on the bootstrapped lasso projection method**

**Description**

Compute p-values based on the lasso projection method, also known as the de-sparsified Lasso, using the bootstrap to approximate the distribution of the estimator.

**Usage**

```r
boot.lasso.proj(x, y, family = "gaussian", standardize = TRUE, 
    multiplecorr.method = "WY", 
    parallel = FALSE, ncores = getOption("mc.cores", 2L), 
    betainit = "cv lasso", sigma = NULL, Z = NULL, verbose = FALSE, 
    return.Z = FALSE, robust = FALSE, 
    B = 1000, boot.shortcut = FALSE, 
    return.bootdist = FALSE, wild = FALSE, 
    gaussian.stub = FALSE)
```

**Arguments**

- `x`  
  Design matrix (without intercept).
- `y`  
  Response vector.
- `family`  
  family
- `standardize`  
  Should design matrix be standardized to unit column standard deviation.
- `multiplecorr.method`  
  Either "WY" or any of `p.adjust.methods`.
- `parallel`  
  Should parallelization be used? (logical)
- `ncores`  
  Number of cores used for parallelization.
- `betainit`  
  Either a numeric vector, corresponding to a sparse estimate of the coefficient vector, or the method to be used for the initial estimation, "scaled lasso" or "cv lasso".
- `sigma`  
  Estimate of the standard deviation of the error term. This estimate needs to be compatible with the initial estimate (see betainit) provided or calculated. Otherwise, results will not be correct.
- `Z`  
  User input, also see `return.Z` below
- `verbose`  
  A boolean to enable reporting on the progress of the computations. (Only prints out information when Z is not provided by the user)
- `return.Z`  
  An option to return the intermediate result which only depends on the design matrix x. This intermediate results can be used when calling the function again and the design matrix is the same as before.
- `robust`  
  Uses a robust variance estimation procedure to be able to deal with model mis-specification.
The number of bootstrap samples to be used.

A boolean to enable the computational shortcut for the bootstrap. If set to true, the lasso is not re-tuned for each bootstrap iteration, but it uses the tuning parameter computed on the original data instead.

return.bootdist
A boolean specifying if one is to return the computed bootstrap distributions to the estimator. (Matrix size: ncol(x)*B) If the multiple testing method was chosen to be WY, the bootstrap distribution computer under the complete null hypothesis is returned as well. This option is required if one wants to compute confidence intervals afterwards.

Perform the wild bootstrap based on N(0,1) distributed random variables

DEVELOPER OPTION. Only enable if you know what you are doing. A boolean to run stub code instead of actually bootstrapping the estimator. It generates a finite sample distribution for each estimate by sampling B samples from N(0,\hat{s.e.}_j^2). (Note: we do not sample from the multivariate gaussian with the covariance matrix. Therefore, no dependencies are modelled at all.) Useful for debugging and for checking if the bootstrap is way off for some reason.

Value

Individual p-values for each parameter.

Multiple testing corrected p-values for each parameter.

\( \hat{\sigma} \) coming from the scaled lasso.

Only different from NULL if the option return.Z is on. This is an intermediate result from the computation which only depends on the design matrix x. These are the residuals of the nodewise regressions.

The number of bootstrap samples used.

If the bootstrap shortcut has been used or not.

What tuning parameter was used for the bootstrap shortcut. NULL if no shortcut was used or if no valid lambda was available to use for the shortcut.

Only different from NULL if the option return.bootdist is on. This is a ncol(x)*B matrix where each row contains the computed centered bootstrap distribution for that estimate.

Only different from NULL if the option return.bootdist is on and if the multiple testing method is WY. This is a ncol(x)*B matrix where each row contains the computed centered bootstrap distribution for that estimate. These bootstrap distributions were computed under the complete null hypothesis (b_1 = ... = b_p = 0).

Author(s)

Ruben Dezeure
References


Examples

```r
x <- matrix(rnorm(100 * 10), nrow = 100, ncol = 10)
y <- x[,1] + x[,2] + rnorm(100)

fit.lasso <- boot.lasso.proj(x, y)
which(fit.lasso$pval.corr < 0.05) # typically: '1' and '2' and no other

## Use the computational shortcut for the bootstrap to speed up computations
fit.lasso.shortcut <- boot.lasso.proj(x, y, boot.shortcut = TRUE)
which(fit.lasso.shortcut$pval.corr < 0.05) # typically: '1' and '2' and no other

## Return the bootstrap distribution as well and compute confidence intervals based on it
fit.lasso.allinfo <- boot.lasso.proj(x, y, return.bootdist = TRUE)
confint(fit.lasso.allinfo, level = 0.95)
confint(fit.lasso.allinfo, parm = 1:3)

## Use the scaled lasso for the initial estimate
fit.lasso.scaled <- boot.lasso.proj(x, y, betainit = "scaled lasso")
which(fit.lasso.scaled$pval.corr < 0.05)

## Use a robust estimate for the standard error
fit.lasso.robust <- boot.lasso.proj(x, y, robust = TRUE)
which(fit.lasso.robust$pval.corr < 0.05)
```

**clusterGroupBound**

Hierarchical structure group tests in linear model

**Description**

Computes confidence intervals for the L1-norm of groups of linear regression coefficients in a hierarchical clustering tree.
Usage

clusterGroupBound(x, y, method = "average",
                 dist = as.dist(1 - abs(cor(x))), alpha = 0.05,
                 eps = 0.1, hcloutput, nsplit = 11,
                 s = min(10, ncol(x) - 1),
                 silent = FALSE, setseed = TRUE, lpSolve = TRUE)

Arguments

x  numeric design matrix of the regression \( n \times p \) with \( p \) columns for \( p \) predictor variables and \( n \) rows corresponding to \( n \) observations.
y  numeric response variable of length \( n \).
method  a character string; the method used for constructing the hierarchical clustering tree (default: "average" for "average linkage") via hclust. Alternatively, you can provide your own hierarchical clustering through the optional argument hcloutput.
dist  a distance matrix can be specified on which the hierarchical clustering will be based (see dist). The default option is that the distance between variables will be calculated as 1 less the absolute correlation matrix. Alternatively, you can provide your own hierarchical clustering through the optional argument hcloutput.
alpha  numeric level in \((0,1)\) at which the test / confidence intervals are to be constructed.
eps  a level of \( \epsilon \)\*\( \alpha \) is used and the values of different splits are aggregated using the \((1-\epsilon)\) quantile. See reference below for more details.
hcloutput  optionally, the value of a hclust() call. If it is provided, the arguments dist and method are ignored.
nsplit  the number of data splits used.
s  the dimensionality of the projection that is used. Lower values lead to faster computation and if \( n > 50 \), then \( s \) is set to 50 if left unspecified, to avoid lengthy computations.
silent  logical enabling progress output.
setseed  a logical; if this is true (recommended), then the same random seeds are used for all groups, which makes the confidence intervals simultaneously valid over all groups of variables tested.
lpSolve  logical; only set it to false if lpSolve() is not working on the current machine: setting it to false will result in much slower computations; only use on small problems.

Value

Returns a list with components

groupNumber  The index of the group tested in the original hierarchical clustering tree
members  A list containing the variables that belong into each testes group
clusterGroupBound

noMembers A vector containing the number of members in each group
lowerBound The lower bound on the l1-norm in each group
position The position on the x-axis of each group (used for plotting)
leftChild Gives the index of the group that corresponds to the left child node in the tested
tree (negative values correspond to leaf nodes)
rightChild Same as leftChild for the right child of each node
isLeaf Logical vector. Is TRUE for a group if it is a leaf node in the tested tree or if both
child nodes have a zero lower bound on their group l1-norm

Author(s)
Nicolai Meinshausen

References
Meinshausen, N. (2015); JRSS B, see groupBound.

See Also
Use groupBound to compute the lower bound for selected groups of variables whereas you use this
clusterGroupBound to test all groups in a hierarchical clustering tree.

Examples

## Create a regression problem with correlated design (n = 10, p = 3):
## a block of size 2 and a block of size 1, within-block correlation is 0.99

set.seed(29)
p <- 3
n <- 10

Sigma <- diag(p)
Sigma[1,2] <- Sigma[2,1] <- 0.99

x <- matrix(rnorm(n * p), nrow = n) %*% chol(Sigma)

## Create response with active variable 1
beta <- rep(0, p)
beta[1] <- 5

y <- as.numeric(x %*% beta + rnorm(n))

out <- clusterGroupBound(x, y, nsplit = 4) ## use larger value for nsplit!

## Plot and print the hierarchical group-test
plot(out)
`fdr.adjust`  

```r
print(out)
out$members
out$lowerBound
```

---

**fdr.adjust**  

Function to calculate FDR adjusted p-values

---

### Description

Calculates FDR adjusted p-values similar to R-function `p.adjust` but *without* adjustment for multiplicity.

### Usage

```r
fdr.adjust(p)
```

### Arguments

- **p**  
  Vector of p-values.

### Details

It is assumed that the p-values are already corrected for multiplicity. P-values with a value of 1 are currently ignored.

### Value

Vector of p-values.

### Author(s)

Lukas Meier

### References


### See Also

- `p.adjust`
glm.pval

Function to calculate p-values for a generalized linear model.

Description
Calculates (classical) p-values for an ordinary generalized linear model in the n > p situation.

Usage
glm.pval(x, y, family = "binomial", verbose = FALSE, ...)

Arguments
- x: Design matrix (without intercept).
- y: Response vector.
- family: As in glm.
- verbose: Logical. Should information be printed out if algorithm did not converge?
- ...: Additional arguments to be passed to glm.

Details
A model with intercept is fitted but the p-value of the intercept is not reported in the output.

Value
Vector of p-values (not including the intercept).

Author(s)
Lukas Meier

See Also
hdi

Examples
## ...
Description

Computes a lower bound that forms a one-sided confidence interval for the group l1-norm of a specified group of regression parameters. It is assumed that errors have a Gaussian distribution with unknown noise level. The underlying vector that inference is made about is the l1-sparsest approximation to the noiseless data.

Usage

```r
groupBound(x, y, group, alpha = 0.05, eps = 0.1, nsplit = 11,
    s = min(10, ncol(x) - 1), setseed = TRUE,
    silent = FALSE, lpSolve = TRUE, parallel = FALSE,
    ncores = getOption("mc.cores", 2L))
```

Arguments

- **x**: numeric design matrix of the regression \( n \times p \) with \( p \) columns for \( p \) predictor variables and \( n \) rows corresponding to \( n \) observations.
- **y**: numeric response variable of length \( n \).
- **group**: either a numeric vector with entries in \( \{1, \ldots, p\} \) or a list with such numeric vectors. If `group` is a numeric vector, this is the group of variables for which a lower bound is computed. If `group` is a list, the lower bound is computed for each group in the list.
- **alpha**: numeric level in \( (0, 1) \) at which the test / confidence interval is computed.
- **eps**: a level of \( \epsilon \cdot \alpha \) is used and the values of different splits are aggregated using the \((1 - \epsilon)\) quantile. See reference below for more details.
- **nsplit**: the number of data splits used.
- **s**: the dimensionality of the projection that is used. Lower values lead to faster computation and if \( n > 50 \), then \( s \) is set to 50 if left unspecified, to avoid lengthy computations.
- **setseed**: a logical; if this is true (recommended), then the same random seeds are used for all groups, which makes the confidence intervals simultaneously valid over all groups of variables tested.
- **silent**: logical enabling progress output.
- **lpSolve**: logical; only set it to false if `lpSolve()` is not working on the current machine: setting it to false will result in much slower computations; only use on small problems.
- **parallel**: should parallelization be used? (logical)
- **ncores**: number of cores used for parallelization.
Details

The data are split since the noise level is unknown. On the first part of the random split, a cross-
validated lasso solution is computed, using the \texttt{glmnet} implementation. This estimator is used as an
initial estimator on the second half of the data. Results at level \(\alpha\) are aggregated over \(n_{\text{split}}\) splits via the median of results at levels \(\alpha/2\).

Value

If \texttt{group} is a single numeric vector, a scalar containing the lower bound for this group of variables is
returned. If \texttt{group} is a list, a numeric vector is returned where each entry corresponds to the group
of variables defined in the same order in \texttt{group}.

Author(s)

Nicolai Meinshausen

References

dimensional regression without assumptions on the design. \textit{Journal of the Royal Statistical Society:}

See Also

Use \texttt{clusterGroupBound} to test all groups in a hierarchical clustering tree.

Examples

```r
## Create a regression problem with correlated design: \(p = 6\), \(n = 50\),
## block size \(B = 3\) and within-block correlation of \(\rho = 0.99\)

p <- 6
n <- 50
B <- 3
rho <- 0.99

ind <- rep(1:ceiling(p / B), each = B)[1:p]
Sigma <- diag(p)

for (ii in unique(ind)){
  id <- which(ind == ii)
  Sigma[id, id] <- rho
}

diag(Sigma) <- 1

x <- matrix(rnorm(n * p), nrow = n) %*% chol(Sigma)

## Create response with active variable 1
beta <- rep(0, p)
beta[1] <- 5

y <- as.numeric(x %*% beta + rnorm(n))
```
## Compute lower bounds:

## Lower bound for the L1-norm of *all* variables 1-6 of the sparsest
## optimal vector
nsplit <- 4 ### to make example run fast (use larger value)
lowerBoundAll <- groupBound(x, y, 1:p, nsplit = nsplit)
cat("lower bound for all variables 1-6: ", lowerBoundAll, 
"
")

## Compute additional lower bounds:
q()
## Lower bounds for variable 1 itself, then group {1,3}, 1-2, 1-3, 2-6,
lowerBound <- groupBound(x, y, list(1, c(1,3), 1:2, 1:3, 2:6),
  nsplit = nsplit)
cat("lower bound for the groups\n	{1}, {1,3}, {1,2}, {1..3}, {2..6}:\n",
  format(formatC(c(lowerBound))), 
"
")

---

**hd**

*Function to perform inference in high-dimensional (generalized) linear models*

---

**Description**

Perform inference in high-dimensional (generalized) linear models using various approaches.

**Usage**

```r
hd(x, y, method = "multi.split", B = NULL, fraction = 0.5,
  model.selector = NULL, EV = NULL, threshold = 0.75,
  gamma = seq(0.05, 0.99, by = 0.01),
  classical.fit = NULL,
  args.model.selector = NULL, args.classical.fit = NULL,
  verbose = FALSE, ...)
```

**Arguments**

- **x** Design matrix (without intercept).
- **y** Response vector.
- **method** Multi-splitting ("multi.split") or stability-selection ("stability").
- **B** Number of sample-splits (for "multi.split") or sub-sample iterations (for "stability"). Default is 50 ("multi.split") or 100 ("stability"). Ignored otherwise.
- **fraction** Fraction of data used at each of the B iterations.
- **model.selector** Function to perform model selection. Default is `lasso.cv` ("multi.split") and `lasso.firstq` ("stability"). Function must have at least two arguments: x (the design matrix) and y (the response vector). Return value is the index vector of selected columns. See `lasso.cv` and `lasso.firstq` for examples. Additional arguments can be passed through args.model.selector.
EV (only for "stability"). Bound(s) for expected number of false positives. Can be a vector.

threshold (only for "stability"). Bound on selection frequency.

gamma (only for "multi.split"). Vector of gamma-values.

classical.fit (only for "multi.split"). Function to calculate (classical) p-values. Default is `lm.pval`. Function must have at least two arguments: x (the design matrix) and y (the response vector). Return value is the vector of p-values. See `lm.pval` for an example. Additional arguments can be passed through `args.classical.fit`.

args.model.selector Named list of further arguments for function `model.selector`.

args.classical.fit Named list of further arguments for function `classical.fit`.

verbose Should information be printed out while computing (logical).

... Other arguments to be passed to the underlying functions.

Value

pval (only for "multi.split"). Vector of p-values.

gamma.min (only for "multi.split"). Value of gamma where minimal p-values was attained.

select (only for "stability"). List with selected predictors for the supplied values of EV.

EV (only for "stability"). Vector of corresponding values of EV.

thresholds (only for "stability"). Used thresholds.

freq (only for "stability"). Vector of selection frequencies.

Author(s)

Lukas Meier

References


See Also

`stability`, `multi.split`

Examples

```r
x <- matrix(rnorm(100 * 200), nrow = 100, ncol = 200)
y <- x[,1] * 2 + x[,2] * 2.5 + rnorm(100)

## Multi-splitting with lasso.firstq as model selector function
fit.multi <- hdi(x, y, method = "multi.split",
```
lasso.cv

```r
model.selector = lasso.firstq,
args.model.selector = list(q = 10))
fit.multi
fit.multi$pval.corr[1:10] ## the first 10 p-values

## Stability selection
fit.stab <- hdi(x, y, method = "stability", EV = 2)
fit.stab
fit.stab$freq[1:10] ## frequency of the first 10 predictors
```

---

**lasso.cv**  
*Select Predictors via (10-fold) Cross-Validation of the Lasso*

**Description**

Performs (n-fold) cross-validation of the lasso (via `cv.glmnet`) and determines the prediction optimal set of parameters.

**Usage**

```r
lasso.cv(x, y,
nfolds = 10,
grouped = nrow(x) > 3*nfolds,
...)
```

**Arguments**

- `x` numeric design matrix (without intercept) of dimension \(n \times p\).
- `y` response vector of length \(n\).
- `nfolds` the number of folds to be used in the cross-validation
- `grouped` corresponds to the grouped argument to `cv.glmnet`. This has a smart default such that glmnet does not give a warning about too small sample size.
- `...` further arguments to be passed to `cv.glmnet`.

**Details**

The function basically only calls `cv.glmnet`, see source code.

**Value**

Vector of selected predictors.

**Author(s)**

Lukas Meier
See Also

hdi which uses lasso.cv() by default; cv.glmnet. An alternative for hdi(): lasso.firstq.

Examples

```r
x <- matrix(rnorm(100 * 1000), nrow = 100, ncol = 1000)
y <- x[,1] * 2 + x[,2] * 2.5 + rnorm(100)
sel <- lasso.cv(x, y)
```

Value

Vector of selected predictors.

Author(s)

Lukas Meier

See Also

hdi; the default choice for hdi(), lasso.cv, glmnet
Examples

```r
x <- matrix(rnorm(100 * 1000), nrow = 100, ncol = 1000)
y <- x[,1] * 2 + x[,2] * 2.5 + rnorm(100)
se1 <- lasso.firstq(x, y, q = 5)
se1 # 5 integers from {1,2, ..., 1000}, including '1' and '2', typically
```

---

**Description**

Compute p-values based on the lasso projection method, also known as the de-sparsified Lasso, using an asymptotic gaussian approximation to the distribution of the estimator.

**Usage**

```r
lasso.proj(x, y, family = "gaussian", standardize = TRUE,
multiplecorr.method = "holm", N = 10000,
parallel = FALSE, ncores = getOption("mc.cores", 2L),
betainit = "cv lasso", sigma = NULL, Z = NULL, verbose = FALSE,
return.Z = FALSE, suppress.grouptesting = FALSE, robust = FALSE,
do.ZnZ = FALSE)
```

**Arguments**

- `x`: Design matrix (without intercept).
- `y`: Response vector.
- `family`: family
- `standardize`: Should design matrix be standardized to unit column standard deviation.
- `multiplecorr.method`: Either "WY" or any of `p.adjust.methods`.
- `N`: Number of empirical samples (only used if `multiplecorr.method` == "WY")
- `parallel`: Should parallelization be used? (logical)
- `ncores`: Number of cores used for parallelization.
- `betainit`: Either a numeric vector, corresponding to a sparse estimate of the coefficient vector, or the method to be used for the initial estimation, "scaled lasso" or "cv lasso".
- `sigma`: Estimate of the standard deviation of the error term. This estimate needs to be compatible with the initial estimate (see `betainit`) provided or calculated. Otherwise, results will not be correct.
- `Z`: user input, also see `return.Z` below
- `verbose`: A boolean to enable reporting on the progress of the computations. (Only prints out information when `Z` is not provided by the user)
return.Z  An option to return the intermediate result which only depends on the design matrix x. This intermediate results can be used when calling the function again and the design matrix is the same as before.

suppress.grouptesting  A boolean to optionally suppress the preparations made for testing groups. This will avoid quite a bit of computation and memory usage. The output will also be smaller.

robust  Uses a robust variance estimation procedure to be able to deal with model misspecification.

do.ZnZ  Use a slightly different way of choosing tuning parameters to compute Z, called Z&Z based on Zhang and Zhang (2014). This choice of tuning parameter results in a slightly higher variance of the estimator. More concretely, it achieves a 25 variance of the estimator (over j=1..ncol(x)) in comparison to tuning with cross-validation.

Value

pval  Individual p-values for each parameter.

pval.corr  Multiple testing corrected p-values for each parameter.

groupTest  Function to perform groupwise tests. Groups are indicated using an index vector with entries in 1,...,p or a list thereof.

clusterGroupTest  Function to perform groupwise tests based on hierarchical clustering. You can either provide a distance matrix and clustering method or the output of hierarchical clustering from the function hclust as for clusterGroupBound. P-values are adjusted for multiple testing.

sigmahat  \( \hat{\sigma} \) coming from the scaled lasso.

Z  Only different from NULL if the option return.Z is on. This is an intermediate result from the computation which only depends on the design matrix x. These are the residuals of the nodewise regressions.

Author(s)

Ruben Dezeure

References


**Examples**

```r
x <- matrix(rnorm(100 * 10), nrow = 100, ncol = 10)
y <- x[, 1] + x[, 2] + rnorm(100)
fit.lasso <- lasso.proj(x, y)
which(fit.lasso$pval.corr < 0.05) # typically: '1' and '2' and no other

## Group-wise testing of the first two coefficients
fit.lasso$groupTest(1:2)

## Compute confidence intervals
confint(fit.lasso, level = 0.95)

## Hierarchical testing using distance matrix based on correlation matrix
out.clust <- fit.lasso$clusterGroupTest()
plot(out.clust)

## Fit the lasso projection method without doing the preparations for group testing (saves time and memory)
fit.lasso.faster <- lasso.proj(x, y, suppress.grouptesting = TRUE)

## Use the scaled lasso for the initial estimate
fit.lasso.scaled <- lasso.proj(x, y, betainit = "scaled lasso")
which(fit.lasso.scaled$pval.corr < 0.05)

## Use a robust estimate for the standard error
fit.lasso.robust <- lasso.proj(x, y, robust = TRUE)
which(fit.lasso.robust$pval.corr < 0.05)

## Perform the Z&Z version of the lasso projection method
fit.lasso <- lasso.proj(x, y, do.ZnZ = TRUE)
which(fit.lasso$pval.corr < 0.05) # typically: '1' and '2' and no other
```

---

**lm.ci**  
*Function to calculate confidence intervals for ordinary multiple linear regression.*

---

**Description**

Calculates (classical) confidence intervals for an ordinary multiple linear regression model in the n > p situation.
Usage

```r
lm.ci(x, y, level = 0.95, ...)
```

Arguments

- **x**: Design matrix (without intercept).
- **y**: Response vector.
- **level**: Coverage level.
- **...**: Additional arguments to be passed to `lm`.

Details

A model with intercept is fitted but the p-value of the intercept is not reported in the output.

Value

Matrix of confidence interval bounds (not including the intercept).

Author(s)

Lukas Meier

See Also

- `hdi`

Examples

```r
x <- matrix(rnorm(100 * 5), nrow = 100, ncol = 5)
y <- x[,1] * 2 + x[,2] * 2.5 + rnorm(100)
ci <- lm.ci(x, y)
ci
```

---

**lm.pval**

*Function to calculate p-values for ordinary multiple linear regression.*

Description

Calculates (classical) p-values for an ordinary multiple linear regression in the $n > p$ situation.

Usage

```r
lm.pval(x, y, exact = TRUE, ...)
```
Arguments

- **x**: Design matrix (without intercept).
- **y**: Response vector.
- **exact**: Logical. TRUE if p-values based on t-distribution should be calculated. FALSE if normal distribution should be used as approximation.
- **...**: Additional arguments to be passed to `lm`.

Details

A model with intercept is fitted but the p-value of the intercept is not reported in the output.

Value

Vector of p-values (not including the intercept).

Author(s)

Lukas Meier

See Also

- hdi

Examples

```r
x <- matrix(rnorm(100 * 5), nrow = 100, ncol = 5)
y <- x[,1] * 2 + x[,2] * 2.5 + rnorm(100)
pval <- lm.pval(x, y)
pval
```

---

**multi.split**

*Calculate P-values Based on Multi-Splitting Approach*

Description

Calculate p-values and confidence intervals based on the multi-splitting approach

Usage

```r
multi.split(x, y, B = 100, fraction = 0.5, ci = TRUE, ci.level = 0.95,
            model.selector = lasso.cv,
            classical.fit = lm.pval, classical.ci = lm.ci,
            parallel = FALSE, ncores = getOption("mc.cores", 2L),
            gamma = seq(ceiling(0.05 * B) / B, 1 - 1 / B, by = 1 / B),
            args.model.selector = NULL, args.classical.fit = NULL,
            args.classical.ci = NULL,
            return.nonaggr = FALSE, return.selmodels = FALSE,
            repeat.max = 20,
            verbose = FALSE)
```
Arguments

- **x** numeric design matrix (without intercept).
- **y** numeric response vector.
- **B** the number of sample-splits, a positive integer.
- **fraction** a number in $(0, 1)$, the fraction of data used at each sample split for the model selection process. The remaining data is used for calculating the p-values.
- **ci** logical indicating if a confidence interval should be calculated for each parameter.
- **ci.level** (if ci is true:) a number in $(0, 1)$, typically close to 1, the desired coverage level of the confidence intervals.
- **model.selector** a function to perform model selection, with default `lasso.cv`. The function must have at least two arguments, `x` (the design matrix) and `y` (the response vector). Return value is the index vector of selected columns. See `lasso.cv` and `lasso.firstq` for an example. Additional arguments can be passed through `args.model.selector`.
- **classical.fit** a function to calculate (classical) p-values. Default is `lm.pval`. The function must have at least two arguments, `x` (the design matrix) and `y` (the response vector), and return the vector of p-values. See `lm.pval` for an example. Additional arguments can be passed through `args.classical.fit`.
- **classical.ci** a function to calculate (classical) confidence intervals. Default is `lm.ci`. The function must have at least 3 arguments, `x` (the design matrix), `y` (the response vector) and `level` (the coverage level), and return the matrix of confidence intervals. See `lm.ci` for an example. Additional arguments can be passed through `args.classical.ci`.
- **parallel** logical indicating if parallelization via `mclapply` should be used.
- **ncores** number of cores used for parallelization as `mc.cores` in `mclapply()`.
- **gamma** vector of gamma-values. In case gamma is a scalar, the value $Q_j$ instead of $P_j$ is being calculated (see reference below).
- **args.model.selector** named list of further arguments for function `model.selector`.
- **args.classical.fit** named list of further arguments for function `classical.fit`.
- **args.classical.ci** named list of further arguments for function `classical.ci`.
- **return.nonaggr** logical indicating if the unadjusted p-values be returned.
- **return.selmodels** logical indicating if the selected models (at each split) should be returned. Necessary for the `clusterGroupTest()` part of the result.
- **repeat.max** positive integer indicating the maximal number of split trials. Should not matter in regular cases, but necessary to prevent infinite loops in borderline cases.
- **verbose** should information be printed out while computing? (logical).
Value

- `pval.corr`: Vector of multiple testing corrected p-values.
- `gamma.min`: Value of gamma where minimal p-values was attained.

`clusterGroupTest`
Function to perform groupwise tests based on hierarchical clustering. You can either provide a distance matrix and clustering method or the output of hierarchical clustering from the function `hclust` as for `clusterGroupBound`. P-values are adjusted for multiple testing.

Author(s)

Lukas Meier, Ruben Dezeure, Jacopo Mandozzi

References


See Also

`lasso.cv`, `lasso.firstq`, `lm.pval`, `lm.ci`.

Examples

```r
n <- 40 # a bit small, to keep example "fast"
p <- 256
x <- matrix(rnorm(n * p), nrow = n, ncol = p)
y <- x[,1] * 2 + x[,2] * 2.5 + rnorm(n)

## Multi-splitting with lasso.firstq as model selector function
## 'q' must be specified
fit.multi <- multi.split(x, y, model.selector = lasso.firstq,
                         args.model.selector = list(q = 10))

fit.multi
head(fit.multi$pval.corr, 10) ## the first 10 p-values
ci. <- confint(fit.multi)
head(ci.) # the first 6
stopifnot(all.equal(ci.,
                  with(fit.multi, cbind(lci, uci)), check.attributes=FALSE))

## Use default 'lasso.cv' (slower!!) -- incl cluster group testing:
system.time(fit.m2 <- multi.split(x, y, return.selmodels = TRUE)) # 9 sec (on "i7")
head(fit.m2$pval.corr) ## the first 6 p-values
head(confint(fit.m2)) # the first 6 95% conf.intervals

## Now do clustergroup testing
```
plot.clusterGroupBound

Plot output of hierarchical testing of groups of variables

Description
The `plot()` method for "clusterGroupBound" objects plots the outcome of applying a lower bound on the L1-norm on groups of variables in a hierarchical clustering tree.

Usage
```r
## S3 method for class 'clusterGroupBound'
plot(x, cexfactor = 1, yaxis = "members",
     xlab = "", col = NULL, pch = 20, ...)
```

Arguments
- `x`: an object of class "clusterGroupBound", as resulting from `clusterGroupBound()`.
- `cexfactor`: numeric expansion factor for the size of the node symbols.
- `yaxis`: a string: for the default "members", the hierarchical tree is shown as function of cluster size on the y-axis, whereas the node sizes are proportional to the lower L1-norm of the respective groups of variables. If `yaxis` takes any different value, then this is reversed and the tree is shown against the lower L1-norm on the y-axis, while node sizes are now proportional to the number of elements in each cluster.
- `xlab`: label used for the x-axis; by default none.
- `col`: the colour of the symbols for the nodes.
- `pch`: the plot symbol (see `points`) of the symbols for the nodes.
- `...`: optional additional arguments passed to `plot.default`.

Value
Nothing is returned
### riboflavin

**Author(s)**

Nicolai Meinshausen <meinshausen@stat.math.ethz.ch>

**See Also**

Use `clusterGroupBound()` to test all groups in a hierarchical clustering tree. Use `groupBound()` to compute the lower bound for selected groups of variables.

**Examples**

```r
## Create a regression problem with correlated design (n = 10, p = 3):
## a block of size 2 and a block of size 1, within-block correlation is 0.99

set.seed(29)
p <- 3
n <- 10

Sigma <- diag(p)
Sigma[1,2] <- Sigma[2,1] <- 0.99
x <- matrix(rnorm(n * p), nrow = n) %*% chol(Sigma)

## Create response with active variable 1
beta <- rep(0, p)
beta[1] <- 5
y <- as.numeric(x %*% beta + rnorm(n))

## Compute the lower bound for all groups in a hierarchical clustering tree
## use larger value for nsplit!
cgb5 <- clusterGroupBound(x, y, nsplit = 4)

## Plot the tree with y-axis proportional to the (log) of the number of
## group members and node sizes proportional to the lower l1-norm bound.
plot(cgb5)

## Show the lower bound on the y-axis and node sizes proportional to
## number of group members
plot(cgb5, yaxis = "")
```

---

**riboflavin**

**Riboflavin data set**

**Description**

Dataset of riboflavin production by Bacillus subtilis containing \( n = 71 \) observations of \( p = 4088 \) predictors (gene expressions) and a one-dimensional response (riboflavin production).
Usage
data(riboflavin)

Format

y  Log-transformed riboflavin production rate (original name: q_RIBFLV).
x  (Co-)variables measuring the logarithm of the expression level of 4088 genes.

Details

Data kindly provided by DSM (Switzerland).

References


Examples

data(riboflavin)

---------
ridge.proj  P-values based on ridge projection method
---------

Description

Compute p-values for lasso-type regression coefficients based on the ridge projection method.

Usage

ridge.proj(x, y, family = "gaussian", standardize = TRUE,
lambda = 1, betainit = "cv lasso", sigma = NULL,
suppress.grouptesting = FALSE,
multiplecorr.method = "holm", N = 10000)

Arguments

x  design matrix (without intercept).
y  response vector.
family family
standardize  Should design matrix be standardized to unit column standard deviation (logical)?
lambda  Value of penalty parameter lambda (ridge regression).
betainit Either a numeric vector, corresponding to a sparse estimate of the coefficient vector, or the method to be used for the initial estimation, "scaled lasso" or "cv lasso".
Estimate of the standard deviation of the error term. This estimate needs to be compatible with the initial estimate (see betainit) provided or calculated. Otherwise, results won’t be correct.

A boolean to optionally suppress the preparations made for testing groups. This will avoid quite a bit of computation and memory usage. The output will also be smaller.

Either "WY" or any of p.adjust.methods.

number of empirical samples (only used if multiplecorr.method = "WY").

Individual p-values for each parameter.

Multiple testing corrected p-values for each parameter.

Function to perform groupwise tests. Groups are indicated using an index vector with entries in 1,...,p or a list thereof.

Function to perform groupwise tests based on hierarchical clustering. You can either provide a distance matrix and clustering method or the output of hierarchical clustering from the function hclust as for clusterGroupBound. P-values are adjusted for multiple testing.

σ coming from the scaled lasso.

Peter Buehlmann, Ruben Dezeure, Lukas Meier


x <- matrix(rnorm(100 * 100), nrow = 100, ncol = 100)
y <- x[,1] + x[,2] + rnorm(100)
fit.ridge <- ridge.proj(x, y)
which(fit.ridge$pval.corr < 0.05)

## Use the scaled lasso for the initial estimate
fit.ridge.scaled <- ridge.proj(x, y, betainit = "scaled lasso")
which(fit.ridge.scaled$pval.corr < 0.05)

## Group-wise testing of the first two coefficients
fit.ridge$groupTest(1:2)

## Hierarchical testing using distance matrix based on
## Generate Data Design Matrix and Coefficient Vector β

**Description**

Generate a random design matrix \( X \) and coefficient vector \( \beta \) useful for simulations of (high dimensional) linear models. In particular, the function `rXb()` can be used to exactly recreate the reference linear model datasets of the hdi paper.

**Usage**

```r
rXb(n, p, s0, 
    xtype = c("toeplitz", "exp.decay", "equi.corr"),
    btype = "U[-2,2]",
    permuted = FALSE, iteration = NA, do2S = TRUE,
    x.par = switch(xtype,
        "toeplitz" = 0.9,
        "equi.corr" = 0.8,
        "exp.decay" = c(0.4, 5)),
    verbose = TRUE)
```

```r
rX(n, p, xtype, permuted, do2S = TRUE,
    par = switch(xtype,
        "toeplitz" = 0.9,
        "equi.corr" = 0.8,
        "exp.decay" = c(0.4, 5)))
```

**Arguments**

- **n**: integer; the sample size \( n \) (paper had always \( n = 100 \)).
- **p**: integer; the number of coefficients in the linear model. (paper had always \( p = 500 \)).
- **s0**: integer number of nonzero coefficients desired in the model; hence at most \( p \).
- **xtype**: a character string specifying the type of design matrix one wants to generate. Must be one of "toeplitz", "equi.corr" or "exp.decay".
- **btype**: a character string specifying the type of nonzero coefficients ("beta") one wants to generate. In the hdi paper, this has been one of "U[-2,2]", "U[0,2]", "U[0,4]", "bfix1", "bfix2" and "bfix10". In general, any string of the form "U[a,b]" or "bfix<c>" is allowed, where \( a, b \) and \(<c>\) must be numbers (with \( a \leq b \)).
permuted  logical specifying if the columns of the design matrix should be permuted.
iteration  integer or NA specifying if seeds should be set to generate reproducible realizations of the design type and coefficients type. NA corresponds to not setting seeds. Iteration numbers 1 to 50 correspond to the setups from the paper. If a seed is set, the original .Random.seed at the point of entering the function is saved and is restored upon exit of the data generation. If NA, the current .Random.seed is taken as usual in R.
do2S    logical indicating if in the case of xtypes "toeplitz" or "equi.corr", the \( p \times p \) covariance matrix should be inverted twice. Must be true, to regenerate the \( X \) matrices from the hdi paper exactly "to the last bit".
x.par, par  the parameters to be used for the design matrix. Must be a numeric vector of length one or two. The default uses the parameters also used in the hdi paper.
verbose should the function give a message if seeds are being set? (logical).

Details

**Generation of the design matrix \( X \):**

For all xtype's, the \( X \) matrix will be multivariate normal, with mean zero and (co)variance matrix \( \Sigma = C \) determined from xtype, x.par and \( p \) as follows:

- **xtype = "toeplitz":** \( C \leftarrow \text{par} ^ \text{abs(toeplitz(0:(p-1)))} \)
- **xtype = "equi.corr":** \( \Sigma_{i,j} = \text{par} \) for \( i \neq j \), and = 1 for \( i = j \), i.e., on the diagonal.
- **xtype = "exp.decay":** \( C \leftarrow \text{solve}((\text{par}[1] ^ \text{abs(toeplitz(0:(p-1)))} / \text{par}[2])) \)

Value

For \( rXb() \): A list with components

- **x**  the generated \( n \times p \) design matrix \( X \).
- **beta**  the generated coefficient vector \( \beta \) ('beta').

For \( rX() \): the generated \( n \times p \) design matrix \( X \).

Author(s)

Ruben Dezeure <dezeure@stat.math.ethz.ch>

References


Examples

```r
## Generate the first realization of the linear model with design matrix
## type Toeplitz and coefficients type uniform between -2 and 2

dset <- rXb(n = 80, p = 20, s0 = 3,
           xtype = "toeplitz", btype = "U[-2,2]")
x <- dset$x
```
beta <- dset$beta

## generate 100 response vectors of this linear model
y <- as.vector(x %*% beta) + replicate(100, rnorm(nrow(x)))

## Use 'beta_min' fulfilling beta's (non standard 'btype'):
str(ds2 <- rXb(n = 50, p = 12, s0 = 3,
    xtype = "exp.decay", btype = "U[0.1, 5]"))

## Generate a design matrix of type "toeplitz"
set.seed(3) # making it reproducible
X3 <- rX(n = 800, p = 500, xtype = "toeplitz", permuted = FALSE)

## permute the columns
set.seed(3)
Xp <- rX(n = 800, p = 500, xtype = "toeplitz", permuted = TRUE)

---

### stability

**Function to perform stability selection**

**Description**

Function to perform stability selection

**Usage**

```r
stability(x, y, EV, threshold = 0.75, B = 100, fraction = 0.5,
    model.selector = lasso.firstq, args.model.selector = NULL,
    parallel = FALSE, ncores = getOption(“mc.cores”, 2L),
    verbose = FALSE)
```

**Arguments**

- `x` Design matrix (without intercept).
- `y` Response vector.
- `EV` Bound for expected number of false positives.
- `threshold` Threshold for selection frequency. Must be in (0.5, 1).
- `B` Number of sub-sample iterations.
- `fraction` Fraction of data used at each of the B sub-samples.
- `model.selector` Function to perform model selection. Default is `lasso.firstq`. User supplied function must have at least three arguments: `x` (the design matrix), `y` (the response vector) and `q` (the maximal model size). Return value is the index vector of selected columns. See `lasso.firstq` for an example. Additional arguments can be passed through `args.model.selector`.
- `args.model.selector` Named list of further arguments for function `model.selector`. 
parallel Should parallelization be used? (logical)
ncores Number of cores used for parallelization.
verbose Should information be printed out while computing (logical).

Value
selected Vector of selected predictors.
freq Vector of selection frequencies.
q Size of fitted models in order to control error rate at desired level.

Author(s)
Lukas Meier

References

Examples
```r
x <- matrix(rnorm(100 * 1000), nrow = 100, ncol = 1000)
y <- x[,1] * 2 + x[,2] * 2.5 + rnorm(100)
fit.stab <- stability(x, y, EV = 1)
fit.stab
fit.stab$freq[1:10] ## selection frequency of the first 10 predictors
```
Index

* confidence intervals
  clusterGroupBound, 6
  groupBound, 11
  plot.clusterGroupBound, 24
* datagen
  rXb, 28
* datasets
  riboflavin, 25
* generate datasets
  rXb, 28
* hierarchical clustering
  clusterGroupBound, 6
  plot.clusterGroupBound, 24
* htest
  plot.clusterGroupBound, 24
* models
  boot.lasso.proj, 4
  fdr.adjust, 9
  glm.pval, 10
  hdi, 13
  lasso.cv, 15
  lasso.firstq, 16
  lasso.proj, 17
  lm.ci, 19
  lm.pval, 20
  multi.split, 21
  ridge.proj, 26
  rXb, 28
  stability, 30
  .Random.seed, 29
  boot.lasso.proj, 4
  character, 7, 28
  class, 24
  clusterGroupBound, 6, 12, 18, 23–25, 27
  cv.glmnet, 15, 16
  dist, 7
  fdr.adjust, 9
  function, 22
  glm, 10
  glm.pval, 10
  glmnet, 16
  groupBound, 8, 11, 25
  hclust, 7, 18, 23, 27
  hdi, 10, 13, 16, 20, 21
  hdi-package, 2
  lasso.cv, 13, 15, 16, 22, 23
  lasso.firstq, 13, 16, 16, 22, 23, 30
  lasso.proj, 17
  list, 11, 22, 29
  lm, 20, 21
  lm.ci, 19, 22, 23
  lm.pval, 14, 20, 22, 23
  logical, 22
  mclapply, 22
multi.split, 14, 21
p.adjust, 9
p.adjust.methods, 4, 17, 27
plot, 24
plot.clusterGroupBound, 24
plot.default, 24
points, 24
riboflavin, 25
ridge.proj, 26
rX (rXb), 28
rXb, 28

stability, 14, 30