Package 'robustHD'

October 13, 2021

Type Package

Title Robust Methods for High-Dimensional Data

Version 0.7.1

Date 2021-10-13

Depends R (>= 3.5.0), ggplot2 (>= 0.9.2), perry (>= 0.3.0), robustbase (>= 0.9-5)

Imports MASS, Rcpp (>= 0.9.10), grDevices, parallel, stats, utils

LinkingTo Rcpp (>= 0.9.10), RcppArmadillo (>= 0.3.0)

Suggests lars, mvtnorm, testthat

Description Robust methods for high-dimensional data, in particular linear model selection techniques based on least angle regression and sparse regression.

License GPL (>= 2)

LazyLoad yes

Author Andreas Alfons [aut, cre]

Maintainer Andreas Alfons <alfons@ese.eur.nl>

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-10-13 09:30:02 UTC

R topics documented:

`robustHD-package` *Robust Methods for High-Dimensional Data*

Description

Robust methods for high-dimensional data, in particular linear model selection techniques based on least angle regression and sparse regression.

Details

The DESCRIPTION file:

Package:	robustHD
Type:	Package
Title:	Robust Methods for High-Dimensional Data
Version:	0.7.1
Date:	2021-10-13
Depends:	R (>= 3.5.0), ggplot2 (>= 0.9.2), perry (>= 0.3.0), robustbase (>= 0.9-5)
Imports:	MASS, Rcpp (>= 0.9.10), grDevices, parallel, stats, utils
LinkingTo:	Rcpp (>= 0.9.10), RcppArmadillo (>= 0.3.0)
Suggests:	lars, mvtnorm, testthat

Description:	Robust methods for high-dimensional data, in particular linear model selection techniques based on least ang
License:	GPL (>= 2)
LazyLoad:	yes
Authors@R:	person("Andreas", "Alfons", email = "alfons@ese.eur.nl", role = c("aut", "cre"))
Author:	Andreas Alfons [aut, cre]
Maintainer:	Andreas Alfons <alfons@ese.eur.nl>
Encoding:	UTF-8
RoxygenNote:	7.1.2

Index of help topics:

sparseLTS	Sparse least trimmed squares regression
standardize	Data standardization
tsBlocks	Construct predictor blocks for time series models
tslars	(Robust) least angle regression for time series data
tslarsP	(Robust) least angle regression for time series data with fixed lag length
weights.sparseLTS	Extract outlier weights from sparse LTS regression models
winsorize	Data cleaning by winsorization

Author(s)

Andreas Alfons [aut, cre]

Maintainer: Andreas Alfons <alfons@ese.eur.nl>

`AIC.seqModel` *Information criteria for a sequence of regression models*

Description

Compute the Akaike or Bayes information criterion for for a sequence of regression models, such as submodels along a robust least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
## S3 method for class 'seqModel'
AIC(object, ..., k = 2)

## S3 method for class 'sparseLTS'
AIC(object, ..., fit = c("reweighted", "raw", "both"), k = 2)

## S3 method for class 'seqModel'
BIC(object, ...)

## S3 method for class 'sparseLTS'
BIC(object, ...)
```

Arguments

`object`	the model fit for which to compute the information criterion.
`...`	for the `BIC` method, additional arguments to be passed down to the `AIC` method. For the `AIC` method, additional arguments are currently ignored.
`k`	a numeric value giving the penalty per parameter to be used. The default is to use 2 as in the classical definition of the AIC.

`fit` a character string specifying for which fit to compute the information criterion. Possible values are `"reweighted"` (the default) for the information criterion of the reweighted fit, `"raw"` for the information criterion of the raw fit, or `"both"` for the information criteria of both fits.

Details

The information criteria are computed as $n(\log(2\pi)+1+\log(\hat{\sigma}^2))+df\,k$, where n denotes the number of observations, $\hat{\sigma}$ is the robust residual scale estimate, df is the number of nonzero coefficient estimates, and k is penalty per parameter. The usual definition of the AIC uses $k = 2$, whereas the BIC uses $k = \log(n)$. Consequently, the former is used as the default penalty of the `AIC` method, whereas the `BIC` method calls the `AIC` method with the latter penalty.

Value

A numeric vector or matrix giving the information criteria for the requested model fits.

Note

Computing information criteria for several objects supplied via the `...` argument (as for the default methods of `AIC` and `BIC`) is currently not implemented.

Author(s)

Andreas Alfons

References

Akaike, H. (1970) Statistical predictor identification. *Annals of the Institute of Statistical Mathematics*, **22**(2), 203–217.

Schwarz, G. (1978) Estimating the dimension of a model. *The Annals of Statistics*, **6**(2), 461–464.

See Also

`AIC`, `rlars`, `sparseLTS`

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
```

```
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# compute AIC and BIC
AIC(fitRlars)
BIC(fitRlars)


## fit sparse LTS model over a grid of values for lambda
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# compute AIC and BIC
AIC(fitSparseLTS)
BIC(fitSparseLTS)
```

`coef.seqModel` *Extract coefficients from a sequence of regression models*

Description

Extract coefficients from a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
## S3 method for class 'seqModel'
coef(object, s = NA, zeros = TRUE, drop = !is.null(s), ...)

## S3 method for class 'tslars'
coef(object, p, ...)

## S3 method for class 'perrySeqModel'
coef(object, ...)

## S3 method for class 'sparseLTS'
coef(
  object,
  s = NA,
  fit = c("reweighted", "raw", "both"),
  zeros = TRUE,
  drop = !is.null(s),
  ...
)
```

Arguments

object	the model fit from which to extract coefficients.
s	for the "seqModel" method, an integer vector giving the steps of the submodels for which to extract coefficients (the default is to use the optimal submodel). For the "sparseLTS" method, an integer vector giving the indices of the models for which to extract coefficients. If fit is "both", this can be a list with two components, with the first component giving the indices of the reweighted fits and the second the indices of the raw fits. The default is to use the optimal model for each of the requested estimators. Note that the optimal models may not correspond to the same value of the penalty parameter for the reweighted and the raw estimator.
zeros	a logical indicating whether to keep zero coefficients (TRUE, the default) or to omit them (FALSE).
drop	a logical indicating whether to reduce the dimension to a vector in case of only one submodel.
...	for the "tslars" method, additional arguments to be passed down to the "seqModel" method. For the other methods, additional arguments are currently ignored.
p	an integer giving the lag length for which to extract coefficients (the default is to use the optimal lag length).
fit	a character string specifying which coefficients to extract. Possible values are "reweighted" (the default) for the coefficients from the reweighted estimator, "raw" for the coefficients from the raw estimator, or "both" for the coefficients from both estimators.

Value

A numeric vector or matrix containing the requested regression coefficients.

Author(s)

Andreas Alfons

See Also

coef, rlars, grplars, rgrplars, tslarsP, rtslarsP, tslars, rtslars, sparseLTS

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
```

```
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# extract coefficients
coef(fitRlars, zeros = FALSE)
coef(fitRlars, s = 1:5, zeros = FALSE)


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# extract coefficients
coef(fitSparseLTS, zeros = FALSE)
coef(fitSparseLTS, fit = "both", zeros = FALSE)
coef(fitSparseLTS, s = NULL, zeros = FALSE)
coef(fitSparseLTS, fit = "both", s = NULL, zeros = FALSE)
```

---

coefPlot *Coefficient plot of a sequence of regression models*

Description

Produce a plot of the coefficients from a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
coefPlot(object, ...)

## S3 method for class 'seqModel'
coefPlot(object, zeros = FALSE, labels = NULL, ...)

## S3 method for class 'tslars'
coefPlot(object, p, zeros = FALSE, labels = NULL, ...)

## S3 method for class 'sparseLTS'
coefPlot(
  object,
  fit = c("reweighted", "raw", "both"),
```

```
  zeros = FALSE,
  labels = NULL,
  ...
)

## S3 method for class 'setupCoefPlot'
coefPlot(
  object,
  abscissa = NULL,
  size = c(0.5, 2, 4),
  offset = 1,
  facets = object$facets,
  ...
)
```

Arguments

object	the model fit to be plotted.
...	additional arguments to be passed down, eventually to geom_line and geom_point.
zeros	a logical indicating whether predictors that never enter the model and thus have zero coefficients should be included in the plot (TRUE) or omitted (FALSE, the default). This is useful if the number of predictors is much larger than the number of observations, in which case many coefficients are never nonzero.
labels	an optional character vector containing labels for the predictors. Plotting labels can be suppressed by setting this to NA.
p	an integer giving the lag length for which to produce the plot (the default is to use the optimal lag length).
fit	a character string specifying for which estimator to produce the plot. Possible values are "reweighted" (the default) for the reweighted fits, "raw" for the raw fits, or "both" for both estimators.
abscissa	a character string specifying what to plot on the x-axis. For objects inheriting from class "seqModel", possible values are "step" for the step number (the default), or "df" for the degrees of freedom. For code"sparseLTS" objects, possible values are code"lambda" for the value of the penalty parameter (the default), or "step" for the step number.
size	a numeric vector of length three giving the line width, the point size and the label size, respectively.
offset	an integer giving the offset of the labels from the corresponding coefficient values from the last step (i.e., the number of blank characters to be prepended to the label).
facets	a faceting formula to override the default behavior. If supplied, facet_wrap or facet_grid is called depending on whether the formula is one-sided or two-sided.

Value

An object of class "ggplot" (see ggplot).

Author(s)

Andreas Alfons

See Also

ggplot, rlars, grplars, rgrplars, tslarsP, rtslarsP, tslars, rtslars, sparseLTS

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# create plot
coefPlot(fitRlars)


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# create plot
coefPlot(fitSparseLTS)
coefPlot(fitSparseLTS, fit = "both")
```

---

corHuber *Robust correlation based on winsorization*

Description

Compute a robust correlation estimate based on winsorization, i.e., by shrinking outlying observations to the border of the main part of the data.

Usage

```
corHuber(
  x,
  y,
  type = c("bivariate", "adjusted", "univariate"),
  standardized = FALSE,
  centerFun = median,
  scaleFun = mad,
  const = 2,
  prob = 0.95,
  tol = .Machine$double.eps^0.5,
  ...
)
```

Arguments

x	a numeric vector.
y	a numeric vector.
type	a character string specifying the type of winsorization to be used. Possible values are "univariate" for univariate winsorization, "adjusted" for adjusted univariate winsorization, or "bivariate" for bivariate winsorization.
standardized	a logical indicating whether the data are already robustly standardized.
centerFun	a function to compute a robust estimate for the center to be used for robust standardization (defaults to median). Ignored if standardized is TRUE.
scaleFun	a function to compute a robust estimate for the scale to be used for robust standardization (defaults to mad). Ignored if standardized is TRUE.
const	numeric; tuning constant to be used in univariate or adjusted univariate winsorization (defaults to 2).
prob	numeric; probability for the quantile of the χ^2 distribution to be used in bivariate winsorization (defaults to 0.95).
tol	a small positive numeric value. This is used in bivariate winsorization to determine whether the initial estimate from adjusted univariate winsorization is close to 1 in absolute value. In this case, bivariate winsorization would fail since the points form almost a straight line, and the initial estimate is returned.
...	additional arguments to be passed to robStandardize.

Details

The borders of the main part of the data are defined on the scale of the robustly standardized data. In univariate winsorization, the borders for each variable are given by $+/-$const, thus a symmetric distribution is assumed. In adjusted univariate winsorization, the borders for the two diagonally opposing quadrants containing the minority of the data are shrunken by a factor that depends on the ratio between the number of observations in the major and minor quadrants. It is thus possible to better account for the bivariate structure of the data while maintaining fast computation. In bivariate winsorization, a bivariate normal distribution is assumed and the data are shrunken towards the boundary of a tolerance ellipse with coverage probability prob. The boundary of this ellipse is

thereby given by all points that have a squared Mahalanobis distance equal to the quantile of the χ^2 distribution given by prob. Furthermore, the initial correlation matrix required for the Mahalanobis distances is computed based on adjusted univariate winsorization.

Value

The robust correlation estimate.

Author(s)

Andreas Alfons, based on code by Jafar A. Khan, Stefan Van Aelst and Ruben H. Zamar

References

Khan, J.A., Van Aelst, S. and Zamar, R.H. (2007) Robust linear model selection based on least angle regression. *Journal of the American Statistical Association*, **102**(480), 1289–1299.

See Also

winsorize

Examples

```
## generate data
library("mvtnorm")
set.seed(1234)  # for reproducibility
Sigma <- matrix(c(1, 0.6, 0.6, 1), 2, 2)
xy <- rmvnorm(100, sigma=Sigma)
x <- xy[, 1]
y <- xy[, 2]

## introduce outlier
x[1] <- x[1] * 10
y[1] <- y[1] * (-5)

## compute correlation
cor(x, y)
corHuber(x, y)
```

---

critPlot *Optimality criterion plot of a sequence of regression models*

Description

Produce a plot of the values of the optimality criterion for a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
critPlot(object, ...)

## S3 method for class 'seqModel'
critPlot(object, which = c("line", "dot"), ...)

## S3 method for class 'tslars'
critPlot(object, p, which = c("line", "dot"), ...)

## S3 method for class 'sparseLTS'
critPlot(
  object,
  which = c("line", "dot"),
  fit = c("reweighted", "raw", "both"),
  ...
)

## S3 method for class 'perrySeqModel'
critPlot(object, which = c("line", "dot", "box", "density"), ...)

## S3 method for class 'perrySparseLTS'
critPlot(
  object,
  which = c("line", "dot", "box", "density"),
  fit = c("reweighted", "raw", "both"),
  ...
)

## S3 method for class 'setupCritPlot'
critPlot(object, ...)
```

Arguments

object	the model fit to be plotted, , or an object containing all necessary information for plotting (as generated by setupCritPlot).
...	additional arguments to be passed down, eventually to geom_line, geom_pointrange, geom_boxplot, or geom_density.
which	a character string specifying the type of plot. Possible values are "line" (the default) to plot the (average) results for each model as a connected line, "dot" to create a dot plot, "box" to create a box plot, or "density" to create a smooth density plot. Note that the last two plots are only available in case of prediction error estimation via repeated resampling.
p	an integer giving the lag length for which to produce the plot (the default is to use the optimal lag length).
fit	a character string specifying for which estimator to produce the plot. Possible values are "reweighted" (the default) for the reweighted fits, "raw" for the raw fits, or "both" for both estimators.

Value

An object of class "ggplot" (see ggplot).

Note

Function perryPlot is used to create the plot, even if the optimality criterion does not correspond to resampling-based p rediction error estimation. While this can be seen as as a misuse of its functionality, it ensures that all optimality criteria are displayed in the same way.

Author(s)

Andreas Alfons

See Also

ggplot, perryPlot, rlars, grplars, rgrplars, tslarsP, rtslarsP, tslars, rtslars, sparseLTS

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# create plot
critPlot(fitRlars)


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# create plot
critPlot(fitSparseLTS)
critPlot(fitSparseLTS, fit = "both")
```

diagnosticPlot *Diagnostic plots for a sequence of regression models*

Description

Produce diagnostic plots for a sequence of regression models, such as submodels along a robust least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter. Four plots are currently implemented.

Usage

```
diagnosticPlot(object, ...)

## S3 method for class 'seqModel'
diagnosticPlot(object, s = NA, covArgs = list(), ...)

## S3 method for class 'perrySeqModel'
diagnosticPlot(object, covArgs = list(), ...)

## S3 method for class 'tslars'
diagnosticPlot(object, p, s = NA, covArgs = list(), ...)

## S3 method for class 'sparseLTS'
diagnosticPlot(
  object,
  s = NA,
  fit = c("reweighted", "raw", "both"),
  covArgs = list(),
  ...
)

## S3 method for class 'perrySparseLTS'
diagnosticPlot(
  object,
  fit = c("reweighted", "raw", "both"),
  covArgs = list(),
  ...
)

## S3 method for class 'setupDiagnosticPlot'
diagnosticPlot(
  object,
  which = c("all", "rqq", "rindex", "rfit", "rdiag"),
  ask = (which == "all"),
  facets = object$facets,
  size = c(2, 4),
  id.n = NULL,
```

```
  ...
)
```

Arguments

object	the model fit for which to produce diagnostic plots, or an object containing all necessary information for plotting (as generated by setupDiagnosticPlot).
...	additional arguments to be passed down, eventually to geom_point.
s	for the "seqModel" method, an integer vector giving the steps of the submodels for which to produce diagnostic plots (the default is to use the optimal submodel). For the "sparseLTS" method, an integer vector giving the indices of the models for which to produce diagnostic plots (the default is to use the optimal model for each of the requested fits).
covArgs	a list of arguments to be passed to covMcd for the regression diagnostic plot (see
p	an integer giving the lag length for which to produce the plot (the default is to use the optimal lag length).
fit	a character string specifying for which fit to produce diagnostic plots. Possible values are "reweighted" (the default) for diagnostic plots for the reweighted fit, "raw" for diagnostic plots for the raw fit, or "both" for diagnostic plots for both fits. "Details").
which	a character string indicating which plot to show. Possible values are "all" (the default) for all of the following, "rqq" for a normal Q-Q plot of the standardized residuals, "rindex" for a plot of the standardized residuals versus their index, "rfit" for a plot of the standardized residuals versus the fitted values, or "rdiag" for a regression diagnostic plot (standardized residuals versus robust Mahalanobis distances of the predictor variables).
ask	a logical indicating whether the user should be asked before each plot (see devAskNewPage). The default is to ask if all plots are requested and not ask otherwise.
facets	a faceting formula to override the default behavior. If supplied, facet_wrap or facet_grid is called depending on whether the formula is one-sided or two-sided.
size	a numeric vector of length two giving the point and label size, respectively.
id.n	an integer giving the number of the most extreme observations to be identified by a label. The default is to use the number of identified outliers, which can be different for the different plots. See "Details" for more information.

Details

In the normal Q-Q plot of the standardized residuals, a reference line is drawn through the first and third quartile. The id.n observations with the largest distances from that line are identified by a label (the observation number). The default for id.n is the number of regression outliers, i.e., the number of observations whose residuals are too large (cf. weights).

In the plots of the standardized residuals versus their index or the fitted values, horizontal reference lines are drawn at 0 and +/-2.5. The id.n observations with the largest absolute values of the standardized residuals are identified by a label (the observation number). The default for id.n is

the number of regression outliers, i.e., the number of observations whose absolute residuals are too large (cf. weights).

For the regression diagnostic plot, the robust Mahalanobis distances of the predictor variables are computed via the MCD based on only those predictors with non-zero coefficients (see covMcd). Horizontal reference lines are drawn at +/-2.5 and a vertical reference line is drawn at the upper 97.5% quantile of the χ^2 distribution with p degrees of freedom, where p denotes the number of predictors with non-zero coefficients. The id.n observations with the largest absolute values of the standardized residuals and/or largest robust Mahalanobis distances are identified by a label (the observation number). The default for id.n is the number of all outliers: regression outliers (i.e., observations whose absolute residuals are too large, cf. weights) and leverage points (i.e., observations with robust Mahalanobis distance larger than the 97.5% quantile of the χ^2 distribution with p degrees of freedom).

Value

If only one plot is requested, an object of class "ggplot" (see ggplot), otherwise a list of such objects.

Author(s)

Andreas Alfons

See Also

ggplot, rlars, grplars, rgrplars, tslarsP, rtslarsP, tslars, rtslars, sparseLTS, plot.lts

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# create plot
diagnosticPlot(fitRlars)
```

```
## sparse LTS
# fit model
fitSparseLTS <- sparseLTS(x, y, lambda = 0.05, mode = "fraction")
# create plot
diagnosticPlot(fitSparseLTS)
diagnosticPlot(fitSparseLTS, fit = "both")
```

`fitted.seqModel` *Extract fitted values from a sequence of regression models*

Description

Extract fitted values from a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
## S3 method for class 'seqModel'
fitted(object, s = NA, drop = !is.null(s), ...)

## S3 method for class 'tslars'
fitted(object, p, ...)

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

## S3 method for class 'sparseLTS'
fitted(
  object,
  s = NA,
  fit = c("reweighted", "raw", "both"),
  drop = !is.null(s),
  ...
)
```

Arguments

`object`	the model fit from which to extract fitted values.
`s`	for the `"seqModel"` method, an integer vector giving the steps of the submodels for which to extract the fitted values (the default is to use the optimal submodel). For the `"sparseLTS"` method, an integer vector giving the indices of the models for which to extract fitted values. If `fit` is `"both"`, this can be a list with two components, with the first component giving the indices of the reweighted fits and the second the indices of the raw fits. The default is to use the optimal

	model for each of the requested estimators. Note that the optimal models may not correspond to the same value of the penalty parameter for the reweighted and the raw estimator.
drop	a logical indicating whether to reduce the dimension to a vector in case of only one step.
...	for the "tslars" method, additional arguments to be passed down to the "seqModel" method. For the other methods, additional arguments are currently ignored.
p	an integer giving the lag length for which to extract fitted values (the default is to use the optimal lag length).
fit	a character string specifying which fitted values to extract. Possible values are "reweighted" (the default) for the fitted values from the reweighted estimator, "raw" for the fitted values from the raw estimator, or "both" for the fitted values from both estimators.

Value

A numeric vector or matrix containing the requested fitted values.

Author(s)

Andreas Alfons

See Also

fitted, rlars, grplars, rgrplars, tslarsP, rtslarsP, tslars, rtslars, sparseLTS

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# extract fitted values
```

```
fitted(fitRlars)
head(fitted(fitRlars, s = 1:5))


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# extract fitted values
fitted(fitSparseLTS)
head(fitted(fitSparseLTS, fit = "both"))
head(fitted(fitSparseLTS, s = NULL))
head(fitted(fitSparseLTS, fit = "both", s = NULL))
```

`getScale` *Extract the residual scale of a robust regression model*

Description

Extract the robust scale estimate of the residuals from a robust regression model.

Usage

```
getScale(x, ...)

## S3 method for class 'seqModel'
getScale(x, s = NA, ...)

## S3 method for class 'sparseLTS'
getScale(x, s = NA, fit = c("reweighted", "raw", "both"), ...)
```

Arguments

`x`	the model fit from which to extract the robust residual scale estimate.
`...`	additional arguments to be passed down to methods.
`s`	for the `"seqModel"` method, an integer vector giving the steps of the submodels for which to extract the robust residual scale estimate (the default is to use the optimal submodel). For the `"sparseLTS"` method, an integer vector giving the indices of the models from which to extract the robust residual scale estimate. If `fit` is `"both"`, this can be a list with two components, with the first component giving the indices of the reweighted fits and the second the indices of the raw fits. The default is to use the optimal model for each of the requested estimators. Note that the optimal models may not correspond to the same value of the penalty parameter for the reweighted and the raw estimator.
`fit`	a character string specifying from which fit to extract the robust residual scale estimate. Possible values are `"reweighted"` (the default) for the residual scale of the reweighted fit, `"raw"` for the residual scale of the raw fit, or `"both"` for the residual scale of both fits.

Details

Methods are implemented for models of class "lmrob" (see lmrob), "lts" (see ltsReg), "rlm" (see rlm), "seqModel" (see rlars) and "sparseLTS" (see sparseLTS). The default method computes the MAD of the residuals.

Value

A numeric vector or matrix giving the robust residual scale estimates for the requested model fits.

Author(s)

Andreas Alfons

See Also

AIC, lmrob, ltsReg, rlm, rlars, sparseLTS

Examples

```
data("coleman")
fit <- lmrob(Y ~ ., data=coleman)
getScale(fit)
```

---

grplars *(Robust) groupwise least angle regression*

Description

(Robustly) sequence groups of candidate predictors according to their predictive content and find the optimal model along the sequence.

Usage

```
grplars(x, ...)

## S3 method for class 'formula'
grplars(formula, data, ...)

## S3 method for class 'data.frame'
grplars(x, y, ...)

## Default S3 method:
grplars(
  x,
  y,
  sMax = NA,
  assign,
```

```
  fit = TRUE,
  s = c(0, sMax),
  crit = c("BIC", "PE"),
  splits = foldControl(),
  cost = rmspe,
  costArgs = list(),
  selectBest = c("hastie", "min"),
  seFactor = 1,
  ncores = 1,
  cl = NULL,
  seed = NULL,
  model = TRUE,
  ...
)

rgrplars(x, ...)

## S3 method for class 'formula'
rgrplars(formula, data, ...)

## S3 method for class 'data.frame'
rgrplars(x, y, ...)

## Default S3 method:
rgrplars(
  x,
  y,
  sMax = NA,
  assign,
  centerFun = median,
  scaleFun = mad,
  regFun = lmrob,
  regArgs = list(),
  combine = c("min", "euclidean", "mahalanobis"),
  const = 2,
  prob = 0.95,
  fit = TRUE,
  s = c(0, sMax),
  crit = c("BIC", "PE"),
  splits = foldControl(),
  cost = rtmspe,
  costArgs = list(),
  selectBest = c("hastie", "min"),
  seFactor = 1,
  ncores = 1,
  cl = NULL,
  seed = NULL,
  model = TRUE,
```

```
  ...
)
```

Arguments

x	a matrix or data frame containing the candidate predictors.
...	additional arguments to be passed down.
formula	a formula describing the full model.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which grplars or rgrplars is called.
y	a numeric vector containing the response.
sMax	an integer giving the number of predictor groups to be sequenced. If it is NA (the default), predictor groups are sequenced as long as there are twice as many observations as expected predictor variables (number of predictor groups times the average number of predictor variables per group).
assign	an integer vector giving the predictor group to which each predictor variable belongs.
fit	a logical indicating whether to fit submodels along the sequence (TRUE, the default) or to simply return the sequence (FALSE).
s	an integer vector of length two giving the first and last step along the sequence for which to compute submodels. The default is to start with a model containing only an intercept (step 0) and iteratively add all groups along the sequence (step sMax). If the second element is NA, predictor groups are added to the model as long as there are twice as many observations as predictor variables. If only one value is supplied, it is recycled.
crit	a character string specifying the optimality criterion to be used for selecting the final model. Possible values are "BIC" for the Bayes information criterion and "PE" for resampling-based prediction error estimation.
splits	an object giving data splits to be used for prediction error estimation (see perry).
cost	a cost function measuring prediction loss (see perry for some requirements). The default is to use the root trimmed mean squared prediction error for a robust fit and the root mean squared prediction error otherwise (see cost).
costArgs	a list of additional arguments to be passed to the prediction loss function cost.
selectBest, seFactor	arguments specifying a criterion for selecting the best model (see perrySelect). The default is to use a one-standard-error rule.
ncores	a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to NA, all available processor cores are used. For obtaining the data cleaning weights, for fitting models along the sequence and for prediction error estimation, parallel computing is implemented on the R level using package **parallel**. Otherwise parallel computing for some of of the more computer-intensive computations in the sequencing step is implemented on the C++ level via OpenMP (https://www.openmp.org/).

`cl`	a **parallel** cluster for parallel computing as generated by `makeCluster`. This is preferred over `ncores` for tasks that are parallelized on the R level, in which case `ncores` is only used for tasks that are parallelized on the C++ level.
`seed`	optional initial seed for the random number generator (see `.Random.seed`). This is useful because many robust regression functions (including `lmrob`) involve randomness, or for prediction error estimation. On parallel R worker processes, random number streams are used and the seed is set via `clusterSetRNGStream`.
`model`	a logical indicating whether the model data should be included in the returned object.
`centerFun`	a function to compute a robust estimate for the center (defaults to `median`).
`scaleFun`	a function to compute a robust estimate for the scale (defaults to `mad`).
`regFun`	a function to compute robust linear regressions that can be interpreted as weighted least squares (defaults to `lmrob`).
`regArgs`	a list of arguments to be passed to `regFun`.
`combine`	a character string specifying how to combine the data cleaning weights from the robust regressions with each predictor group. Possible values are `"min"` for taking the minimum weight for each observation, `"euclidean"` for weights based on Euclidean distances of the multivariate set of standardized residuals (i.e., multivariate winsorization of the standardized residuals assuming independence), or `"mahalanobis"` for weights based on Mahalanobis distances of the multivariate set of standardized residuals (i.e., multivariate winsorization of the standardized residuals).
`const`	numeric; tuning constant for multivariate winsorization to be used in the initial corralation estimates based on adjusted univariate winsorization (defaults to 2).
`prob`	numeric; probability for the quantile of the χ^2 distribution to be used in multivariate winsorization (defaults to 0.95).

Value

If `fit` is `FALSE`, an integer vector containing the indices of the sequenced predictor groups.

Else if `crit` is `"PE"`, an object of class `"perrySeqModel"` (inheriting from classes `"perryTuning"`, see `perryTuning`). It contains information on the prediction error criterion, and includes the final model as component `finalModel`.

Otherwise an object of class `"grplars"` (inheriting from class `"seqModel"`) with the following components:

`active` an integer vector containing the sequence of predictor groups.

`s` an integer vector containing the steps for which submodels along the sequence have been computed.

`coefficients` a numeric matrix in which each column contains the regression coefficients of the corresponding submodel along the sequence.

`fitted.values` a numeric matrix in which each column contains the fitted values of the corresponding submodel along the sequence.

`residuals` a numeric matrix in which each column contains the residuals of the corresponding submodel along the sequence.

`df` an integer vector containing the degrees of freedom of the submodels along the sequence (i.e., the number of estimated coefficients).

`robust` a logical indicating whether a robust fit was computed.

`scale` a numeric vector giving the robust residual scale estimates for the submodels along the sequence (only returned for a robust fit).

`crit` an object of class `"bicSelect"` containing the BIC values and indicating the final model (only returned if argument `crit` is `"BIC"` and argument `s` indicates more than one step along the sequence).

`muX` a numeric vector containing the center estimates of the predictor variables.

`sigmaX` a numeric vector containing the scale estimates of the predictor variables.

`muY` numeric; the center estimate of the response.

`sigmaY` numeric; the scale estimate of the response.

`x` the matrix of candidate predictors (if `model` is `TRUE`).

`y` the response (if `model` is `TRUE`).

`assign` an integer vector giving the predictor group to which each predictor variable belongs.

`w` a numeric vector giving the data cleaning weights (only returned for a robust fit).

`call` the matched function call.

Author(s)

Andreas Alfons

References

Alfons, A., Croux, C. and Gelper, S. (2016) Robust groupwise least angle regression. *Computational Statistics & Data Analysis*, **93**, 421–435.

See Also

`coef`, `fitted`, `plot`, `predict`, `residuals`, `lmrob`

Examples

```
data("TopGear")
# keep complete observations
keep <- complete.cases(TopGear)
TopGear <- TopGear[keep, ]
# remove information on car model
info <- TopGear[, 1:3]
TopGear <- TopGear[, -(1:3)]
# log-transform price
TopGear$Price <- log(TopGear$Price)

# robust groupwise LARS
rgrplars(MPG ~ ., data = TopGear, sMax = 15)
```

`lambda0` *Penalty parameter for sparse LTS regression*

Description

Use bivariate winsorization to estimate the smallest value of the penalty parameter for sparse least trimmed squares regression that sets all coefficients to zero.

Usage

```
lambda0(
  x,
  y,
  normalize = TRUE,
  intercept = TRUE,
  const = 2,
  prob = 0.95,
  tol = .Machine$double.eps^0.5,
  eps = .Machine$double.eps,
  ...
)
```

Arguments

`x`	a numeric matrix containing the predictor variables.
`y`	a numeric vector containing the response variable.
`normalize`	a logical indicating whether the winsorized predictor variables should be normalized to have unit L_2 norm (the default is `TRUE`).
`intercept`	a logical indicating whether a constant term should be included in the model (the default is `TRUE`).
`const`	numeric; tuning constant to be used in univariate winsorization (defaults to 2).
`prob`	numeric; probability for the quantile of the χ^2 distribution to be used in bivariate winsorization (defaults to 0.95).
`tol`	a small positive numeric value used to determine singularity issues in the computation of correlation estimates for bivariate winsorization (see `corHuber`).
`eps`	a small positive numeric value used to determine whether the robust scale estimate of a variable is too small (an effective zero).
`...`	additional arguments to be passed to `robStandardize`.

Details

The estimation procedure is inspired by the calculation of the respective penalty parameter in the first step of the classical LARS algorithm. First, two-dimensional data blocks consisting of the response with each predictor variable are cleaned via bivariate winsorization. For each block, the following computations are then performed. If an intercept is included in the model, the cleaned

response is centered and the corresponding cleaned predictor is centered and scaled to have unit norm. Otherwise the variables are not centered, but the predictor is scaled to have unit norm. Finally, the dot product of the response and the corresponding predictor is computed. The largest absolute value of those dot products, rescaled to fit the parametrization of the sparse LTS definition, yields the estimate of the smallest penalty parameter that sets all coefficients to zero.

Value

A robust estimate of the smallest value of the penalty parameter for sparse LTS regression that sets all coefficients to zero.

Author(s)

Andreas Alfons

References

Alfons, A., Croux, C. and Gelper, S. (2013) Sparse least trimmed squares regression for analyzing high-dimensional large data sets. *The Annals of Applied Statistics*, **7**(1), 226–248.

Efron, B., Hastie, T., Johnstone, I. and Tibshirani, R. (2004) Least angle regression. *The Annals of Statistics*, **32**(2), 407–499.

Khan, J.A., Van Aelst, S. and Zamar, R.H. (2007) Robust linear model selection based on least angle regression. *Journal of the American Statistical Association*, **102**(480), 1289–1299.

See Also

sparseLTS, winsorize

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points

## estimate smallest value of the penalty parameter
## that sets all coefficients to 0
lambda0(x, y)
```

nci60 *NCI-60 cancer cell panel*

Description

The data set is a pre-processed version of the NCI-60 cancer cell panel as used in Alfons, Croux & Gelper (2013). One observation was removed since all values in the gene expression data were missing.

Usage

```
data("nci60")
```

Format

Protein and gene expression data on 59 observations are stored in two separate matrices:

`protein` a matrix containing protein expressions based on antibodies (162 variables), acquired via reverse-phase protein lysate arrays and log2 transformed.

`gene` a matrix containing gene expression data (22283 variables), obtained with an Affymetrix HG-U133A chip and normalized with the GCRMA method.

Source

The original data were downloaded from https://discover.nci.nih.gov/cellminer/ on 2012-01-27. They can be obtained from https://github.com/aalfons/nci60, together with our script for pre-processing.

References

Reinhold, W.C., Sunshine, M., Liu, H., Varma, S., Kohn, K.W., Morris, J., Doroshow, J. and Pommier, Y. (2012) CellMiner: A Web-Based Suite of Genomic and Pharmacologic Tools to Explore Transcript and Drug Patterns in the NCI-60 Cell Line Set. *Cancer Research*, **72**(14), 3499–3511.

Alfons, A., Croux, C. and Gelper, S. (2013) Sparse least trimmed squares regression for analyzing high-dimensional large data sets. *The Annals of Applied Statistics*, **7**(1), 226–248.

Examples

```
## Not run:

# load data
data("nci60")
# define response variable
y <- protein[, 92]
# screen most correlated predictor variables
correlations <- apply(gene, 2, corHuber, y)
keep <- partialOrder(abs(correlations), 100, decreasing = TRUE)
X <- gene[, keep]
```

```
## End(Not run)
```

---

partialOrder *Find partial order of smallest or largest values*

Description

Obtain a partial permutation that rearranges the smallest (largest) elements of a vector into ascending (descending) order.

Usage

```
partialOrder(x, h, decreasing = FALSE)
```

Arguments

x	a numeric vector of which to find the order of the smallest or largest elements.
h	an integer specifying how many (smallest or largest) elements to order.
decreasing	a logical indicating whether the sort order should be increasing (FALSE; the default) or decreasing (TRUE).

Value

An integer vector containing the indices of the h smallest or largest elements of x.

Author(s)

Andreas Alfons

See Also

order

Examples

```
# randomly draw some values
values <- rnorm(10)
values

# find largest observations
partialOrder(values, 5, decreasing = TRUE)
```

perry.seqModel *Resampling-based prediction error for a sequential regression model*

Description

Estimate the prediction error of a previously fit sequential regression model such as a robust least angle regression model or a sparse least trimmed squares regression model.

Usage

```
## S3 method for class 'seqModel'
perry(
  object,
  splits = foldControl(),
  cost,
  ncores = 1,
  cl = NULL,
  seed = NULL,
  ...
)

## S3 method for class 'sparseLTS'
perry(
  object,
  splits = foldControl(),
  fit = c("reweighted", "raw", "both"),
  cost = rtmspe,
  ncores = 1,
  cl = NULL,
  seed = NULL,
  ...
)
```

Arguments

object	the model fit for which to estimate the prediction error.
splits	an object of class "cvFolds" (as returned by cvFolds) or a control object of class "foldControl" (see foldControl) defining the folds of the data for (repeated) K-fold cross-validation, an object of class "randomSplits" (as returned by randomSplits) or a control object of class "splitControl" (see splitControl) defining random data splits, or an object of class "bootSamples" (as returned by bootSamples) or a control object of class "bootControl" (see bootControl) defining bootstrap samples.
cost	a cost function measuring prediction loss. It should expect vectors to be passed as its first two arguments, the first corresponding to the observed values of the response and the second to the predicted values, and must return a non-negative

scalar value. The default is to use the root mean squared prediction error for non-robust models and the root trimmed mean squared prediction error for robust models (see cost).

ncores a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to NA, all available processor cores are used.

cl a **parallel** cluster for parallel computing as generated by makeCluster. If supplied, this is preferred over ncores.

seed optional initial seed for the random number generator (see .Random.seed). Note that also in case of parallel computing, resampling is performed on the manager process rather than the worker processes. On the parallel worker processes, random number streams are used and the seed is set via clusterSetRNGStream.

... additional arguments to be passed to the prediction loss function cost.

fit a character string specifying for which fit to estimate the prediction error. Possible values are "reweighted" (the default) for the prediction error of the reweighted fit, "raw" for the prediction error of the raw fit, or "both" for the prediction error of both fits.

Details

The prediction error can be estimated via (repeated) K-fold cross-validation, (repeated) random splitting (also known as random subsampling or Monte Carlo cross-validation), or the bootstrap. In each iteration, the optimal model is thereby selected from the training data and used to make predictions for the test data.

Value

An object of class "perry" with the following components:

pe a numeric vector containing the estimated prediction errors for the requested model fits. In case of more than one replication, this gives the average value over all replications.

se a numeric vector containing the estimated standard errors of the prediction loss for the requested model fits.

reps a numeric matrix in which each column contains the estimated prediction errors from all replications for the requested model fits. This is only returned in case of more than one replication.

splits an object giving the data splits used to estimate the prediction error.

y the response.

yHat a list containing the predicted values from all replications.

call the matched function call.

Author(s)

Andreas Alfons

See Also

rlars, sparseLTS, predict, perry, cost

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## fit and evaluate robust LARS model
fitRlars <- rlars(x, y, sMax = 10)
perry(fitRlars)

## fit and evaluate sparse LTS model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
perry(fitSparseLTS)
```

---

plot.seqModel *Plot a sequence of regression models*

Description

Produce a plot of the coefficients, the values of the optimality criterion, or diagnostic plots for a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
## S3 method for class 'seqModel'
plot(x, method = c("coefficients", "crit", "diagnostic"), ...)

## S3 method for class 'perrySeqModel'
plot(x, method = c("crit", "diagnostic"), ...)
```

```
## S3 method for class 'tslars'
plot(x, p, method = c("coefficients", "crit", "diagnostic"), ...)

## S3 method for class 'sparseLTS'
plot(x, method = c("coefficients", "crit", "diagnostic"), ...)

## S3 method for class 'perrySparseLTS'
plot(x, method = c("crit", "diagnostic"), ...)
```

Arguments

x	the model fit to be plotted.
method	a character string specifying the type of plot. Possible values are "coefficients" to plot the coefficients from the submodels via coefPlot (only for the "seqModel" and "sparseLTS" methods), "crit" to plot the values of the optimality criterion for the submodels via critPlot, or "diagnostic" for diagnostic plots via diagnosticPlot.
...	additional arguments to be passed down.
p	an integer giving the lag length for which to produce the plot (the default is to use the optimal lag length).

Value

An object of class "ggplot" (see ggplot).

Author(s)

Andreas Alfons

See Also

coefPlot, critPlot, diagnosticPlot, rlars, grplars, rgrplars, tslarsP, rtslarsP, tslars, rtslars, sparseLTS

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
```

```
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# create plots
plot(fitRlars, method = "coef")
plot(fitRlars, method = "crit")
plot(fitRlars, method = "diagnostic")


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# create plots
plot(fitSparseLTS, method = "coef")
plot(fitSparseLTS, method = "crit")
plot(fitSparseLTS, method = "diagnostic")
```

`predict.seqModel` *Predict from a sequence of regression models*

Description

Make predictions from a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter. For autoregressive time series models with exogenous inputs, h-step ahead forecasts are performed.

Usage

```
## S3 method for class 'seqModel'
predict(object, newdata, s = NA, ...)

## S3 method for class 'tslarsP'
predict(object, newdata, ...)

## S3 method for class 'tslars'
predict(object, newdata, p, ...)

## S3 method for class 'sparseLTS'
predict(object, newdata, s = NA, fit = c("reweighted", "raw", "both"), ...)
```

Arguments

`object`	the model fit from which to make predictions.
`newdata`	new data for the predictors. If the model fit was computed with the formula method, this should be a data frame from which to extract the predictor variables. Otherwise this should be a matrix containing the same variables as the predictor matrix used to fit the model (including a column of ones to account for the intercept).
`s`	for the `"seqModel"` method, an integer vector giving the steps of the submodels for which to make predictions (the default is to use the optimal submodel). For the `"sparseLTS"` method, an integer vector giving the indices of the models for which to make predictions. If `fit` is `"both"`, this can be a list with two components, with the first component giving the indices of the reweighted fits and the second the indices of the raw fits. The default is to use the optimal model for each of the requested estimators. Note that the optimal models may not correspond to the same value of the penalty parameter for the reweighted and the raw estimator.
`...`	for the `"tslars"` method, additional arguments to be passed down to the `"tslarsP"` method. For the other methods, additional arguments to be passed down to the respective method of `coef`.
`p`	an integer giving the lag length for which to make predictions (the default is to use the optimal lag length).
`fit`	a character string specifying for which fit to make predictions. Possible values are `"reweighted"` (the default) for predicting values from the reweighted fit, `"raw"` for predicting values from the raw fit, or `"both"` for predicting values from both fits.

Details

The `newdata` argument defaults to the matrix of predictors used to fit the model such that the fitted values are computed.

For autoregressive time series models with exogenous inputs with forecast horizon h, the h most recent observations of the predictors are omitted from fitting the model since there are no corresponding values for the response. Hence the `newdata` argument for `predict.tslarsP` and `predict.tslars` defaults to those h observations of the predictors.

Value

A numeric vector or matrix containing the requested predicted values.

Author(s)

Andreas Alfons

See Also

`predict`, `rlars`, `grplars`, `rgrplars`, `tslarsP`, `rtslarsP`, `tslars`, `rtslars`, `sparseLTS`

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# compute fitted values via predict method
predict(fitRlars)
head(predict(fitRlars, s = 1:5))


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# compute fitted values via predict method
predict(fitSparseLTS)
head(predict(fitSparseLTS, fit = "both"))
head(predict(fitSparseLTS, s = NULL))
head(predict(fitSparseLTS, fit = "both", s = NULL))
```

`residuals.seqModel` *Extract residuals from a sequence of regression models*

Description

Extract residuals from a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
## S3 method for class 'seqModel'
```

```
residuals(object, s = NA, standardized = FALSE, drop = !is.null(s), ...)

## S3 method for class 'tslars'
residuals(object, p, ...)

## S3 method for class 'perrySeqModel'
residuals(object, ...)

## S3 method for class 'sparseLTS'
residuals(
  object,
  s = NA,
  fit = c("reweighted", "raw", "both"),
  standardized = FALSE,
  drop = !is.null(s),
  ...
)
```

Arguments

object	the model fit from which to extract residuals.
s	for the "seqModel" method, an integer vector giving the steps of the submodels for which to extract the residuals (the default is to use the optimal submodel). For the "sparseLTS" method, an integer vector giving the indices of the models for which to extract residuals. If fit is "both", this can be a list with two components, with the first component giving the indices of the reweighted fits and the second the indices of the raw fits. The default is to use the optimal model for each of the requested estimators. Note that the optimal models may not correspond to the same value of the penalty parameter for the reweighted and the raw estimator.
standardized	a logical indicating whether the residuals should be standardized (the default is FALSE).
drop	a logical indicating whether to reduce the dimension to a vector in case of only one step.
...	for the "tslars" method, additional arguments to be passed down to the "seqModel" method. For the other methods, additional arguments are currently ignored.
p	an integer giving the lag length for which to extract residuals (the default is to use the optimal lag length).
fit	a character string specifying which residuals to extract. Possible values are "reweighted" (the default) for the residuals from the reweighted estimator, "raw" for the residuals from the raw estimator, or "both" for the residuals from both estimators.

Value

A numeric vector or matrix containing the requested residuals.

Author(s)

Andreas Alfons

See Also

residuals, rlars, grplars, rgrplars, tslarsP, rtslarsP, tslars, rtslars, sparseLTS

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# extract residuals
residuals(fitRlars)
head(residuals(fitRlars, s = 1:5))


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# extract residuals
residuals(fitSparseLTS)
head(residuals(fitSparseLTS, fit = "both"))
head(residuals(fitSparseLTS, s = NULL))
head(residuals(fitSparseLTS, fit = "both", s = NULL))
```

---

rlars *Robust least angle regression*

Description

Robustly sequence candidate predictors according to their predictive content and find the optimal model along the sequence.

Usage

```
rlars(x, ...)

## S3 method for class 'formula'
rlars(formula, data, ...)

## Default S3 method:
rlars(
  x,
  y,
  sMax = NA,
  centerFun = median,
  scaleFun = mad,
  winsorize = FALSE,
  const = 2,
  prob = 0.95,
  fit = TRUE,
  s = c(0, sMax),
  regFun = lmrob,
  regArgs = list(),
  crit = c("BIC", "PE"),
  splits = foldControl(),
  cost = rtmspe,
  costArgs = list(),
  selectBest = c("hastie", "min"),
  seFactor = 1,
  ncores = 1,
  cl = NULL,
  seed = NULL,
  model = TRUE,
  tol = .Machine$double.eps^0.5,
  ...
)
```

Arguments

x	a matrix or data frame containing the candidate predictors.
...	additional arguments to be passed down. For the default method, additional arguments to be passed down to robStandardize.
formula	a formula describing the full model.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data,

	the variables are taken from `environment(formula)`, typically the environment from which `rlars` is called.
`y`	a numeric vector containing the response.
`sMax`	an integer giving the number of predictors to be sequenced. If it is `NA` (the default), predictors are sequenced as long as there are twice as many observations as predictors.
`centerFun`	a function to compute a robust estimate for the center (defaults to `median`).
`scaleFun`	a function to compute a robust estimate for the scale (defaults to `mad`).
`winsorize`	a logical indicating whether to clean the full data set by multivariate winsorization, i.e., to perform data cleaning RLARS instead of plug-in RLARS (defaults to `FALSE`).
`const`	numeric; tuning constant to be used in the initial corralation estimates based on adjusted univariate winsorization (defaults to 2).
`prob`	numeric; probability for the quantile of the χ^2 distribution to be used in bivariate or multivariate winsorization (defaults to 0.95).
`fit`	a logical indicating whether to fit submodels along the sequence (`TRUE`, the default) or to simply return the sequence (`FALSE`).
`s`	an integer vector of length two giving the first and last step along the sequence for which to compute submodels. The default is to start with a model containing only an intercept (step 0) and iteratively add all variables along the sequence (step `sMax`). If the second element is `NA`, predictors are added to the model as long as there are twice as many observations as predictors. If only one value is supplied, it is recycled.
`regFun`	a function to compute robust linear regressions along the sequence (defaults to `lmrob`).
`regArgs`	a list of arguments to be passed to `regFun`.
`crit`	a character string specifying the optimality criterion to be used for selecting the final model. Possible values are `"BIC"` for the Bayes information criterion and `"PE"` for resampling-based prediction error estimation.
`splits`	an object giving data splits to be used for prediction error estimation (see `perry`).
`cost`	a cost function measuring prediction loss (see `perry` for some requirements). The default is to use the root trimmed mean squared prediction error (see `cost`).
`costArgs`	a list of additional arguments to be passed to the prediction loss function `cost`.
`selectBest, seFactor`	arguments specifying a criterion for selecting the best model (see `perrySelect`). The default is to use a one-standard-error rule.
`ncores`	a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to `NA`, all available processor cores are used. For fitting models along the sequence and for prediction error estimation, parallel computing is implemented on the R level using package **parallel**. Otherwise parallel computing for some of of the more computer-intensive computations in the sequencing step is implemented on the C++ level via OpenMP (`https://www.openmp.org/`).

`cl` a **parallel** cluster for parallel computing as generated by `makeCluster`. This is preferred over `ncores` for tasks that are parallelized on the R level, in which case `ncores` is only used for tasks that are parallelized on the C++ level.

`seed` optional initial seed for the random number generator (see `.Random.seed`). This is useful because many robust regression functions (including `lmrob`) involve randomness, or for prediction error estimation. On parallel R worker processes, random number streams are used and the seed is set via `clusterSetRNGStream`.

`model` a logical indicating whether the model data should be included in the returned object.

`tol` a small positive numeric value. This is used in bivariate winsorization to determine whether the initial estimate from adjusted univariate winsorization is close to 1 in absolute value. In this case, bivariate winsorization would fail since the points form almost a straight line, and the initial estimate is returned.

Value

If `fit` is `FALSE`, an integer vector containing the indices of the sequenced predictors.

Else if `crit` is `"PE"`, an object of class `"perrySeqModel"` (inheriting from class `"perrySelect"`, see `perrySelect`). It contains information on the prediction error criterion, and includes the final model as component `finalModel`.

Otherwise an object of class `"rlars"` (inheriting from class `"seqModel"`) with the following components:

`active` an integer vector containing the indices of the sequenced predictors.

`s` an integer vector containing the steps for which submodels along the sequence have been computed.

`coefficients` a numeric matrix in which each column contains the regression coefficients of the corresponding submodel along the sequence.

`fitted.values` a numeric matrix in which each column contains the fitted values of the corresponding submodel along the sequence.

`residuals` a numeric matrix in which each column contains the residuals of the corresponding submodel along the sequence.

`df` an integer vector containing the degrees of freedom of the submodels along the sequence (i.e., the number of estimated coefficients).

`robust` a logical indicating whether a robust fit was computed (`TRUE` for `"rlars"` models).

`scale` a numeric vector giving the robust residual scale estimates for the submodels along the sequence.

`crit` an object of class `"bicSelect"` containing the BIC values and indicating the final model (only returned if argument `crit` is `"BIC"` and argument `s` indicates more than one step along the sequence).

`muX` a numeric vector containing the center estimates of the predictors.

`sigmaX` a numeric vector containing the scale estimates of the predictors.

`muY` numeric; the center estimate of the response.

`sigmaY` numeric; the scale estimate of the response.

`x` the matrix of candidate predictors (if `model` is `TRUE`).

`y` the response (if `model` is `TRUE`).

`w` a numeric vector giving the data cleaning weights (if `winsorize` is `TRUE`).

`call` the matched function call.

Author(s)

Andreas Alfons, based on code by Jafar A. Khan, Stefan Van Aelst and Ruben H. Zamar

References

Khan, J.A., Van Aelst, S. and Zamar, R.H. (2007) Robust linear model selection based on least angle regression. *Journal of the American Statistical Association*, **102**(480), 1289–1299.

See Also

`coef`, `fitted`, `plot`, `predict`, `residuals`, `lmrob`

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points

## fit robust LARS model
rlars(x, y, sMax = 10)
```

`robustHD-deprecated` *Deprecated functions in package* **robustHD**

Description

These functions are provided for compatibility with older versions only, and may be defunct as soon as the next release.

Usage

```
## S3 method for class 'seqModel'
fortify(model, data, s = NA, covArgs = list(...), ...)

## S3 method for class 'sparseLTS'
fortify(
  model,
  data,
  s = NA,
  fit = c("reweighted", "raw", "both"),
  covArgs = list(...),
  ...
)

## Default S3 method:
diagnosticPlot(
  object,
  which = c("all", "rqq", "rindex", "rfit", "rdiag"),
  ask = (which == "all"),
  facets = attr(object, "facets"),
  size = c(2, 4),
  id.n = NULL,
  ...
)
```

Arguments

model	the model fit to be converted.
data	currently ignored.
s	for the "seqModel" method, an integer vector giving the steps of the submodels to be converted (the default is to use the optimal submodel). For the "sparseLTS" method, an integer vector giving the indices of the models to be converted (the default is to use the optimal model for each of the requested fits).
covArgs	a list of arguments to be passed to covMcd for computing robust Mahalanobis distances.
...	for the fortify methods, additional arguments to be passed to covMcd can be specified directly instead of via covArgs. For the default method of diagnosticPlot, additional arguments to be passed down to geom_point.
fit	a character string specifying which fit to convert. Possible values are "reweighted" (the default) to convert the reweighted fit, "raw" to convert the raw fit, or "both" to convert both fits.
object	a data frame containing all necessary information for plotting (as generated by the fortify methods).
which	a character string indicating which plot to show. Possible values are "all" (the default) for all of the following, "rqq" for a normal Q-Q plot of the standardized residuals, "rindex" for a plot of the standardized residuals versus their

index, "rfit" for a plot of the standardized residuals versus the fitted values, or "rdiag" for a regression diagnostic plot (standardized residuals versus robust Mahalanobis distances of the predictor variables).

ask a logical indicating whether the user should be asked before each plot (see devAskNewPage). The default is to ask if all plots are requested and not ask otherwise.

facets a faceting formula to override the default behavior. If supplied, facet_wrap or facet_grid is called depending on whether the formula is one-sided or two-sided.

size a numeric vector of length two giving the point and label size, respectively.

id.n an integer giving the number of the most extreme observations to be identified by a label. The default is to use the number of identified outliers, which can be different for the different plots.

Details

The fortify methods supplement the fitted values and residuals of a sequence of regression models (such as robust least angle regression models or sparse least trimmed squares regression models) with other useful information for diagnostic plots.

The default method of diagnosticPlot creates the corresponding plot from the data frame returned by fortify.

Value

The fortify methods return data frame containing the columns listed below, as well as additional information stored in the attributes "qqLine" (intercepts and slopes of the respective reference lines to be displayed in residual Q-Q plots), "q" (quantiles of the Mahalanobis distribution used as cutoff points for detecting leverage points), and "facets" (default faceting formula for the diagnostic plots).

step the steps (for the "seqModel" method) or indices (for the "sparseLTS" method) of the models (only returned if more than one model is requested).

fit the model fits (only returned if both the reweighted and raw fit are requested in the "sparseLTS" method).

index the indices of the observations.

fitted the fitted values.

residual the standardized residuals.

theoretical the corresponding theoretical quantiles from the standard normal distribution.

qqd the absolute distances from a reference line through the first and third sample and theoretical quartiles.

rd the robust Mahalanobis distances computed via the MCD (see covMcd).

xyd the pairwise maxima of the absolute values of the standardized residuals and the robust Mahalanobis distances, divided by the respective other outlier detection cutoff point.

weight the weights indicating regression outliers.

leverage logicals indicating leverage points (i.e., outliers in the predictor space).

classification a factor with levels "outlier" (regression outliers) and "good" (data points following the model).

Author(s)

Andreas Alfons

`setupCoefPlot` *Set up a coefficient plot of a sequence of regression models*

Description

Extract the relevent information for a plot of the coefficients for a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
setupCoefPlot(object, ...)

## S3 method for class 'seqModel'
setupCoefPlot(object, zeros = FALSE, labels = NULL, ...)

## S3 method for class 'tslars'
setupCoefPlot(object, p, ...)

## S3 method for class 'sparseLTS'
setupCoefPlot(
  object,
  fit = c("reweighted", "raw", "both"),
  zeros = FALSE,
  labels = NULL,
  ...
)
```

Arguments

`object`	the model fit from which to extract information.
`...`	additional arguments to be passed down.
`zeros`	a logical indicating whether predictors that never enter the model and thus have zero coefficients should be included in the plot (`TRUE`) or omitted (`FALSE`, the default). This is useful if the number of predictors is much larger than the number of observations, in which case many coefficients are never nonzero.
`labels`	an optional character vector containing labels for the predictors. Information on labels can be suppressed by setting this to `NA`.
`p`	an integer giving the lag length for which to extract information (the default is to use the optimal lag length).
`fit`	a character string specifying for which estimator to extract information. Possible values are `"reweighted"` (the default) for the reweighted fits, `"raw"` for the raw fits, or `"both"` for both estimators.

Value

An object inheriting from class `"setupCoefPlot"` with the following components:

`coefficients` a data frame containing the following columns:

- `fit` the model fit for which the coefficient is computed (only returned if both the reweighted and raw fit are requested in the `"sparseLTS"` method).
- `lambda` the value of the penalty parameter for which the coefficient is computed (only returned for the `"sparseLTS"` method).
- `step` the step along the sequence for which the coefficient is computed.
- `df` the degrees of freedom of the submodel along the sequence for which the coefficient is computed.
- `coefficient` the value of the coefficient.
- `variable` a character string specifying to which variable the coefficient belongs.

`abscissa` a character string specifying available options for what to plot on the x-axis

`lambda` a numeric vector giving the values of the penalty parameter. (only returned for the `"sparseLTS"` method).

`step` an integer vector containing the steps for which submodels along the sequence have been computed.

`df` an integer vector containing the degrees of freedom of the submodels along the sequence (i.e., the number of estimated coefficients; only returned for the `"seqModel"` method).

`includeLabels` a logical indicating whether information on labels for the variables should be included in the plot.

`labels` a data frame containing the following columns (not returned if information on labels is suppressed):

- `fit` the model fit for which the coefficient is computed (only returned if both the reweighted and raw fit are requested in the `"sparseLTS"` method).
- `lambda` the smallest value of the penalty parameter (only returned for the `"sparseLTS"` method).
- `step` the last step along the sequence.
- `df` the degrees of freedom of the last submodel along the sequence.
- `coefficient` the value of the coefficient.
- `label` the label of the corresponding variable to be displayed in the plot.

`facets` default faceting formula for the plots (only returned if both estimators are requested in the `"sparseLTS"` method).

Author(s)

Andreas Alfons

See Also

`coefPlot`, `rlars`, `grplars`, `rgrplars`, `tslarsP`, `rtslarsP`, `tslars`, `rtslars`, `sparseLTS`

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# extract information for plotting
setup <- setupCoefPlot(fitRlars)
coefPlot(setup)


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# extract information for plotting
setup1 <- setupCoefPlot(fitSparseLTS)
coefPlot(setup1)
setup2 <- setupCoefPlot(fitSparseLTS, fit = "both")
coefPlot(setup2)
```

`setupCritPlot` *Set up an optimality criterion plot of a sequence of regression models*

Description

Extract the relevent information for a plot of the values of the optimality criterion for a sequence of regression models, such as submodels along a robust or groupwise least angle regression sequence, or sparse least trimmed squares regression models for a grid of values for the penalty parameter.

Usage

```
setupCritPlot(object, ...)
```

```
## S3 method for class 'seqModel'
setupCritPlot(object, which = c("line", "dot"), ...)

## S3 method for class 'tslars'
setupCritPlot(object, p, ...)

## S3 method for class 'sparseLTS'
setupCritPlot(
  object,
  which = c("line", "dot"),
  fit = c("reweighted", "raw", "both"),
  ...
)

## S3 method for class 'perrySeqModel'
setupCritPlot(object, which = c("line", "dot", "box", "density"), ...)

## S3 method for class 'perrySparseLTS'
setupCritPlot(
  object,
  which = c("line", "dot", "box", "density"),
  fit = c("reweighted", "raw", "both"),
  ...
)
```

Arguments

`object`	the model fit from which to extract information.
`...`	additional arguments to be passed down.
`which`	a character string specifying the type of plot. Possible values are `"line"` (the default) to plot the (average) results for each model as a connected line, `"dot"` to create a dot plot, `"box"` to create a box plot, or `"density"` to create a smooth density plot. Note that the last two plots are only available in case of prediction error estimation via repeated resampling.
`p`	an integer giving the lag length for which to extract information (the default is to use the optimal lag length).
`fit`	a character string specifying for which estimator to extract information. Possible values are `"reweighted"` (the default) for the reweighted fits, `"raw"` for the raw fits, or `"both"` for both estimators.

Value

An object inheriting from class `"setupCritPlot"` with the following components:

`data` a data frame containing the following columns:

`Fit` a vector or factor containing the identifiers of the models along the sequence.

`Name` a factor specifying the estimator for which the optimality criterion was estimated (`"reweighted"` or `"raw"`; only returned if both are requested in the `"sparseLTS"` or `"perrySparseLTS"` methods).

`PE` the estimated prediction errors (only returned if applicable).

`BIC` the estimated values of the Bayesian information criterion (only returned if applicable).

`Lower` the lower end points of the error bars (only returned if possible to compute).

`Upper` the upper end points of the error bars (only returned if possible to compute).

`which` a character string specifying the type of plot.

`grouped` a logical indicating whether density plots should be grouped due to multiple model fits along the sequence (only returned in case of density plots for the `"perrySeqModel"` and `"perrySparseLTS"` methods).

`includeSE` a logical indicating whether error bars based on standard errors are available (only returned in case of line plots or dot plots).

`mapping` default aesthetic mapping for the plots.

`facets` default faceting formula for the plots (only returned if both estimators are requested in the `"sparseLTS"` or `"perrySparseLTS"` methods).

`tuning` a data frame containing the grid of tuning parameter values for which the optimality criterion was estimated (only returned for the `"sparseLTS"` and `"perrySparseLTS"` methods).

Author(s)

Andreas Alfons

See Also

`critPlot`, `rlars`, `grplars`, `rgrplars`, `tslarsP`, `rtslarsP`, `tslars`, `rtslars`, `sparseLTS`

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
```

```
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# extract information for plotting
setup <- setupCritPlot(fitRlars)
critPlot(setup)


## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitSparseLTS <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# extract information for plotting
setup1 <- setupCritPlot(fitSparseLTS)
critPlot(setup1)
setup2 <- setupCritPlot(fitSparseLTS, fit = "both")
critPlot(setup2)
```

`setupDiagnosticPlot` *Set up a diagnostic plot for a sequence of regression models*

Description

Extract the fitted values and residuals of a sequence of regression models (such as robust least angle regression models or sparse least trimmed squares regression models) and other useful information for diagnostic plots.

Usage

```
setupDiagnosticPlot(object, ...)

## S3 method for class 'seqModel'
setupDiagnosticPlot(object, s = NA, covArgs = list(...), ...)

## S3 method for class 'perrySeqModel'
setupDiagnosticPlot(object, ...)

## S3 method for class 'tslars'
setupDiagnosticPlot(object, p, ...)

## S3 method for class 'sparseLTS'
setupDiagnosticPlot(
  object,
  s = NA,
  fit = c("reweighted", "raw", "both"),
  covArgs = list(...),
  ...
)
```

```
## S3 method for class 'perrySparseLTS'
setupDiagnosticPlot(object, ...)
```

Arguments

object	the model fit from which to extract information.
...	additional arguments to be passed to covMcd can be specified directly instead of via covArgs.
s	for the "seqModel" method, an integer vector giving the steps of the submodels from which to extract information (the default is to use the optimal submodel). For the "sparseLTS" method, an integer vector giving the indices of the models from which to extract information (the default is to use the optimal model for each of the requested fits).
covArgs	a list of arguments to be passed to covMcd for computing robust Mahalanobis distances.
p	an integer giving the lag length for which to extract information (the default is to use the optimal lag length).
fit	a character string specifying from which fit to extract information. Possible values are "reweighted" (the default) to convert the reweighted fit, "raw" to convert the raw fit, or "both" to convert both fits.

Value

An object of class "setupDiagnosticPlot" with the following components:

data a data frame containing the columns listed below.

- step the steps (for the "seqModel" method) or indices (for the "sparseLTS" method) of the models (only returned if more than one model is requested).
- fit the model fits (only returned if both the reweighted and raw fit are requested in the "sparseLTS" method).
- index the indices of the observations.
- fitted the fitted values.
- residual the standardized residuals.
- theoretical the corresponding theoretical quantiles from the standard normal distribution.
- qqd the absolute distances from a reference line through the first and third sample and theoretical quartiles.
- rd the robust Mahalanobis distances computed via the MCD (see covMcd).
- xyd the pairwise maxima of the absolute values of the standardized residuals and the robust Mahalanobis distances, divided by the respective other outlier detection cutoff point.
- weight the weights indicating regression outliers.
- leverage logicals indicating leverage points (i.e., outliers in the predictor space).
- Diagnostics a factor with levels "Potential outlier" (potential regression outliers) and "Regular observation" (data points following the model).

qqLine a data frame containing the intercepts and slopes of the respective reference lines to be displayed in residual Q-Q plots.

`q` a data frame containing the quantiles of the Mahalanobis distribution used as cutoff points for detecting leverage points.

`facets` default faceting formula for the diagnostic plots (only returned where applicable).

Author(s)

Andreas Alfons

See Also

diagnosticPlot, rlars, grplars, rgrplars, tslarsP, rtslarsP, tslars, rtslars, sparseLTS

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points


## robust LARS
# fit model
fitRlars <- rlars(x, y, sMax = 10)
# extract information for plotting
setup <- setupDiagnosticPlot(fitRlars)
diagnosticPlot(setup)


## sparse LTS
# fit model
fitSparseLTS <- sparseLTS(x, y, lambda = 0.05, mode = "fraction")
# extract information for plotting
setup1 <- setupDiagnosticPlot(fitSparseLTS)
diagnosticPlot(setup1)
setup2 <- setupDiagnosticPlot(fitSparseLTS, fit = "both")
diagnosticPlot(setup2)
```

sparseLTS *Sparse least trimmed squares regression*

Description

Compute least trimmed squares regression with an L_1 penalty on the regression coefficients, which allows for sparse model estimates.

Usage

```
sparseLTS(x, ...)

## S3 method for class 'formula'
sparseLTS(formula, data, ...)

## Default S3 method:
sparseLTS(
  x,
  y,
  lambda,
  mode = c("lambda", "fraction"),
  alpha = 0.75,
  normalize = TRUE,
  intercept = TRUE,
  nsamp = c(500, 10),
  initial = c("sparse", "hyperplane", "random"),
  ncstep = 2,
  use.correction = TRUE,
  tol = .Machine$double.eps^0.5,
  eps = .Machine$double.eps,
  use.Gram,
  crit = c("BIC", "PE"),
  splits = foldControl(),
  cost = rtmspe,
  costArgs = list(),
  selectBest = c("hastie", "min"),
  seFactor = 1,
  ncores = 1,
  cl = NULL,
  seed = NULL,
  model = TRUE,
  ...
)
```

Arguments

x	a numeric matrix containing the predictor variables.

...	additional arguments to be passed down.
formula	a formula describing the model.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which sparseLTS is called.
y	a numeric vector containing the response variable.
lambda	a numeric vector of non-negative values to be used as penalty parameter.
mode	a character string specifying the type of penalty parameter. If "lambda", lambda gives the grid of values for the penalty parameter directly. If "fraction", the smallest value of the penalty parameter that sets all coefficients to 0 is first estimated based on bivariate winsorization, then lambda gives the fractions of that estimate to be used (hence all values of lambda should be in the interval [0,1] in that case).
alpha	a numeric value giving the percentage of the residuals for which the L_1 penalized sum of squares should be minimized (the default is 0.75).
normalize	a logical indicating whether the predictor variables should be normalized to have unit L_2 norm (the default is TRUE). Note that normalization is performed on the subsamples rather than the full data set.
intercept	a logical indicating whether a constant term should be included in the model (the default is TRUE).
nsamp	a numeric vector giving the number of subsamples to be used in the two phases of the algorithm. The first element gives the number of initial subsamples to be used. The second element gives the number of subsamples to keep after the first phase of ncstep C-steps. For those remaining subsets, additional C-steps are performed until convergence. The default is to first perform ncstep C-steps on 500 initial subsamples, and then to keep the 10 subsamples with the lowest value of the objective function for additional C-steps until convergence.
initial	a character string specifying the type of initial subsamples to be used. If "sparse", the lasso fit given by three randomly selected data points is first computed. The corresponding initial subsample is then formed by the fraction alpha of data points with the smallest squared residuals. Note that this is optimal from a robustness point of view, as the probability of including an outlier in the initial lasso fit is minimized. If "hyperplane", a hyperplane through p randomly selected data points is first computed, where p denotes the number of variables. The corresponding initial subsample is then again formed by the fraction alpha of data points with the smallest squared residuals. Note that this cannot be applied if p is larger than the number of observations. Nevertheless, the probability of including an outlier increases with increasing dimension p. If "random", the initial subsamples are given by a fraction alpha of randomly selected data points. Note that this leads to the largest probability of including an outlier.
ncstep	a positive integer giving the number of C-steps to perform on all subsamples in the first phase of the algorithm (the default is to perform two C-steps).
use.correction	currently ignored. Small sample correction factors may be added in the future.
tol	a small positive numeric value giving the tolerance for convergence.

eps — a small positive numeric value used to determine whether the variability within a variable is too small (an effective zero).

use.Gram — a logical indicating whether the Gram matrix of the explanatory variables should be precomputed in the lasso fits on the subsamples. If the number of variables is large, computation may be faster when this is set to FALSE. The default is to use TRUE if the number of variables is smaller than the number of observations in the subsamples and smaller than 100, and FALSE otherwise.

crit — a character string specifying the optimality criterion to be used for selecting the final model. Possible values are "BIC" for the Bayes information criterion and "PE" for resampling-based prediction error estimation. This is ignored if lambda contains only one value of the penalty parameter, as selecting the optimal value is trivial in that case.

splits — an object giving data splits to be used for prediction error estimation (see perryTuning). This is only relevant if selecting the optimal lambda via prediction error estimation.

cost — a cost function measuring prediction loss (see perryTuning for some requirements). The default is to use the root trimmed mean squared prediction error (see cost). This is only relevant if selecting the optimal lambda via prediction error estimation.

costArgs — a list of additional arguments to be passed to the prediction loss function cost. This is only relevant if selecting the optimal lambda via prediction error estimation.

selectBest, seFactor — arguments specifying a criterion for selecting the best model (see perryTuning). The default is to use a one-standard-error rule. This is only relevant if selecting the optimal lambda via prediction error estimation.

ncores — a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to NA, all available processor cores are used. For prediction error estimation, parallel computing is implemented on the R level using package **parallel**. Otherwise parallel computing is implemented on the C++ level via OpenMP (https://www.openmp.org/).

cl — a **parallel** cluster for parallel computing as generated by makeCluster. This is preferred over ncores for prediction error estimation, in which case ncores is only used on the C++ level for computing the final model.

seed — optional initial seed for the random number generator (see .Random.seed). On parallel R worker processes for prediction error estimation, random number streams are used and the seed is set via clusterSetRNGStream.

model — a logical indicating whether the data x and y should be added to the return object. If intercept is TRUE, a column of ones is added to x to account for the intercept.

Value

If crit is "PE" and lambda contains more than one value of the penalty parameter, an object of class "perrySparseLTS" (inheriting from class "perryTuning", see perryTuning). It contains

information on the prediction error criterion, and includes the final model with the optimal tuning paramter as component `finalModel`.

Otherwise an object of class `"sparseLTS"` with the following components:

`lambda` a numeric vector giving the values of the penalty parameter.

`best` an integer vector or matrix containing the respective best subsets of h observations found and used for computing the raw estimates.

`objective` a numeric vector giving the respective values of the sparse LTS objective function, i.e., the L_1 penalized sums of the h smallest squared residuals from the raw fits.

`coefficients` a numeric vector or matrix containing the respective coefficient estimates from the reweighted fits.

`fitted.values` a numeric vector or matrix containing the respective fitted values of the response from the reweighted fits.

`residuals` a numeric vector or matrix containing the respective residuals from the reweighted fits.

`center` a numeric vector giving the robust center estimates of the corresponding reweighted residuals.

`scale` a numeric vector giving the robust scale estimates of the corresponding reweighted residuals.

`cnp2` a numeric vector giving the respective consistency factors applied to the scale estimates of the reweighted residuals.

`wt` an integer vector or matrix containing binary weights that indicate outliers from the respective reweighted fits, i.e., the weights are 1 for observations with reasonably small reweighted residuals and 0 for observations with large reweighted residuals.

`df` an integer vector giving the respective degrees of freedom of the obtained reweighted model fits, i.e., the number of nonzero coefficient estimates.

`intercept` a logical indicating whether the model includes a constant term.

`alpha` a numeric value giving the percentage of the residuals for which the L_1 penalized sum of squares was minimized.

`quan` the number h of observations used to compute the raw estimates.

`raw.coefficients` a numeric vector or matrix containing the respective coefficient estimates from the raw fits.

`raw.fitted.values` a numeric vector or matrix containing the respective fitted values of the response from the raw fits.

`raw.residuals` a numeric vector or matrix containing the respective residuals from the raw fits.

`raw.center` a numeric vector giving the robust center estimates of the corresponding raw residuals.

`raw.scale` a numeric vector giving the robust scale estimates of the corresponding raw residuals.

`raw.cnp2` a numeric value giving the consistency factor applied to the scale estimate of the raw residuals.

`raw.wt` an integer vector or matrix containing binary weights that indicate outliers from the respective raw fits, i.e., the weights used for the reweighted fits.

`crit` an object of class `"bicSelect"` containing the BIC values and indicating the final model (only returned if argument `crit` is `"BIC"` and argument `lambda` contains more than one value for the penalty parameter).

`x` the predictor matrix (if `model` is `TRUE`).

`y` the response variable (if `model` is `TRUE`).

`call` the matched function call.

Note

The underlying C++ code uses the C++ library Armadillo. From package version 0.6.0, the back end for sparse least trimmed squares from package **sparseLTSEigen**, which uses the C++ library Eigen, is no longer supported and can no longer be used.

Parallel computing is implemented via OpenMP (https://www.openmp.org/).

Author(s)

Andreas Alfons

References

Alfons, A., Croux, C. and Gelper, S. (2013) Sparse least trimmed squares regression for analyzing high-dimensional large data sets. *The Annals of Applied Statistics*, **7**(1), 226–248.

See Also

coef, fitted, plot, predict, residuals, weights, ltsReg

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points

## fit sparse LTS model for one value of lambda
sparseLTS(x, y, lambda = 0.05, mode = "fraction")

## fit sparse LTS models over a grid of values for lambda
frac <- seq(0.2, 0.05, by = -0.05)
sparseLTS(x, y, lambda = frac, mode = "fraction")
```

`standardize` *Data standardization*

Description

Standardize data with given functions for computing center and scale.

Usage

```
standardize(x, centerFun = mean, scaleFun = sd)

robStandardize(
  x,
  centerFun = median,
  scaleFun = mad,
  fallback = FALSE,
  eps = .Machine$double.eps,
  ...
)
```

Arguments

`x`	a numeric vector, matrix or data frame to be standardized.
`centerFun`	a function to compute an estimate of the center of a variable (defaults to `mean`).
`scaleFun`	a function to compute an estimate of the scale of a variable (defaults to `sd`).
`fallback`	a logical indicating whether standardization with `mean` and `sd` should be performed as a fallback mode for variables whose robust scale estimate is too small. This is useful, e.g., for data containing dummy variables.
`eps`	a small positive numeric value used to determine whether the robust scale estimate of a variable is too small (an effective zero).
`...`	currently ignored.

Details

`robStandardize` is a wrapper function for robust standardization, hence the default is to use `median` and `mad`.

Value

An object of the same type as the original data `x` containing the centered and scaled data. The center and scale estimates of the original data are returned as attributes `"center"` and `"scale"`, respectively.

Note

The implementation contains special cases for the typically used combinations `mean`/`sd` and `median`/`mad` in order to reduce computation time.

Author(s)

Andreas Alfons

See Also

scale, sweep

Examples

```
## generate data
set.seed(1234)     # for reproducibility
x <- rnorm(10)     # standard normal
x[1] <- x[1] * 10  # introduce outlier

## standardize data
x
standardize(x)     # mean and sd
robStandardize(x)  # median and MAD
```

TopGear *Top Gear car data*

Description

The data set contains information on cars featured on the website of the popular BBC television show *Top Gear*.

Usage

```
data("TopGear")
```

Format

A data frame with 297 observations on the following 32 variables.

`Maker` factor; the car maker.

`Model` factor; the car model.

`Type` factor; the exact model type.

`Fuel` factor; the type of fuel (`"Diesel"` or `"Petrol"`).

`Price` numeric; the list price (in UK pounds)

`Cylinders` numeric; the number of cylinders in the engine.

`Displacement` numeric; the displacement of the engine (in cc).

`DriveWheel` factor; the type of drive wheel (`"4WD"`, `"Front"` or `"Rear"`).

`BHP` numeric; the power of the engine (in bhp).

`Torque` numeric; the torque of the engine (in lb/ft).

`Acceleration` numeric; the time it takes the car to get from 0 to 62 mph (in seconds).

`TopSpeed` numeric; the car's top speed (in mph).

`MPG` numeric; the combined fuel consuption (urban + extra urban; in miles per gallon).

`Weight` numeric; the car's curb weight (in kg).

`Length` numeric; the car's length (in mm).

`Width` numeric; the car's width (in mm).

`Height` numeric; the car's height (in mm).

`AdaptiveHeadlights` factor; whether the car has adaptive headlights (`"no"`, `"optional"` or `"standard"`).

`AdjustableSteering` factor; whether the car has adjustable steering (`"no"` or `"standard"`).

`AlarmSystem` factor; whether the car has an alarm system (`"no/optional"` or `"standard"`).

`Automatic` factor; whether the car has an automatic transmission (`"no"`, `"optional"` or `"standard"`).

`Bluetooth` factor; whether the car has bluetooth (`"no"`, `"optional"` or `"standard"`).

`ClimateControl` factor; whether the car has climate control (`"no"`, `"optional"` or `"standard"`).

`CruiseControl` factor; whether the car has cruise control (`"no"`, `"optional"` or `"standard"`).

`ElectricSeats` factor; whether the car has electric seats (`"no"`, `"optional"` or `"standard"`).

`Leather` factor; whether the car has a leather interior (`"no"`, `"optional"` or `"standard"`).

`ParkingSensors` factor; whether the car has parking sensors (`"no"`, `"optional"` or `"standard"`).

`PowerSteering` factor; whether the car has power steering (`"no"` or `"standard"`).

`SatNav` factor; whether the car has a satellite navigation system (`"no"`, `"optional"` or `"standard"`).

`ESP` factor; whether the car has ESP (`"no"`, `"optional"` or `"standard"`).

`Verdict` numeric; review score between 1 (lowest) and 10 (highest).

`Origin` factor; the origin of the car maker (`"Asia"`, `"Europe"` or `"USA"`).

Source

The data were scraped from `http://www.topgear.com/uk/` on 2014-02-24. Variable `Origin` was added based on the car maker information.

Examples

```
data("TopGear")
summary(TopGear)
```

tsBlocks *Construct predictor blocks for time series models*

Description

Construct blocks of original and lagged values for autoregressive time series models with exogenous inputs. The typical use case is to supply the output as `newdata` argument to the `predict` method of robust groupwise least angle regression models.

Usage

```
tsBlocks(x, y, p = 2, subset = NULL, intercept = TRUE)
```

Arguments

`x`	a numeric matrix or data frame containing the exogenous predictor series.
`y`	a numeric vector containing the response series.
`p`	an integer giving the number of lags to include (defaults to 2).
`subset`	a logical or integer vector defining a subset of observations from which to construct the matrix of predictor blocks.
`intercept`	a logical indicating whether a column of ones should be added to the matrix of predictor blocks to account for the intercept.

Value

A matrix containing blocks of original and lagged values of the time series `y` and `x`.

Author(s)

Andreas Alfons

See Also

`predict.tslars`, `tslars`, `predict.tslarsP`, `tslarsP`

tslars *(Robust) least angle regression for time series data*

Description

(Robustly) sequence groups of candidate predictors and their respective lagged values according to their predictive content and find the optimal model along the sequence. Note that lagged values of the response are included as a predictor group as well.

Usage

```
tslars(x, ...)

## S3 method for class 'formula'
tslars(formula, data, ...)

## Default S3 method:
tslars(
  x,
  y,
  h = 1,
  pMax = 3,
  sMax = NA,
  fit = TRUE,
  s = c(0, sMax),
  crit = "BIC",
  ncores = 1,
  cl = NULL,
  model = TRUE,
  ...
)

rtslars(x, ...)

## S3 method for class 'formula'
rtslars(formula, data, ...)

## Default S3 method:
rtslars(
  x,
  y,
  h = 1,
  pMax = 3,
  sMax = NA,
  centerFun = median,
  scaleFun = mad,
  regFun = lmrob,
```

```
  regArgs = list(),
  combine = c("min", "euclidean", "mahalanobis"),
  winsorize = FALSE,
  const = 2,
  prob = 0.95,
  fit = TRUE,
  s = c(0, sMax),
  crit = "BIC",
  ncores = 1,
  cl = NULL,
  seed = NULL,
  model = TRUE,
  ...
)
```

Arguments

x	a numeric matrix or data frame containing the candidate predictor series.
...	additional arguments to be passed down.
formula	a formula describing the full model.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which tslars or rtslars is called.
y	a numeric vector containing the response series.
h	an integer giving the forecast horizon (defaults to 1).
pMax	an integer giving the maximum number of lags in the model (defaults to 3).
sMax	an integer giving the number of predictor series to be sequenced. If it is NA (the default), predictor groups are sequenced as long as there are twice as many observations as predictor variables.
fit	a logical indicating whether to fit submodels along the sequence (TRUE, the default) or to simply return the sequence (FALSE).
s	an integer vector of length two giving the first and last step along the sequence for which to compute submodels. The default is to start with a model containing only an intercept (step 0) and iteratively add all series along the sequence (step sMax). If the second element is NA, predictor groups are added to the model as long as there are twice as many observations as predictor variables. If only one value is supplied, it is recycled.
crit	a character string specifying the optimality criterion to be used for selecting the final model. Currently, only "BIC" for the Bayes information criterion is implemented.
ncores	a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to NA, all available processor cores are used. For each lag length, parallel computing for obtaining the data cleaning weights and for fitting models along the sequence is implemented on the R level using package **parallel**. Otherwise parallel computing for

some of of the more computer-intensive computations in the sequencing step is implemented on the C++ level via OpenMP (https://www.openmp.org/).

`cl` a **parallel** cluster for parallel computing as generated by `makeCluster`. This is preferred over `ncores` for tasks that are parallelized on the R level, in which case `ncores` is only used for tasks that are parallelized on the C++ level.

`model` a logical indicating whether the model data should be included in the returned object.

`centerFun` a function to compute a robust estimate for the center (defaults to `median`).

`scaleFun` a function to compute a robust estimate for the scale (defaults to `mad`).

`regFun` a function to compute robust linear regressions that can be interpreted as weighted least squares (defaults to `lmrob`).

`regArgs` a list of arguments to be passed to `regFun`.

`combine` a character string specifying how to combine the data cleaning weights from the robust regressions with each predictor group. Possible values are `"min"` for taking the minimum weight for each observation, `"euclidean"` for weights based on Euclidean distances of the multivariate set of standardized residuals (i.e., multivariate winsorization of the standardized residuals assuming independence), or `"mahalanobis"` for weights based on Mahalanobis distances of the multivariate set of standardized residuals (i.e., multivariate winsorization of the standardized residuals).

`winsorize` a logical indicating whether to clean the data by multivariate winsorization.

`const` numeric; tuning constant for multivariate winsorization to be used in the initial corralation estimates based on adjusted univariate winsorization (defaults to 2).

`prob` numeric; probability for the quantile of the χ^2 distribution to be used in multivariate winsorization (defaults to 0.95).

`seed` optional initial seed for the random number generator (see `.Random.seed`), which is useful because many robust regression functions (including `lmrob`) involve randomness. On parallel R worker processes, random number streams are used and the seed is set via `clusterSetRNGStream`.

Value

If `fit` is `FALSE`, an integer matrix in which each column contains the indices of the sequenced predictor series for the corresponding lag length.

Otherwise an object of class `"tslars"` with the following components:

`pFit` a list containing the fits for the respective lag lengths (see `tslarsP`).

`pOpt` an integer giving the optimal number of lags.

`pMax` the maximum number of lags considered.

`x` the matrix of candidate predictor series (if `model` is `TRUE`).

`y` the response series (if `model` is `TRUE`).

`call` the matched function call.

Note

The predictor group of lagged values of the response is indicated by the index 0.

Author(s)

Andreas Alfons, based on code by Sarah Gelper

References

Alfons, A., Croux, C. and Gelper, S. (2016) Robust groupwise least angle regression. *Computational Statistics & Data Analysis*, **93**, 421–435.

See Also

coef, fitted, plot, predict, residuals, tslarsP, lmrob

`tslarsP` *(Robust) least angle regression for time series data with fixed lag length*

Description

(Robustly) sequence groups of candidate predictors and their respective lagged values according to their predictive content and find the optimal model along the sequence. Note that lagged values of the response are included as a predictor group as well.

Usage

```
tslarsP(x, ...)

## S3 method for class 'formula'
tslarsP(formula, data, ...)

## Default S3 method:
tslarsP(
  x,
  y,
  h = 1,
  p = 2,
  sMax = NA,
  fit = TRUE,
  s = c(0, sMax),
  crit = "BIC",
  ncores = 1,
  cl = NULL,
  model = TRUE,
  ...
```

```
)

rtslarsP(x, ...)

## S3 method for class 'formula'
rtslarsP(formula, data, ...)

## Default S3 method:
rtslarsP(
  x,
  y,
  h = 1,
  p = 2,
  sMax = NA,
  centerFun = median,
  scaleFun = mad,
  regFun = lmrob,
  regArgs = list(),
  combine = c("min", "euclidean", "mahalanobis"),
  winsorize = FALSE,
  const = 2,
  prob = 0.95,
  fit = TRUE,
  s = c(0, sMax),
  crit = "BIC",
  ncores = 1,
  cl = NULL,
  seed = NULL,
  model = TRUE,
  ...
)
```

Arguments

x	a numeric matrix or data frame containing the candidate predictor series.
...	additional arguments to be passed down.
formula	a formula describing the full model.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which tslarsP or rtslarsP is called.
y	a numeric vector containing the response series.
h	an integer giving the forecast horizon (defaults to 1).
p	an integer giving the number of lags in the model (defaults to 2).
sMax	an integer giving the number of predictor series to be sequenced. If it is NA (the default), predictor groups are sequenced as long as there are twice as many observations as predictor variables.

`fit` a logical indicating whether to fit submodels along the sequence (`TRUE`, the default) or to simply return the sequence (`FALSE`).

`s` an integer vector of length two giving the first and last step along the sequence for which to compute submodels. The default is to start with a model containing only an intercept (step 0) and iteratively add all series along the sequence (step `sMax`). If the second element is `NA`, predictor groups are added to the model as long as there are twice as many observations as predictor variables. If only one value is supplied, it is recycled.

`crit` a character string specifying the optimality criterion to be used for selecting the final model. Currently, only `"BIC"` for the Bayes information criterion is implemented.

`ncores` a positive integer giving the number of processor cores to be used for parallel computing (the default is 1 for no parallelization). If this is set to `NA`, all available processor cores are used. For obtaining the data cleaning weights and for fitting models along the sequence, parallel computing is implemented on the R level using package **parallel**. Otherwise parallel computing for some of of the more computer-intensive computations in the sequencing step is implemented on the C++ level via OpenMP (`https://www.openmp.org/`).

`cl` a **parallel** cluster for parallel computing as generated by `makeCluster`. This is preferred over `ncores` for tasks that are parallelized on the R level, in which case `ncores` is only used for tasks that are parallelized on the C++ level.

`model` a logical indicating whether the model data should be included in the returned object.

`centerFun` a function to compute a robust estimate for the center (defaults to `median`).

`scaleFun` a function to compute a robust estimate for the scale (defaults to `mad`).

`regFun` a function to compute robust linear regressions that can be interpreted as weighted least squares (defaults to `lmrob`).

`regArgs` a list of arguments to be passed to `regFun`.

`combine` a character string specifying how to combine the data cleaning weights from the robust regressions with each predictor group. Possible values are `"min"` for taking the minimum weight for each observation, `"euclidean"` for weights based on Euclidean distances of the multivariate set of standardized residuals (i.e., multivariate winsorization of the standardized residuals assuming independence), or `"mahalanobis"` for weights based on Mahalanobis distances of the multivariate set of standardized residuals (i.e., multivariate winsorization of the standardized residuals).

`winsorize` a logical indicating whether to clean the data by multivariate winsorization.

`const` numeric; tuning constant for multivariate winsorization to be used in the initial corralation estimates based on adjusted univariate winsorization (defaults to 2).

`prob` numeric; probability for the quantile of the χ^2 distribution to be used in multivariate winsorization (defaults to 0.95).

`seed` optional initial seed for the random number generator (see `.Random.seed`), which is useful because many robust regression functions (including `lmrob`) involve randomness. On parallel R worker processes, random number streams are used and the seed is set via `clusterSetRNGStream`.

Value

If `fit` is `FALSE`, an integer vector containing the indices of the sequenced predictor series.

Otherwise an object of class `"tslarsP"` (inheriting from classes `"grplars"` and `"seqModel"`) with the following components:

`active` an integer vector containing the sequence of predictor series.

`s` an integer vector containing the steps for which submodels along the sequence have been computed.

`coefficients` a numeric matrix in which each column contains the regression coefficients of the corresponding submodel along the sequence.

`fitted.values` a numeric matrix in which each column contains the fitted values of the corresponding submodel along the sequence.

`residuals` a numeric matrix in which each column contains the residuals of the corresponding submodel along the sequence.

`df` an integer vector containing the degrees of freedom of the submodels along the sequence (i.e., the number of estimated coefficients).

`robust` a logical indicating whether a robust fit was computed.

`scale` a numeric vector giving the robust residual scale estimates for the submodels along the sequence (only returned for a robust fit).

`crit` an object of class `"bicSelect"` containing the BIC values and indicating the final model (only returned if argument `crit` is `"BIC"` and argument `s` indicates more than one step along the sequence).

`muX` a numeric vector containing the center estimates of the predictor variables.

`sigmaX` a numeric vector containing the scale estimates of the predictor variables.

`muY` numeric; the center estimate of the response.

`sigmaY` numeric; the scale estimate of the response.

`x` the matrix of candidate predictor series (if `model` is `TRUE`).

`y` the response series (if `model` is `TRUE`).

`assign` an integer vector giving the predictor group to which each predictor variable belongs.

`w` a numeric vector giving the data cleaning weights (only returned for a robust fit).

`h` the forecast horizon.

`p` the number of lags in the model.

`call` the matched function call.

Note

The predictor group of lagged values of the response is indicated by the index 0.

Author(s)

Andreas Alfons, based on code by Sarah Gelper

References

Alfons, A., Croux, C. and Gelper, S. (2016) Robust groupwise least angle regression. *Computational Statistics & Data Analysis*, **93**, 421–435.

See Also

coef, fitted, plot, predict, residuals, tslars, lmrob

`weights.sparseLTS` *Extract outlier weights from sparse LTS regression models*

Description

Extract binary weights that indicate outliers from sparse least trimmed squares regression models.

Usage

```
## S3 method for class 'sparseLTS'
weights(
  object,
  type = "robustness",
  s = NA,
  fit = c("reweighted", "raw", "both"),
  drop = !is.null(s),
  ...
)
```

Arguments

`object`	the model fit from which to extract outlier weights.
`type`	the type of weights to be returned. Currently only robustness weights are implemented (`"robustness"`).
`s`	an integer vector giving the indices of the models for which to extract outlier weights. If `fit` is `"both"`, this can be a list with two components, with the first component giving the indices of the reweighted fits and the second the indices of the raw fits. The default is to use the optimal model for each of the requested estimators. Note that the optimal models may not correspond to the same value of the penalty parameter for the reweighted and the raw estimator.
`fit`	a character string specifying for which estimator to extract outlier weights. Possible values are `"reweighted"` (the default) for weights indicating outliers from the reweighted fit, `"raw"` for weights indicating outliers from the raw fit, or `"both"` for the outlier weights from both estimators.
`drop`	a logical indicating whether to reduce the dimension to a vector in case of only one model.
`...`	currently ignored.

Value

A numeric vector or matrix containing the requested outlier weights.

Note

The weights are 1 for observations with reasonably small residuals and 0 for observations with large residuals.

Author(s)

Andreas Alfons

See Also

sparseLTS

Examples

```
## generate data
# example is not high-dimensional to keep computation time low
library("mvtnorm")
set.seed(1234)  # for reproducibility
n <- 100  # number of observations
p <- 25   # number of variables
beta <- rep.int(c(1, 0), c(5, p-5))  # coefficients
sigma <- 0.5      # controls signal-to-noise ratio
epsilon <- 0.1    # contamination level
Sigma <- 0.5^t(sapply(1:p, function(i, j) abs(i-j), 1:p))
x <- rmvnorm(n, sigma=Sigma)    # predictor matrix
e <- rnorm(n)                   # error terms
i <- 1:ceiling(epsilon*n)       # observations to be contaminated
e[i] <- e[i] + 5                # vertical outliers
y <- c(x %*% beta + sigma * e)  # response
x[i,] <- x[i,] + 5              # bad leverage points

## sparse LTS over a grid of values for lambda
# fit model
frac <- seq(0.2, 0.05, by = -0.05)
fitGrid <- sparseLTS(x, y, lambda = frac, mode = "fraction")
# extract outlier weights
weights(fitGrid)
head(weights(fitGrid, fit = "both"))
head(weights(fitGrid, s = NULL))
head(weights(fitGrid, fit = "both", s = NULL))
```

`winsorize` *Data cleaning by winsorization*

Description

Clean data by means of winsorization, i.e., by shrinking outlying observations to the border of the main part of the data.

Usage

```
winsorize(x, ...)

## Default S3 method:
winsorize(
  x,
  standardized = FALSE,
  centerFun = median,
  scaleFun = mad,
  const = 2,
  return = c("data", "weights"),
  ...
)

## S3 method for class 'matrix'
winsorize(
  x,
  standardized = FALSE,
  centerFun = median,
  scaleFun = mad,
  const = 2,
  prob = 0.95,
  tol = .Machine$double.eps^0.5,
  return = c("data", "weights"),
  ...
)

## S3 method for class 'data.frame'
winsorize(x, ...)
```

Arguments

`x`	a numeric vector, matrix or data frame to be cleaned.
`...`	for the generic function, additional arguments to be passed down to methods. For the `"data.frame"` method, additional arguments to be passed down to the `"matrix"` method. For the other methods, additional arguments to be passed down to `robStandardize`.

standardized	a logical indicating whether the data are already robustly standardized.
centerFun	a function to compute a robust estimate for the center to be used for robust standardization (defaults to median). Ignored if standardized is TRUE.
scaleFun	a function to compute a robust estimate for the scale to be used for robust standardization (defaults to mad). Ignored if standardized is TRUE.
const	numeric; tuning constant to be used in univariate winsorization (defaults to 2).
return	character string; if standardized is TRUE, this specifies the type of return value. Possible values are "data" for returning the cleaned data, or "weights" for returning data cleaning weights.
prob	numeric; probability for the quantile of the χ^2 distribution to be used in multivariate winsorization (defaults to 0.95).
tol	a small positive numeric value used to determine singularity issues in the computation of correlation estimates based on bivariate winsorization (see corHuber).

Details

The borders of the main part of the data are defined on the scale of the robustly standardized data. In the univariate case, the borders are given by $+/-$const, thus a symmetric distribution is assumed. In the multivariate case, a normal distribution is assumed and the data are shrunken towards the boundary of a tolerance ellipse with coverage probability prob. The boundary of this ellipse is thereby given by all points that have a squared Mahalanobis distance equal to the quantile of the χ^2 distribution given by prob.

Value

If standardize is TRUE and return is "weights", a set of data cleaning weights. Multiplying each observation of the standardized data by the corresponding weight yields the cleaned standardized data.

Otherwise an object of the same type as the original data x containing the cleaned data is returned.

Note

Data cleaning weights are only meaningful for standardized data. In the general case, the data need to be standardized first, then the data cleaning weights can be computed and applied to the standardized data, after which the cleaned standardized data need to be backtransformed to the original scale.

Author(s)

Andreas Alfons, based on code by Jafar A. Khan, Stefan Van Aelst and Ruben H. Zamar

References

Khan, J.A., Van Aelst, S. and Zamar, R.H. (2007) Robust linear model selection based on least angle regression. *Journal of the American Statistical Association*, **102**(480), 1289–1299.

See Also

corHuber

Examples

```
## generate data
set.seed(1234)     # for reproducibility
x <- rnorm(10)     # standard normal
x[1] <- x[1] * 10  # introduce outlier

## winsorize data
x
winsorize(x)
```

Index