Package: RSC (via r-universe)

June 6, 2026

Type Package

Title Robust and Sparse Correlation Matrix

Description Performs robust and sparse correlation matrix estimation. Robustness is achieved based on a simple robust pairwise correlation estimator, while sparsity is obtained based on thresholding. The optimal thresholding is tuned via cross-validation. See Serra, Coretto, Fratello and Tagliaferri (2018) <doi:10.1093/bioinformatics/btx642>.

Author Luca Coraggio [cre, aut], Pietro Coretto [aut], Angela Serra [aut], Roberto Tagliaferri [ctb]

Maintainer Luca Coraggio <luca.coraggio@unina.it>

NeedsCompilation yes

Imports stats, graphics, Matrix, methods, parallel, foreach, doParallel, utils

License GPL (>= 2)

Version 2.0.5

Date 2025-09-09

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

Date/Publication 2025-09-09 12:40:13 UTC

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

RemoteRef HEAD

RemoteSha 28218d537e7f8a0864612abe99685e77cd854e77

Contents

plot.rsc_cv *Plot method for rsc_cv objects*

Description

Plot the cross-validation estimates of the Frobenius loss.

Usage

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

Arguments

x	Output from rsc_cv, that is an S3 object of class "rsc_cv".
...	additional arguments passed to plot.default.

Value

Plot the Frobenius loss estimated via cross-validation (y-axis) vs threshold values (x-axis). The dotted blue line represents the average expected normalized Frobenius loss, while the vertical segments around the average are *1-standard-error* error bars (see *Details* in rsc_cv.

The vertical dashed red line identifies the minimum of the average loss, that is the optimal threshold flagged as "minimum". The vertical dashed green line identifies the optimal selection flagged as "minimum1se" in the output of rsc_cv (see *Details* in rsc_cv).

References

Serra, A., Coretto, P., Fratello, M., and Tagliaferri, R. (2018). Robust and sparsecorrelation matrix estimation for the analysis of high-dimensional genomics data. *Bioinformatics*, 34(4), 625-634. doi:10.1093/bioinformatics/btx642

See Also

rsc_cv

Examples

```
## simulate a random sample from a multivariate Cauchy distribution
## note: example in high-dimension are obtained increasing p
set.seed(1)
n   <- 100  # sample size
p   <- 10   # dimension
dat <- matrix(rt(n*p, df = 1), nrow = n, ncol = p)
colnames(dat) <- paste0("Var", 1:p)


## perform 10-fold cross-validation repeated R=10 times
```

```
## note: for multi-core machines experiment with 'ncores'
set.seed(2)
a <- rsc_cv(x = dat, R = 10, K = 10, ncores = 1)
a

## plot the cross-validation estimates
plot(a)

## pass additional parameters to graphics::plot
plot(a , cex = 2)
```

---

rmad *RMAD correlation matrix*

Description

Compute the RMAD robust correlation matrix proposed in Serra et al. (2018) based on the robust correlation coefficient proposed in Pasman and Shevlyakov (1987).

Usage

```
rmad(x , y = NULL, na.rm = FALSE , even.correction = FALSE, num.threads = "half-max")
```

Arguments

x	A numeric vector, a matrix or a data.frame. If x is a matrix or a data.frame, rows of x correspond to sample units and columns correspond to variables. If x is a numerical vector, and y is not NULL, the RMAD correlation coefficient between x and y is computed. Categorical variables are not allowed.
y	A numerical vector if not NULL. If both x and y are numerical vectors, the RMAD correlation coefficient between x and y is computed.
na.rm	A logical value, if TRUE sample observation containing NA values are excluded (see *Details*).
even.correction	A logical value, if TRUE a correction for the calculation of the medians is applied to reduce the bias when the number of samples even (see *Details*).
num.threads	An integer value or the string "half-max" (default), specifying the number of threads for parallel execution (see *Details*).

Details

The rmad function computes the correlation matrix based on the pairwise robust correlation coefficient of Pasman and Shevlyakov (1987). This correlation coefficient is based on repeated median calculations for all pairs of variables. This is a computational intensive task when the number of variables (that is ncol(x)) is large.

The software is optimized for large dimensional data sets, the median is approximated as the central observation obtained based on the *find* algorithm of Hoare (1961) (also known as *quickselect*) implemented in C language. For small samples this may be a crude approximation, however, it makes the computational cost feasible for high-dimensional data sets. With the option `even.correction = TRUE` a correction is applied to reduce the bias for data sets with an even number of samples. Although `even.correction = TRUE` has a small computational cost for each pair of variables, it is suggested to use the default `even.correction = FALSE` for large dimensional data sets.

The function can handle a data matrix with missing values (`NA` records). If `na.rm = TRUE` then missing values are handled by casewise deletion (and if there are no complete cases, an error is returned). In practice, if `na.rm = TRUE` all rows of `x` that contain at least an `NA` are removed.

Since the software is optimized to work with high-dimensional data sets, the output RMAD matrix is packed into a storage efficient format using the `"dspMatrix"` S4 class from the `Matrix` package. The latter is specifically designed for dense real symmetric matrices. A sparse correlation matrix can be obtained applying thresholding using the `rsc_cv` and `rsc`.

`rmad` function supports parallel execution. This is provided via *openmp* (http://www.openmp.org), which must be already available on the system at installation time; otherwise, falls back to single-core execution. For later installation of openmp, the RSC package needs to be re-installed (re-compiled) to provide multi-threads execution. If `num.threads > 0`, function is executed using `min(num.threads, max.threads)` threads, where `max.threads` is the maximum number of available threads. That is, if positive the specified number of threads (up to the maximum available) are used. If `num.threads < 0`, function is executed using `max(max.threads - num.threads, 1)` threads, i.e. when negative `num.threads` indicates the number of threads not to use (at least one thread is used). If `num.threads == 0`, a single thread is used (equivalent to `num.threads = 1`). If `num.threads == "half-max"`, function is executed using half of the available threads (`max(max.threads/2, 1)`). This is the default.

Value

If `x` is a matrix or a data.frame, returns a correlation matrix of class `"dspMatrix"` (S4 class object) as defined in the `Matrix` package.

If `x` and `y` are numerical vectors, returns a numerical value, that is the RMAD correlation coefficient between `x` and `y`.

References

Hoare, C. A. (1961). Algorithm 65: find. *Communications of the ACM*, 4(7), 321-322.

Musser, D. R. (1997). Introspective sorting and selection algorithms. *Software: Practice and Experience*, 27(8), 983-993.

Pasman,V. and Shevlyakov,G. (1987). Robust methods of estimation of correlation coefficient. *Automation Remote Control*, 48, 332-340.

Serra, A., Coretto, P., Fratello, M., and Tagliaferri, R. (2018). Robust and sparsecorrelation matrix estimation for the analysis of high-dimensional genomics data. *Bioinformatics*, 34(4), 625-634. doi: 10.1093/bioinformatics/btx642

See Also

`rsc_cv`, `rsc`

Examples

```
## simulate a random sample from a multivariate Cauchy distribution
set.seed(1)
n   <- 100    # sample size
p   <- 7      # dimension
dat <- matrix(rt(n*p, df = 1), nrow = n, ncol = p)
colnames(dat) <- paste0("Var", 1:p)


## compute the rmad correlation coefficient between dat[,1] and dat[,2]
a <- rmad(x = dat[,1], y = dat[,2])


## compute the RMAD correlaiton matrix
b <- rmad(x = dat)
b
```

`rsc` *Robust and Sparse Correlation Matrix Estimator*

Description

Compute the Robust and Sparse Correlation Matrix (RSC) estimator proposed in Serra et al. (2018).

Usage

```
rsc(cv, threshold = "minimum")
```

Arguments

`cv`	An S3 object of class `"rsc_cv"` (see `rsc_cv`).
`threshold`	Threshold parameter to compute the RSC estimate. This is a numeric value taken onto the interval (0,1), or it is equal to `"minimum"` or `"minimum1se"` for selecting the optimal threshold according to the selection performed in `rsc_cv`.

Details

The setting `threshold = "minimum"` or `threshold = "minimum1se"` applies thresholding according to the criteria discussed in the *Details* section in `rsc_cv`. When `cv` is obtained using `rsc_cv` with `cv.type = "random"`, the default settings for `rsc` implements exactly the RSC estimator proposed in Serra et al., (2018).

Although `threshold = "minimum"` is the default choice, in high-dimensional situations `threshold = "minimum1se"` usually provides a more parsimonious representation of the correlation structure. Since the underlying RMAD matrix is passed through the `cv` input, any other hand-tuned threshold to the RMAD matrix can be applied without significant additional computational costs. The latter can be done setting `threshold` to any value onto the (0,1) interval.

The software is optimized to handle high-dimensional data sets, therefore, the output RSC matrix is packed into a storage efficient sparse format using the `"dsCMatrix"` S4 class from the `Matrix` package. The latter is specifically designed for sparse real symmetric matrices.

Value

Returns a sparse correlaiton matrix of class "dsCMatrix" (S4 class object) as defined in the Matrix package.

References

Serra, A., Coretto, P., Fratello, M., and Tagliaferri, R. (2018). Robust and sparsecorrelation matrix estimation for the analysis of high-dimensional genomics data. *Bioinformatics*, 34(4), 625-634. doi:10.1093/bioinformatics/btx642

See Also

rsc_cv

Examples

```
## simulate a random sample from a multivariate Cauchy distribution
## note: example in high-dimension are obtained increasing p
set.seed(1)
n   <- 100  # sample size
p   <- 10   # dimension
dat <- matrix(rt(n*p, df = 1), nrow = n, ncol = p)
colnames(dat) <- paste0("Var", 1:p)


## perform 10-fold cross-validation repeated R=10 times
## note: for multi-core machines experiment with 'ncores'
set.seed(2)
a <- rsc_cv(x = dat, R = 10, K = 10, ncores = 1)
a

## obtain the RSC matrix with "minimum" flagged solution
b <- rsc(cv = a, threshold = "minimum")
b

## obtain the RSC matrix with "minimum1se" flagged solution
d <- rsc(cv = a, threshold = "minimum1se")
d

## since the object 'a' stores the RMAD underlying estimator, we can
## apply thresholding at any level without re-estimating the RMAD
## matrix
e <- rsc(cv = a, threshold = 0.5)
e
```

---

rsc_cv *Optimal threshold selection for the RSC estimator*

Description

Perform cross-validation to select an adaptive optimal threshold for the RSC estimator proposed in Serra et al. (2018).

Usage

```
rsc_cv(x, cv.type = "kfold", R = 10, K = 10, threshold = seq(0.05, 0.95, by = 0.025),
       even.correction = FALSE, na.rm = FALSE, ncores = NULL, monitor = TRUE)
```

Arguments

x	A matrix or a data.frame. Rows of x correspond to sample units and columns correspond to variables. Categorical variables are not allowed.
cv.type	A character string indicating the cross-validation algorithm. Possible values are "kfold" for repeated K-fold cross-validation, and "random" for random cross-validation (see *Details*).
R	An integer corresponding to the number of repeated foldings when cv.type = "kfold". When cv.type = "random" R defines the number of random splits (see *Details*).
K	An integer corresponding to the number of *folds* in K-fold cross-validation. Therefore this argument is not relevant when cv.type = "random".
threshold	A sequence of reals taken onto the interval (0,1) defining the threshold values at which the loss is estimated.
even.correction	A logical value. It sets the parameter even.correction in each of the underlying RMAD computations (see *Details* in rmad).
na.rm	A logical value, it defines the treatment of missing values in each of the underlying RMAD computations (see *Details*).
ncores	An integer value defining the number of cores used for parallel computing. When ncores=NULL (default), the number r of available cores is detected, and (r-1) of them are used (see *Details*).
monitor	A logical value. If TRUE progress messages are printed on screen.

Details

The rsc_cv function performs cross-validation to estimate the expected Frobenius loss proposed in Bickel and Levina (2008). The original contribution of Bickel and Levina (2008), and its extension in Serra et al. (2018), is based on a random cross-validation algorithm where the training/test size depends on the sample size *n*. The latter is implemented selecting cv.type = "ramdom", and fixing an appropriate number R of random train/test splits. R should be as large as possible, but in practice this impacts the computing time strongly for high-dimensional data sets.

Although Serra et al. (2018) showed that the random cross-validation of Bickel and Levina (2008) works well for the RSC estimator, subsequent experiments suggested that repeated K-fold cross-validation on average produces better results. Repeated K-fold cross-validation is implemented with the default cv.type = "kfold". In this case K defines the number of *folds*, while R defines the

number of times that the K-fold cross-validation is repeated with R independent shuffles of the original data. Selecting R=1 and K=10 one performs the standard 10-fold cross-validation. Ten replicates (R=10) of the K-fold cross-validation are generally sufficient to obtain reasonable estimates of the underlying loss, but for extremely high-dimensional data R may be varied to speed up calculations.

On multi-core hardware the cross-validation is executed in parallel setting ncores. The parallelism is implemented on the total number of data splits, that is R for the random cross-validation, and R*K for the repeated K-fold cross-validation. The software is optimized so that generally the total computing time scales almost linearly with the number of available computer cores (ncores).

For both the random and the K-fold cross-validation it is computed the normalized version of the expected squared Frobenius loss proposed in Bickel and Levina (2008). The normalization is such that the squared Frobenius norm of the identity matrix equals to 1 whatever is its dimension.

Two optimal threshold selection types are reported with flags (see *Value* section below): "minimum" and "minimum1se". The flag "minimum" denotes the threshold value that minimizes the average loss. The flag "minimum1se" implements the so called *1-SE rule*: this is the maximum threshold value such that the corresponding average loss is within *1-standard-error* with respect to the threshold that minimizes the average loss (that is the one corresponding to the "minimum" flag).

Since unbiased standard errors for the K-fold cross-validation are impossible to compute (see Bengio and Grandvalet, 2004), when cv.type="kfold" the reported standard errors have to be considered as a downward biased approximation.

Value

An S3 object of class 'cv_rsc' with the following components:

rmadvec	A vector containing the lower triangle of the underlying RMAD matrix.
varnames	A character vector if variable names are available for the input data set x. Otherwise this is NULL.
loss	A data.frame reporting cross-validation estimates. Columns of loss are as follows: loss$Threshold is the threshold value; loss$Average is averaged loss; loss$SE is the standard error for the average loss; loss$Flag="minimum" denotes the threshold achieving the minimum average loss; loss$Flag = "*" denotes threshold values such that the average loss is within *1-standard-error* with respect to the "minimum" solution.
minimum	A numeric value. This is the minimum of the average loss. This corresponds to the flag "minimum" in the loss component above (see *Details*).
minimum1se	A numeric value. This is the largest threshold such that the corresponding flag = "*". In practice this selects the optimal threshold based on the *1-SE rule* discussed in the *Details* Section above.

References

Bengio, Y., and Grandvalet, Y. (2004). No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5(Sep), 1089-1105.

Bickel, P. J., and Levina, E. (2008). Covariance regularization by thresholding. The *Annals of Statistics*, 36(6), 2577-2604. doi:10.1214/08-AOS600

Serra, A., Coretto, P., Fratello, M., and Tagliaferri, R. (2018). Robust and sparsecorrelation matrix estimation for the analysis of high-dimensional genomics data. *Bioinformatics*, 34(4), 625-634. doi:10.1093/bioinformatics/btx642

See Also

rsc, plot.rsc_cv

Examples

```
## simulate a random sample from a multivariate Cauchy distribution
## note: example in high-dimension are obtained increasing p
set.seed(1)
n   <- 100  # sample size
p   <- 10   # dimension
dat <- matrix(rt(n*p, df = 1), nrow = n, ncol = p)
colnames(dat) <- paste0("Var", 1:p)


## perform 10-fold cross-validation repeated R=10 times
## note: for multi-core machines experiment with 'ncores'
set.seed(2)
a <- rsc_cv(x = dat, R = 10, K = 10, ncores = 1)
a


## threshold selection: note that here, knowing the sampling designs,
## we would like to threshold any correlation larger than zero in
## absolute value
##
a$minimum        ## "minimum"    flagged solution
a$minimum1se     ## "minimum1se" flagged solution

## plot the cross-validation estimates
plot(a)

## to obtain the RSC matrix we pass 'a' to the rsc() function
b <- rsc(cv = a, threshold = "minimum")
b

d <- rsc(cv = a, threshold = "minimum1se")
d

## since the object 'a' stores the RMAD underlying estimator, we can
## apply thresholding at any level without re-estimating the RMAD
## matrix
e <- rsc(cv = a, threshold = 0.5)
e
```

Index