Package: multiblock (via r-universe)

October 21, 2026

Encoding UTF-8

Type Package

Title Multiblock Data Fusion in Statistics and Machine Learning

Version 0.8.12

Date 2026-xx-xx

Description Functions and datasets to support Smilde, Næs and Liland (2021, ISBN: 978-1-119-60096-1) ``Multiblock Data Fusion in Statistics and Machine Learning - Applications in the Natural and Life Sciences". This implements and imports a large collection of methods for multiblock data analysis with common interfaces, result- and plotting functions, several real data sets and six vignettes covering a range different applications.

License GPL (>= 2)

URL https://khliland.github.io/multiblock/, https://github.com/khliland/multiblock/

BugReports https://github.com/khliland/multiblock/issues/

Depends R (>= 3.5.0)

Imports ade4, car, HDANOVA (>= 0.8.2), MASS, mixlm, plotrix, pls, plsVarSel, pracma, progress, Rcpp, RSpectra, SSBtools

Suggests EMSC, FactoMineR, geigen, RGCCA (>= 3.0.0), r.jive, rmarkdown, knitr

LinkingTo Rcpp, RcppEigen

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev

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

Date/Publication 2026-10-21 16:55:32 UTC

RemoteUrl https://github.com/khliland/multiblock

RemoteRef HEAD

RemoteSha e43e75738679212450f24df75bdc5c46c8e16847

Contents

basic . . . 3
block.data.frame . . . 4
candies . . . 4
cca . . . 5
complex . . . 6
compnames . . . 6
disco . . . 7
DISCOsca . . . 8
dummycode . . . 9
emulsions . . . 10
explvar . . . 10
extended.model.frame . . . 11
gca . . . 12
gpa . . . 13
gsvd . . . 14
hogsvd . . . 15
hpca . . . 16
ifa . . . 17
jive . . . 18
lpls . . . 19
lpls_results . . . 21
lplsData . . . 23
maage . . . 24
mbpls . . . 26
mbrda . . . 28
mcoa . . . 29
mcolors . . . 30
mfa . . . 31
mobile . . . 32
multiblock_plots . . . 33
multiblock_results . . . 36
mvrVal . . . 37
path_effects . . . 40
pca . . . 42
pcagca . . . 43
plot.path_effects . . . 45
popls . . . 46
potato . . . 48
predict.mbpls . . . 49
preprocess . . . 50
print.path_effects . . . 52
rosa . . . 53
rosa_plots . . . 55
rosa_results . . . 56
sca . . . 59
simulated . . . 60

smbpls 61
SO_TDI 64
sopls 66
sopls_plots 68
sopls_results 72
statis 75
supervised 76
unique_combos 77
unsupervised 78
wine 79

Index **80**

`basic` *Single- and Two-Block Methods*

Description

This documentation covers a range of single- and two-block methods. In particular:

- PCA - Principal Component Analysis (`pca`)
- PCR - Principal Component Regression (`pcr`)
- PLSR - Partial Least Squares Regression (`plsr`)
- CCA - Canonical Correlation Analysis (`cca`)
- IFA - Interbattery Factor Analysis (`ifa`)
- GSVD - Generalized SVD (`gsvd`)

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`.

Examples

```
data(potato)
X <- potato$Chemical
y <- potato$Sensory[,1,drop=FALSE]

pca.pot  <- pca(X, ncomp = 2)
pcr.pot  <- pcr(y ~ X, ncomp = 2)
pls.pot  <- plsr(y ~ X, ncomp = 2)
cca.pot  <- cca(potato[1:2])
ifa.pot  <- ifa(potato[1:2])
gsvd.pot <- gsvd(lapply(potato[3:4], t))
```

block.data.frame *Block-wise indexable data.frame*

Description

This is a convenience function for making `data.frame`s that are easily indexed on a block-wise basis.

Usage

```
block.data.frame(X, block_inds = NULL, to.matrix = TRUE)
```

Arguments

`X`	Either a single `data.frame` to index or a `list` of matrices/data.frames
`block_inds`	Named `list` of indexes if `X` is a single `data.frame`, otherwise `NULL`.
`to.matrix`	`logical` indicating if input list elements should be converted to matrices.

Value

A `data.frame` which can be indexed block-wise.

Examples

```
# Random data
M <- matrix(rnorm(200), nrow = 10)
# .. with dimnames
dimnames(M) <- list(LETTERS[1:10], as.character(1:20))

# A named list for indexing
inds <- list(B1 = 1:10, B2 = 11:20)

X <- block.data.frame(M, inds)
str(X)
```

candies *Sensory assessment of candies.*

Description

A dataset containing 9 sensory attributes for 5 candies assessed by 11 trained assessors.

Usage

```
data(candies)
```

Format

A data.frame having 165 rows and 3 variables:

assessment Matrix of sensory attributes

assessor Factor of assessors

candy Factor of candies

References

Luciano G, Næs T. Interpreting sensory data by combining principal component analysis and analysis of variance. Food Qual Prefer. 2009;20(3):167-175.

`cca` *Canonical Correlation Analysis - CCA*

Description

This is a wrapper for the `stats::cancor` function for computing CCA.

Usage

```
cca(X)
```

Arguments

`X`	`list` of input data blocks.

Details

CCA is a method which maximises correlation between linear combinations of the columns of two blocks, i.e. max(cor(X1 x a, X2 x b)). This is done sequentially with deflation in between, such that a sequence of correlations and weight vectors a and b are associated with a pair of matrices.

Value

`multiblock` object with associated with printing, scores, loadings. Relevant plotting functions: `multiblock_plots` and result functions: `multiblock_results`.

References

Hotelling, H. (1936) Relations between two sets of variates. Biometrika, 28, 321–377.

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`. Common functions for computation and extraction of results and plotting are found in `multiblock_results` and `multiblock_plots`, respectively.

Examples

```
data(potato)
X <- potato$Chemical

cca.pot  <- cca(potato[1:2])
```

---

complex *Methods With Complex Linkage*

Description

This documentation covers a few complex methods. In particular:

- L-PLS - Partial Least Squares in L configuration (`lpls`)
- SO-PLS-PM - Sequential and Orthogonalised PLS Path Modeling (`sopls_pm`)

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`.

Examples

```
# L-PLS
sim <- lplsData(I = 30, N = 20, J = 5, K = 6, ncomp = 2)
X1  <- sim$X1; X2 <- sim$X2; X3 <- sim$X3
lp  <- lpls(X1,X2,X3) # exo-L-PLS
```

---

compnames *Vector of component names*

Description

Convenience function for creating a vector of component names based on the dimensions the input object (`matrix` or object having a `score` function).

Usage

```
compnames(object, comps, explvar = FALSE, ...)
```

Arguments

`object`	An object fitted using the multiblock package.
`comps`	`integer` vector of components.
`explvar`	`logical` indicating if explained variances should be included.
`...`	Unused

Details

This is a copy of `compnames` from the `pls` package to work with `multiblock` objects.

Value

A `character` vector of component names.

`disco` *Distinctive and Common Components with SCA - DISCO*

Description

This is a wrapper for the `DISCOsca` function by Zhengguo Gu for computing DISCO.

Usage

```
disco(X, ncomp = 2, ...)
```

Arguments

`X`	`list` of input blocks.
`ncomp`	`integer` number of components to extract.
`...`	additional arguments (not used).

Details

DISCO is a restriction of SCA where Alternating Least Squares is used for estimation of loadings and scores. The SCA solution is rotated towards loadings (in sample linked mode) which are filled with zeros in a pattern resembling distinct, local and common components. When used in sample linked mode and only selecting distinct components, it shares a resemblance to SO-PLS, only in an unsupervised setting. Explained variances are computed as proportion of block variation explained by scores*loadings'.

Value

`multiblock` object including relevant scores and loadings. Relevant plotting functions: `multiblock_plots` and result functions: `multiblock_results`.

References

Schouteden, M., Van Deun, K., Wilderjans, T. F., & Van Mechelen, I. (2014). Performing DISCO-SCA to search for distinctive and common information in linked data. Behavior research methods, 46(2), 576-587.

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`.

Examples

```
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.disco  <- disco(potList)
plot(scores(pot.disco), labels="names")
```

DISCOsca *DISCO-SCA rotation.*

Description

A DISCO-SCA procedure for identifying common and distinctive components. The code is adapted from the orphaned RegularizedSCA package by Zhengguo Gu.

Usage

```
DISCOsca(DATA, R, Jk)
```

Arguments

DATA	A matrix, which contains the concatenated data with the same subjects from multiple blocks. Note that each row represents a subject.
R	Number of components (R>=2).
Jk	A vector containing number of variables in the concatenated data matrix.

Value

Trot_best	Estimated component score matrix (i.e., T)
Prot_best	Estimated component loading matrix (i.e., P)
comdist	A matrix representing common distinctive components. (Rows are data blocks and columns are components.) 0 in the matrix indicating that the corresponding component of that block is estimated to be zeros, and 1 indicates that (at least one component loading in) the corresponding component of that block is not zero. Thus, if a column in the comdist matrix contains only 1's, then this column is a common component, otherwise distinctive component.
propExp_component	Proportion of variance per component.

References

Schouteden, M., Van Deun, K., Wilderjans, T. F., & Van Mechelen, I. (2014). Performing DISCO-SCA to search for distinctive and common information in linked data. Behavior research methods, 46(2), 576-587.

Examples

```
## Not run:
DATA1 <- matrix(rnorm(50), nrow=5)
DATA2 <- matrix(rnorm(100), nrow=5)
DATA <- cbind(DATA1, DATA2)
R <- 5
Jk <- c(10, 20)
DISCOsca(DATA, R, Jk)

## End(Not run)
```

---

dummycode *Dummy-coding of a single vector*

Description

Flexible dummy-coding allowing for all R's built-in types of contrasts and optional dropping of a factor level to reduce rank defficiency probability.

Usage

```
dummycode(Y, contrast = "contr.sum", drop = TRUE)
```

Arguments

Y	`vector` to dummy code.
contrast	Contrast type, default = "contr.sum".
drop	`logical` indicating if one level should be dropped (default = TRUE).

Value

`matrix` made by dummy-coding the input vector.

Examples

```
vec <- c("a","a","b","b","c","c")
dummycode(vec)
```

emulsions *Spectroscopic analysis of emulsions*

Description

A dataset containing NIR and Raman measurements of emulsions in a food model system of fat, water and protein mixtures.

Usage

```
data(emulsions)
```

Format

A data.frame having 69 rows and 3 variables:

Raman Matrix of Raman measurements

NIR Matrix of Near-infrared measurements

PUFA Matrix of polyunsaturated fatty acid contents as a percentage of the sample and as a percentage of the total fat content.

References

N.K. Afseth, V.H. Segtnan, B.J. Marquardt, and J.P. Wold. Raman and Near-Infrared Spectroscopy for Quantification of Fat Composition in a Complex Food Model System. Applied Spectroscopy 2005;59(11):1324-1332.

explvar *Explained predictor variance*

Description

Extraction and/or computation of explained variances for various object classes in the `multiblock` package.

Usage

```
explvar(object)
```

Arguments

`object`	An object fitted using a method from the multiblock package

Value

A `vector` of component-wise explained variances for predictors.

Examples

```
data(potato)
so <- sopls(Sensory ~ Chemical + Compression, data=potato, ncomp=c(10,10),
            max_comps=10)
explvar(so)
```

`extended.model.frame` *Extracting the Extended Model Frame from a Formula or Fit*

Description

This function attempts to apply `model.frame` and extend the result with columns of interactions.

Usage

```
extended.model.frame(formula, data, ..., sep = ".")
```

Arguments

`formula`	a model formula or terms object or an R object.
`data`	a data.frame, list or environment (see `model.frame`).
`...`	further arguments to pass to `model.frame`.
`sep`	separator in contraction of names for interactions (default = ".").

Value

A `data.frame` that includes everything a `model.frame` does plus interaction terms.

Examples

```
dat <- data.frame(Y = c(1,2,3,4,5,6),
                  X = factor(LETTERS[c(1,1,2,2,3,3)]),
                  W = factor(letters[c(1,2,1,2,1,2)]))
extended.model.frame(Y ~ X*W, dat)
```

gca *Generalized Canonical Analysis - GCA*

Description

This is an interface to both SVD-based (default) and RGCCA-based GCA (wrapping the `RGCCA::rgcca` function)

Usage

```
gca(X, ncomp = "max", svd = TRUE, tol = 10^-12, corrs = TRUE, ...)
```

Arguments

`X`	`list` of input blocks.
`ncomp`	`integer` number of components to extract, either single integer (equal for all blocks), vector (individual per block) or 'max' for maximum possible number of components.
`svd`	`logical` indicating if Singular Value Decomposition approach should be used (default=TRUE).
`tol`	`numeric` tolerance for component inclusion (singular values).
`corrs`	`logical` indicating if correlations should be calculated for RGCCA based approach.
`...`	additional arguments for RGCCA approach.

Details

GCA is a generalisation of Canonical Correlation Analysis to handle three or more blocks. There are several ways to generalise, and two of these are available through `gca`. The default is an SVD based approach estimating a common subspace and measuring mean squared correlation to this. An alternative approach is available through RGCCA. For the SVD based approach, the `ncomp` parameter controls the block-wise decomposition while the following the consensus decomposition is limited to the minimum number of components from the individual blocks.

Value

`multiblock` object including relevant scores and loadings. Relevant plotting functions: `multiblock_plots` and result functions: `multiblock_results`. `blockCoef` contains canonical coefficients, while `blockDecomp` contains decompositions of each block.

References

- Carroll, J. D. (1968). Generalization of canonical correlation analysis to three or more sets of variables. Proceedings of the American Psychological Association, pages 227-22.
- Van der Burg, E. and Dijksterhuis, G. (1996). Generalised canonical analysis of individual sensory profiles and instrument data, Elsevier, pp. 221–258.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.gca <- gca(potList)
plot(scores(pot.gca), labels="names")
```

---

gpa *Generalized Procrustes Analysis - GPA*

Description

This is a wrapper for the FactoMineR::GPA function for computing GPA.

Usage

```
gpa(X, graph = FALSE, ...)
```

Arguments

X	list of input blocks.
graph	logical indicating if decomposition should be plotted.
...	additional arguments for RGCCA approach.

Details

GPA is a generalisation of Procrustes analysis, where one matrix is scaled and rotated to be as similar as possible to another one. Through the generalisation, individual scaling and rotation of each input matrix is performed against a common representation which is estimated in an iterative manner.

Value

multiblock object including relevant scores and loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

Gower, J. C. (1975). Generalized procrustes analysis. Psychometrika. 40: 33–51.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.gpa    <- gpa(potList)
plot(scores(pot.gpa), labels="names")
```

---

gsvd *Generalised Singular Value Decomposition - GSVD*

Description

This is a wrapper for the geigen::gsvd function for computing GSVD.

Usage

```
gsvd(X)
```

Arguments

X	list of input data blocks.

Details

GSVD is a generalisation of SVD to two variable-linked matrices where common loadings and block-wise scores are estimated.

Value

multiblock object with associated with printing, scores, loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

Van Loan, C. (1976) Generalizing the singular value decomposition. SIAM Journal on Numerical Analysis, 13, 76–83.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
X <- potato$Chemical

gsvd.pot <- gsvd(lapply(potato[3:4], t))
```

---

hogsvd *Higher Order Generalized SVD - HOGSVD*

Description

This is a simple implementation for computing HOGSVD

Usage

```
hogsvd(X)
```

Arguments

X	list of input blocks.

Details

HOGSVD is a generalisation of SVD to two or more blocks. It finds a common set of loadings across blocks and individual sets of scores per block.

Value

multiblock object including relevant scores and loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

Ponnapalli, S. P., Saunders, M. A., Van Loan, C. F., & Alter, O. (2011). A higher-order generalized singular value decomposition for comparison of global mRNA expression from multiple organisms. PloS one, 6(12), e28072.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(candies)
candyList <- lapply(1:nlevels(candies$candy),function(x)candies$assessment[candies$candy==x,])
can.hogsvd <- hogsvd(candyList)
scoreplot(can.hogsvd, block=1, labels="names")
```

---

hpca *Hierarchical Principal component analysis - HPCA*

Description

This is a wrapper for the RGCCA::rgcca function for computing HPCA.

Usage

```
hpca(X, ncomp = 2, scale = FALSE, verbose = FALSE, ...)
```

Arguments

X	list of input blocks.
ncomp	integer number of components to extract.
scale	logical indicating if variables should be scaled.
verbose	logical indicating if diagnostic information should be printed.
...	additional arguments for RGCCA.

Details

HPCA is a hierarchical PCA analysis which combines two or more blocks into a two-level decomposition with block-wise loadings and scores and superlevel common loadings and scores. The method is closely related to the supervised method MB-PLS in structure.

Value

multiblock object including relevant scores and loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

Westerhuis, J.A., Kourti, T., and MacGregor,J.F. (1998). Analysis of multiblock and hierarchical PCA and PLS models. Journal of Chemometrics, 12, 301–321.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.hpca   <- hpca(potList)
plot(scores(pot.hpca), labels="names")
```

---

ifa *Inter-battery Factor Analysis - IFA*

Description

This is a wrapper for the RGCCA::rgcca function for computing IFA.

Usage

```
ifa(X, ncomp = 1, scale = FALSE, verbose = FALSE, ...)
```

Arguments

X	list of input data blocks.
ncomp	integer number of principal components to return.
scale	logical indicating if variables should be standardised (default=FALSE).
verbose	logical indicating if intermediate results should be printed.
...	additional arguments to RGCCA::rgcca.

Details

IFA rotates two matrices to align one or more factors against each other, maximising correlations.

Value

multiblock object with associated with printing, scores, loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

Tucker, L. R. (1958). An inter-battery method of factor analysis. Psychometrika, 23(2), 111-136.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
X <- potato$Chemical

ifa.pot  <- ifa(potato[1:2])
```

---

jive *Joint and Individual Variation Explained - JIVE*

Description

This is a wrapper for the r.jive::jive function for computing JIVE.

Usage

```
jive(X, ...)
```

Arguments

X	list of input blocks.
...	additional arguments for r.jive::jive.

Details

Jive performs a decomposition of the variation in two or more blocks into low-dimensional representations of individual and joint variation plus residual variation.

Value

multiblock object including relevant scores and loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

Lock, E., Hoadley, K., Marron, J., and Nobel, A. (2013) Joint and individual variation explained (JIVE) for integrated analysis of multiple data types. Ann Appl Stat, 7 (1), 523–542.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex.

Examples

```
# Too time consuming for testing
 data(candies)
candyList <- lapply(1:nlevels(candies$candy),function(x)candies$assessment[candies$candy==x,])
 can.jive  <- jive(candyList)
 summary(can.jive)
```

---

lpls *L-PLS regression*

Description

Simultaneous decomposition of three blocks connected in an L pattern.

Usage

```
lpls(
  X1,
  X2,
  X3,
  ncomp = 2,
  doublecenter = TRUE,
  scale = c(FALSE, FALSE, FALSE),
  type = c("exo"),
  impute = FALSE,
  niter = 25,
  subsetX2 = NULL,
  subsetX3 = NULL,
  ...
)
```

Arguments

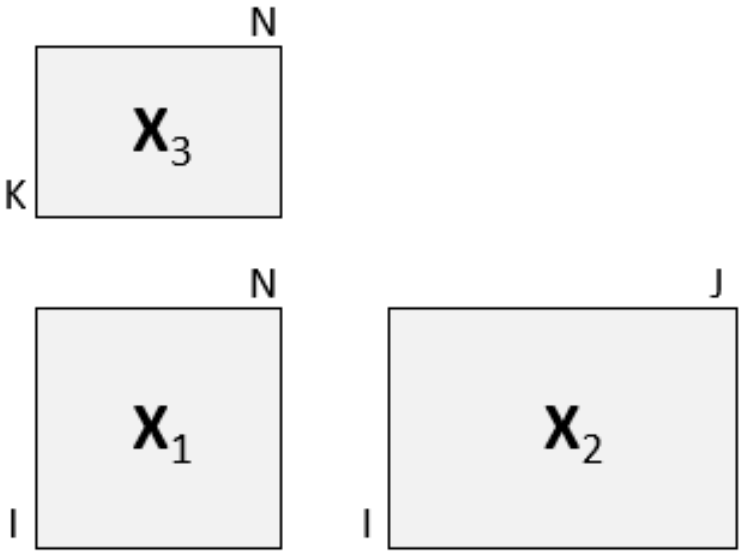
X1	matrix of size IxN (middle matrix)
X2	matrix of size IxJ (left matrix)
X3	matrix of size KxN (top matrix)
ncomp	number of L-PLS components
doublecenter	logical indicating if centering should be done both ways for X1 (default=TRUE)
scale	logical vector of length three indicating if each of the matrices should be autoscaled.
type	character indicating type of L-PLS ("exo"=default, "exo_ort" or "endo")
impute	logical indicating if SVD-based imputation of missing data is required.
niter	numeric giving number of iterations in component extraction loop.

subsetX2	vector defining optional sub-setting of X2 data.
subsetX3	vector defining optional sub-setting of X3 data.
...	Additional arguments, not used.

Details

Two versions of L-PLS are available: exo- and endo-L-PLS which assume an outward or inward relationship between the main block X1 and the two other blocks X2 and X3.

The exo_ort algorithm returns orthogonal scores and should be chosen for visual exploration in correlation loading plots. If exo-L-PLS with prediction is the main purpose of the model then the non-orthogonal exo type L-PLS should be chosen for which the predict function has prediction implemented.



Value

An object of type lpls and multiblock containing all results from the L-PLS analysis. The object type lpls is associated with functions for correlation loading plots, prediction and cross-validation. The type multiblock is associated with the default functions for result presentation (multiblock_results) and plotting (multiblock_plots).

Author(s)

Solve Sæbø (adapted by Kristian Hovde Liland)

References

- Martens, H., Anderssen, E., Flatberg, A.,Gidskehaug, L.H., Høy, M., Westad, F.,Thybo, A., and Martens, M. (2005). Regression of a data matrix on descriptors of both its rows and of its columns via latent variables: L-PLSR. Computational Statistics & Data Analysis, 48(1), 103 – 123.

- Sæbø, S., Almøy, T., Flatberg, A., Aastveit, A.H., and Martens, H. (2008). LPLS-regression: a method for prediction and classification under the influence of background information on predictor variables. Chemometrics and Intelligent Laboratory Systems, 91, 121–132.
- Sæbø, S., Martens, M. and Martens H. (2010) Three-block data modeling by endo- and exo-LPLS regression. In Handbook of Partial Least Squares: Concepts, Methods and Applications. Esposito Vinzi, V.; Chin, W.W.; Henseler, J.; Wang, H. (Eds.). Springer.

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`. Functions for computation and extraction of results and plotting are found in `lpls_results`.

Examples

```
# Simulate data set
sim <- lplsData(I = 30, N = 20, J = 5, K = 6, ncomp = 2)
X1  <- sim$X1; X2 <- sim$X2; X3 <- sim$X3
lp  <- lpls(X1,X2,X3) # exo-L-PLS
```

---

lpls_results *Result functions for L-PLS objects (*`lpls`*)*

Description

Correlation loading plot, prediction and cross-validation for L-PLS models with class `lpls`.

Usage

```
## S3 method for class 'lpls'
plot(
  x,
  comps = c(1, 2),
  doplot = c(TRUE, TRUE, TRUE),
  level = c(2, 2, 2),
  arrow = c(1, 0, 1),
  xlim = c(-1, 1),
  ylim = c(-1, 1),
  samplecol = 4,
  pathcol = 2,
  varcol = "grey70",
  varsize = 1,
  sampleindex = 1:dim(x$corloadings$R22)[1],
  pathindex = 1:dim(x$corloadings$R3)[1],
  varindex = 1:dim(x$corloadings$R21)[1],
  ...
```

```
)

## S3 method for class 'lpls'
predict(
  object,
  X1new = NULL,
  X2new = NULL,
  X3new = NULL,
  exo.direction = c("X2", "X3"),
  ...
)

lplsCV(object, segments1 = NULL, segments2 = NULL, trace = TRUE)
```

Arguments

x	lpls object
comps	integer vector of components.
doplot	logical indicating if plotting should be performed.
level	integer vector of length 3 for selecting plot symbol. 1=dots. 2=dimnames.
arrow	integer vector of length 3 indicating arrows (1) or not (0).
xlim	numeric x limits.
ylim	numeric y limits.
samplecol	character for sample colours.
pathcol	character for third colour.
varcol	character for variable colours.
varsize	numeric size of symbols for variables.
sampleindex	integer for selecting samples.
pathindex	integer for selecting in third direction.
varindex	integer for selecting variables.
...	Not implemented.
object	lpls object.
X1new	matrix of new X1 samples.
X2new	matrix of new X2 samples.
X3new	matrix of new X3 samples.
exo.direction	character selecting "X2" or "X3" prediction.
segments1	list of sample segments.
segments2	list of variable segments.
trace	logical indicating if verbose mode should be selected.

Value

Nothing is return for plotting (plot.lpls), predicted values are returned for predictions (predict.lpls) and cross-validation metrics are returned for for cross-validation (lplsCV).

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex.

Examples

```
# Simulate data set
sim <- lplsData(I = 30, N = 20, J = 5, K = 6, ncomp = 2)
X1  <- sim$X1; X2 <- sim$X2; X3 <- sim$X3

# exo-L-PLS:
lp.exo  <- lpls(X1,X2,X3, ncomp = 2)
# Predict X1
pred.exo.X2 <- predict(lp.exo, X1new = X1, exo.direction = "X2")
# Predict X3
pred.exo.X2 <- predict(lp.exo, X1new = X1, exo.direction = "X3")

# endo-L-PLS:
lp.endo <- lpls(X1,X2,X3, ncomp = 2, type = "endo")
# Predict X1 from X2 and X3 (in this case fitted values):
pred.endo.X1 <- predict(lp.endo, X2new = X2, X3new = X3)

# LOO cross-validation horizontally
lp.cv1 <- lplsCV(lp.exo, segments1 = as.list(1:dim(X1)[1]))

# LOO cross-validation vertically
lp.cv2 <- lplsCV(lp.exo, segments2 = as.list(1:dim(X1)[2]))

# Three-fold CV, horizontal
lp.cv3 <- lplsCV(lp.exo, segments1 = as.list(1:10, 11:20, 21:30))
```

`lplsData` *L-PLS data simulation for exo-type analysis*

Description

Three data blocks are simulated to express covariance in an exo-L-PLS direction (see lpls. Dimensionality and number of underlying components can be controlled.

Usage

```
lplsData(I = 30, N = 20, J = 5, K = 6, ncomp = 2)
```

Arguments

I	numeric number of rows of X1 and X2
N	numeric number of columns in X1 and X3

J	numeric number of columns in X2
K	numeric number of rows in X3
ncomp	numeric number of latent components

Value

A list of three matrices with dimensions matching in an L-shape.

Author(s)

Solve Sæbø (adapted by Kristian Hovde Liland)

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex.

Examples

```
lp <- lplsData(I = 30, N = 20, J = 5, K = 6, ncomp = 2)
names(lp)
```

---

maage *Måge plot*

Description

Måge plot for SO-PLS (sopls) cross-validation visualisation.

Usage

```
maage(
  object,
  expl_var = TRUE,
  pure.trace = FALSE,
  pch = 20,
  xlab = "# components",
  ylab = ifelse(expl_var, "Explained variance (%)", "RMSECV"),
  xlim = NULL,
  ylim = NULL,
  cex.text = 0.8,
  ...
)

maageSeq(
  object,
```

```
  compSeq = TRUE,
  expl_var = TRUE,
  pch = 20,
  xlab = "# components",
  ylab = ifelse(expl_var, "Explained variance (%)", "RMSECV"),
  xlim = NULL,
  ylim = NULL,
  cex.text = 0.8,
  col = "gray",
  col.block = c("red", "blue", "darkgreen", "purple", "black", "red", "blue",
    "darkgreen"),
  ...
)
```

Arguments

object	An SO-PLS model (sopls object)
expl_var	Logical indicating if explained variance (default) or RMSECV should be displayed.
pure.trace	Logical indicating if single block solutions should be traced in the plot.
pch	Scalar or symbol giving plot symbol.
xlab	Label for x-axis.
ylab	Label for y-axis.
xlim	Plot limits for x-axis (numeric vector).
ylim	Plot limits for y-axis (numeric vector).
cex.text	Text scaling (scalar) for better readability of plots.
...	Additional arguments to plot.
compSeq	Integer vector giving the sequence of previous components chosen for maageSeq (see example).
col	Line colour in plot.
col.block	Line colours for blocks (default = c('red','blue','darkgreen','purple','black'))

Details

This function can either be used for global optimisation across blocks or sequential optimisation, using maageSeq. The examples below show typical usage.

Value

The maage plot has no return.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex.

Examples

```
data(wine)
ncomp <- unlist(lapply(wine, ncol))[-5]
so.wine <- sopls(`Global quality` ~ ., data=wine, ncomp=ncomp,
            max_comps=10, validation="CV", segments=10)
maage(so.wine)

# Sequential search for optimal number of components per block
old.par <- par(mfrow=c(2,2), mar=c(3,3,0.5,1), mgp=c(2,0.7,0))
maageSeq(so.wine)
maageSeq(so.wine, 2)
maageSeq(so.wine, c(2,1))
maageSeq(so.wine, c(2,1,1))
par(old.par)
```

---

mbpls *Multiblock Partial Least Squares - MB-PLS*

Description

A function computing MB-PLS scores, loadings, etc. on the super-level and block-level.

Usage

```
mbpls(
  formula,
  data,
  subset,
  na.action,
  X = NULL,
  Y = NULL,
  ncomp = 1,
  scale = FALSE,
  blockScale = c("sqrtnvar", "ssq", "none"),
  ...
)
```

Arguments

formula	Model formula accepting a single response (block) and predictor block names separated by + signs.
data	The data set to analyse.
subset	Expression for subsetting the data before modelling.
na.action	How to handle NAs (no action implemented).
X	list of input blocks. If X is supplied, the formula interface is skipped.
Y	matrix of responses.

ncomp	integer number of PLS components.
scale	logical for autoscaling inputs (default = FALSE).
blockScale	Either a character indicating type of block scaling or a numeric vector of block weights (see Details).
...	additional arguments to pls::plsr.

Details

MB-PLS is the prototypical component based supervised multiblock method. It was originally formulated as a two-level method with a block-level and a super-level, but it was later discovered that it could be expressed as an ordinary PLS on concatenated weighted X blocks followed by a simple loop for calculating block-level loading weights, loadings and scores. This implementation uses the plsr function on the scaled input blocks (1/sqrt(ncol)) enabling all summaries and plots from the pls package.

Block weighting is performed after scaling all variables and is by default "sqrtnvar": 1/sqrt(ncol(X[[i]])) in each block. Alternatives are "ssq": 1/norm(X[[i]], "F")^2 and "none": 1/1. Finally, if a numeric vector is supplied, it will be used to scale the blocks after "ssq" scaling, i.e., Z[[i]] = X[[i]] / norm(X[[i]], "F")^2 * blockScale[i].

Value

multiblock, mvr object with super-scores, super-loadings, block-scores and block-loading, and the underlying mvr (PLS) object for the super model, with all its result and plot possibilities. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

- Wangen, L.E. and Kowalski, B.R. (1988). A multiblock partial least squares algorithm for investigating complex chemical systems. Journal of Chemometrics, 3, 3–20.
- Westerhuis, J.A., Kourti, T., and MacGregor,J.F. (1998). Analysis of multiblock and hierarchical PCA and PLS models. Journal of Chemometrics, 12, 301–321.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex.

Examples

```
data(potato)
# Formula interface
mb <- mbpls(Sensory ~ Chemical+Compression, data=potato, ncomp = 5)

# ... or X and Y
mb.XY <- mbpls(X=potato[c('Chemical','Compression')], Y=potato[['Sensory']], ncomp = 5)
identical(mb$scores, mb.XY$scores)
print(mb)
scoreplot(mb, labels="names") # Exploiting mvr object structure from pls package
```

```
# Block scaling with emphasis on first block
mbs <- mbpls(Sensory ~ Chemical+Compression, data=potato, ncomp = 5, blockScale = c(10, 1))
scoreplot(mbs, labels="names") # Exploiting mvr object structure from pls package
```

`mbrda` *Multiblock Redundancy Analysis - mbRDA*

Description

This is a wrapper for the `ade4::mbpcaiv` function for computing mbRDA.

Usage

```
mbrda(formula, data, subset, na.action, X = NULL, Y = NULL, ncomp = 1, ...)
```

Arguments

`formula`	Model formula accepting a single response (block) and predictor block names separated by + signs.
`data`	The data set to analyse.
`subset`	Expression for subsetting the data before modelling.
`na.action`	How to handle NAs (no action implemented).
`X`	`list` of input blocks.
`Y`	`matrix` of responses.
`ncomp`	`integer` number of PLS components.
`...`	additional arguments to ade4::mbpcaiv.

Details

mbRDA is a multiblock formulation of Redundancy (Data) Analysis. RDA is theoretically between PLS and GCA. Like GCA, RDA does not consider correlations within X, but like PLS it does consider correlations within Y. RDA can also be viewed as a PCR of Y constrained to have scores that are also linear combinations of X. If the `adegraphics` package is attached, a nice overview can be plotted as `plot(mbr$mbpcaiv)` following the example below.

Value

`multiblock`, `mvr` object with scores, block-scores and block-loading. Relevant plotting functions: `multiblock_plots` and result functions: `multiblock_results`.

References

Bougeard, S., Qannari, E.M., Lupo, C., andHanafi, M. (2011). From Multiblock Partial Least Squares to Multiblock Redundancy Analysis. A Continuum Approach. Informatica, 22(1), 11–26.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex.

Examples

```
# Convert data.frame with AsIs objects to list of matrices
data(potato)
potatoList <- lapply(potato, unclass)

mbr <- mbrda(Sensory ~ Chemical + Compression, data = potatoList, ncomp = 10)
mbr.XY <- mbrda(X = potatoList[c('Chemical','Compression')], Y = potatoList[['Sensory']],
                ncomp = 10)
print(mbr)
scoreplot(mbr) # Exploiting mvr object structure from pls package
```

---

mcoa *Multiple Co-Inertia Analysis - MCOA*

Description

This is a wrapper for the RGCCA::rgcca function for computing MCOA.

Usage

```
mcoa(X, ncomp = 2, scale = FALSE, verbose = FALSE, ...)
```

Arguments

X	list of input blocks.
ncomp	integer number of components to extract.
scale	logical indicating if variables should be scaled.
verbose	logical indicating if diagnostic information should be printed.
...	additional arguments for RGCCA.

Details

MCOA resembles GCA and MFA in that it creates a set of reference scores, for which each block's individual scores should correlate maximally too, but also the variance within each block should be taken into account. A single component solution is equivalent to a PCA on concatenated blocks scaled by the so called inverse inertia.

Value

multiblock object including relevant scores and loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

- Le Roux; B. and H. Rouanet (2004). Geometric Data Analysis, From Correspondence Analysis to Structured Data Analysis. Dordrecht. Kluwer: p.180.
- Greenacre, Michael and Blasius, Jörg (editors) (2006). Multiple Correspondence Analysis and Related Methods. London: Chapman & Hall/CRC.

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`. Common functions for computation and extraction of results and plotting are found in `multiblock_results` and `multiblock_plots`, respectively.

Examples

```
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.mcoa   <- mcoa(potList)
plot(scores(pot.mcoa), labels="names")
```

`mcolors` *Colour palette generation from matrix of RGB values*

Description

Colour palette generation from matrix of RGB values

Usage

```
mcolors(
  n,
  colmatrix = matrix(c(0, 0, 1, 1, 1, 1, 1, 0, 0), 3, 3, byrow = TRUE)
)
```

Arguments

`n`	Integer number of colorus to produce.
`colmatrix`	A numeric `matrix` of three columns (R,G,B) to generate colour palette from.

Value

A vector of n colours in hexadecimal RGB format.

Examples

```
mcolors(5)
```

mfa *Multiple Factor Analysis - MFA*

Description

This is a wrapper for the `FactoMineR::MFA` function for computing MFA.

Usage

```
mfa(X, type = rep("c", length(X)), graph = FALSE, ...)
```

Arguments

`X`	`list` of input blocks.
`type`	`character` vector indicating block types, defaults to `rep("c", length(X))` for continuous values.
`graph`	`logical` indicating if decomposition should be plotted.
`...`	additional arguments for RGCCA approach.

Details

MFA is a methods typically used to compare several equally sized matrices. It is often used in sensory analyses, where matrices consist of sensory characteristics and products, and each assessor generates one matrix each. In its basic form, MFA scales all matrices by their largest eigenvalue, concatenates them and performs PCA on the result. There are several possibilities for plots and inspections of the model, handling of categorical and continuous inputs etc. connected to MFA.

Value

`multiblock` object including relevant scores and loadings. Relevant plotting functions: `multiblock_plots` and result functions: `multiblock_results`.

References

Pagès, J. (2005). Collection and analysis of perceived product inter-distances using multiple factor analysis: Application to the study of 10 white wines from the Loire valley. Food Quality and Preference, 16(7), 642–649.

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`. Common functions for computation and extraction of results and plotting are found in `multiblock_results` and `multiblock_plots`, respectively.

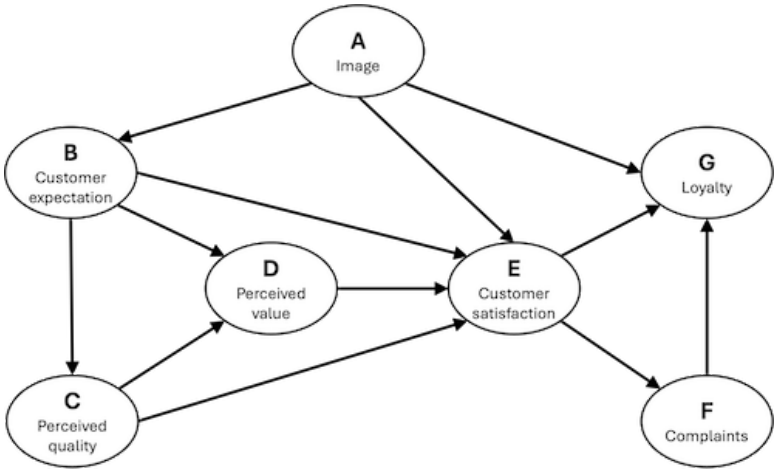
Examples

```
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.mfa    <- mfa(potList)
if(interactive()){
  plot(pot.mfa$MFA)
}
```

`mobile` *ECSI Mobile Mobile Phone Provider Dataset*

Description

Mobile data questionnaire often used as an example in path modelling. All the items are scaled from 1 to 10. Score 1 expresses a very negative point of view on the product while score 10 a very positive opinion. For details, see the original publication.



Usage

```
data(mobile)
```

Format

A data.frame having 250 rows and 7 variables:

A Image
B Customer expectation
C Perceived quality
D Perceived value
E Customer satisfaction
F Customer complaints
G Customer loyalty

References

Tenenhaus M, Esposito Vinzi V, Chatelin YM, Lauro C. PLS path modeling. Comput Stat Data Anal. 2005;48(1):159-205.

`multiblock_plots` *Plot Functions for Multiblock Objects*

Description

Plotting procedures for `multiblock` objects.

Usage

```
## S3 method for class 'multiblock'
scoreplot(
  object,
  comps = 1:2,
  block = 0,
  labels,
  identify = FALSE,
  type = "p",
  xlab,
  ylab,
  main,
  ...
)

## S3 method for class 'multiblock'
loadingplot(
  object,
  comps = 1:2,
  block = 0,
  scatter = TRUE,
  labels,
  identify = FALSE,
  type,
  lty,
  lwd = NULL,
  pch,
  cex = NULL,
  col,
  legendpos,
  xlab,
  ylab,
  main,
  pretty.xlabels = TRUE,
```

```
  xlim,
  ...
)

loadingweightplot(object, main = "Loading weights", ...)

## S3 method for class 'multiblock'
biplot(
  x,
  block = 0,
  comps = 1:2,
  which = c("x", "y", "scores", "loadings"),
  var.axes = FALSE,
  xlabs,
  ylabs,
  main,
  ...
)

corrplot(object, ...)

## Default S3 method:
corrplot(object, ...)

## S3 method for class 'mvr'
corrplot(object, ...)

## S3 method for class 'multiblock'
corrplot(
  object,
  comps = 1:2,
  labels = TRUE,
  col = 1:5,
  plotx = TRUE,
  ploty = TRUE,
  blockScores = FALSE,
  ...
)
```

Arguments

object	multiblock object.
comps	integer vector giving components, within block, to plot.
block	integer/character for block selection.
labels	character indicating if "names" or "numbers" should be plot symbols (optional).
identify	logical for activating identify to interactively identify points.

type	character for selecting type of plot to make. Defaults to "p" (points) for scatter plots and "l" (lines) for line plots.
xlab	character text for x labels.
ylab	character text for y labels.
main	character text for main header.
...	Not implemented.
scatter	logical indicating if a scatterplot of loadings should be made (default = TRUE).
lty	Vector of line type specifications (see par for details).
lwd	numeric vector of line width specifications.
pch	Vector of point specifications (see points for details).
cex	numeric vector of plot size expansions (see par for details).
col	integer vector of symbol/line colours (see par for details).
legendpos	character indicating legend position (if scatter is FALSE), e.g. legendpos = "topright".
pretty.xlabels	logical indicating if xlabels should be more nicely plotted (default = TRUE).
xlim	numeric vector of length two, with the x limits of the plot (optional).
x	multiblock object.
which	character for selecting type of biplot ("x" = default, "y", "scores", "loadings").
var.axes	logical indicating if second axes of a biplot should have arrows.
xlabs	character vector for labelling first set of biplot points (optional).
ylabs	character vector for labelling second set of biplot points (optional).
plotx	locical or integer/character. Whether to plot the X correlation loadings, optionally which block(s). Defaults to TRUE.
ploty	logical. Whether to plot the Y correlation loadings. Defaults to TRUE.
blockScores	logical. Correlation loadings from blockScores (default = FALSE).

Details

Plot functions for scores, loadings and loading.weights based on the functions found in the pls package.

Value

These plotting routines only generate plots and return no values.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results are found in multiblock_results.

Examples

```
data(wine)
sc <- sca(wine[c('Smell at rest', 'View', 'Smell after shaking')], ncomp = 4)
loadingplot(sc, block = 1, labels = "names", scatter = TRUE)
scoreplot(sc, labels = "names")
corrplot(sc)

data(potato)
so <- sopls(Sensory ~ NIRraw + Chemical + Compression, data=potato, ncomp = c(2,2,2),
            max_comps = 6, validation = "CV", segments = 10)
scoreplot(so, ncomp = c(2,1), block = 3, labels = "names")
corrplot(pcp(so, ncomp = c(2,2,2)))
```

---

multiblock_results *Result Functions for Multiblock Objects*

Description

Standard result computation and extraction functions for `multiblock` objects.

Usage

```
## S3 method for class 'multiblock'
scores(object, block = 0, ...)

## S3 method for class 'multiblock'
loadings(object, block = 0, ...)

## S3 method for class 'multiblock'
print(x, ...)

## S3 method for class 'multiblock'
summary(object, ...)
```

Arguments

`object`	`multiblock` object.
`block`	`integer/character` for block selection.
`...`	Not implemented.
`x`	`multiblock` object.

Details

Usage of the functions are shown using generics in the examples below. Object printing and summary are available through: `print.multiblock` and `summary.multiblock`. Scores and loadings have their own extensions of `scores()` and `loadings()` throught `scores.multiblock` and `loadings.multiblock`.

Value

Scores or loadings are returned by `scores.multiblock` and `loadings.multiblock`, while print and summary methods invisibly returns the object.

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`. Common functions for plotting are found in `multiblock_plots`, respectively.

Examples

```
data(wine)
sc <- sca(wine[c('Smell at rest', 'View', 'Smell after shaking')], ncomp = 4)
print(sc)
summary(sc)
head(loadings(sc, block = 1))
head(scores(sc))
```

---

mvrVal *MSEP, RMSEP and R2 of the MB-PLS model*

Description

Functions to estimate the mean squared error of prediction (MSEP), root mean squared error of prediction (RMSEP) and R^2 (A.K.A. coefficient of multiple determination) for a fitted MB-PLS models. Test-set, cross-validation and calibration-set estimates are implemented.

Usage

```
## S3 method for class 'mbpls'
R2(
  object,
  estimate,
  newdata,
  ncomp = 1:object$ncomp,
  comps,
  intercept = TRUE,
  se = FALSE,
  ...
)

## S3 method for class 'mbpls'
MSEP(
  object,
  estimate,
```

```
  newdata,
  ncomp = 1:object$ncomp,
  comps,
  intercept = TRUE,
  se = FALSE,
  ...
)

## S3 method for class 'mbpls'
RMSEP(object, ...)
```

Arguments

object	an mvr object
estimate	a character vector. Which estimators to use. Should be a subset of c("all", "train", "CV", "adjCV", "test"). "adjCV" is only available for (R)MSEP. See below for how the estimators are chosen.
newdata	a data frame with test set data.
ncomp, comps	a vector of positive integers. The components or number of components to use. See below.
intercept	logical. Whether estimates for a model with zero components should be returned as well.
se	logical. Whether estimated standard errors of the estimates should be calculated. Not implemented yet.
...	further arguments sent to underlying functions or (for RMSEP) to MSEP

Details

RMSEP simply calls MSEP and takes the square root of the estimates. It therefore accepts the same arguments as MSEP.

Several estimators can be used. "train" is the training or calibration data estimate, also called (R)MSEC. For R2, this is the unadjusted R^2. It is overoptimistic and should not be used for assessing models. "CV" is the cross-validation estimate, and "adjCV" (for RMSEP and MSEP) is the bias-corrected cross-validation estimate. They can only be calculated if the model has been cross-validated. Finally, "test" is the test set estimate, using newdata as test set.

Which estimators to use is decided as follows (see below for pls:mvrValstats). If estimate is not specified, the test set estimate is returned if newdata is specified, otherwise the CV and adjusted CV (for RMSEP and MSEP) estimates if the model has been cross-validated, otherwise the training data estimate. If estimate is "all", all possible estimates are calculated. Otherwise, the specified estimates are calculated.

Several model sizes can also be specified. If comps is missing (or is NULL), length(ncomp) models are used, with ncomp[1] components, ..., ncomp[length(ncomp)] components. Otherwise, a single model with the components comps[1], ..., comps[length(comps)] is used. If intercept is TRUE, a model with zero components is also used (in addition to the above).

The R^2 values returned by "R2" are calculated as $1 - SSE/SST$, where SST is the (corrected) total sum of squares of the response, and SSE is the sum of squared errors for either the fitted

values (i.e., the residual sum of squares), test set predictions or cross-validated predictions (i.e., the $PRESS$). For `estimate = "train"`, this is equivalent to the squared correlation between the fitted values and the response. For `estimate = "train"`, the estimate is often called the prediction R^2.

`mvrValstats` is a utility function that calculates the statistics needed by `MSEP` and `R2`. It is not intended to be used interactively. It accepts the same arguments as `MSEP` and `R2`. However, the `estimate` argument must be specified explicitly: no partial matching and no automatic choice is made. The function simply calculates the types of estimates it knows, and leaves the other untouched.

Value

`mvrValstats` returns a list with components

SSE three-dimensional array of SSE values. The first dimension is the different estimators, the second is the response variables and the third is the models.

SST matrix of SST values. The first dimension is the different estimators and the second is the response variables.

nobj a numeric vector giving the number of objects used for each estimator.

comps the components specified, with `0` prepended if `intercept` is `TRUE`.

cumulative `TRUE` if `comps` was `NULL` or not specified.

The other functions return an object of class `"mvrVal"`, with components

val three-dimensional array of estimates. The first dimension is the different estimators, the second is the response variables and the third is the models.

type `"MSEP"`, `"RMSEP"` or `"R2"`.

comps the components specified, with `0` prepended if `intercept` is `TRUE`.

cumulative `TRUE` if `comps` was `NULL` or not specified.

call the function call

Author(s)

Kristian Hovde Liland

References

Mevik, B.-H., Cederkvist, H. R. (2004) Mean Squared Error of Prediction (MSEP) Estimates for Principal Component Regression (PCR) and Partial Least Squares Regression (PLSR). *Journal of Chemometrics*, **18**(9), 422–429.

See Also

`mbpls`

Examples

```
data(oliveoil, package = "pls")
mod <- pls::plsr(sensory ~ chemical, ncomp = 4, data = oliveoil, validation = "LOO")
RMSEP(mod)
## Not run: plot(R2(mod))
```

`path_effects` *Analyse Multivariate Path Effects*

Description

Decomposes effects between blocks in a multiblock system into total effects, unique effects, common contributions, and additional effects. Supports both SO-PLS (Sequential Orthogonalised PLS) and ordinary least squares (OLS) approaches with cross-validation or fitted values.

Usage

```
path_effects(
  relations,
  blocks,
  validation,
  segments,
  SO = TRUE,
  fits = FALSE,
  boot = 0,
  ncomp = NULL,
  transitive_closure = FALSE,
  ...
)
```

Arguments

`relations`	A matrix defining the path structure. Rows specify relationships between blocks, with 2 columns: `c(from_block_index, to_block_index)`.
`blocks`	A multiblock data frame or list of named matrices containing the block data.
`validation`	Validation method passed to `pls::plsr()`. Common options: `"CV"` for cross-validation, "LOO" for leave-one-out validation, or `"none"` for fitted values.
`segments`	Number of segments for cross-validation. Default is 5.
`SO`	Logical. If `TRUE` (default), use SO-PLS; if `FALSE`, use OLS.
`fits`	Logical. If `TRUE`, use fitted values instead of cross-validation. Default is `FALSE`.
`boot`	Number of bootstrap samples for computing standard errors. Default is 0 (no bootstrapping).
`ncomp`	Optional component settings per predictor block. Supply either:

- a vector with length equal to the number of predictor blocks (`unique(relations[,1])`), in ascending predictor-block order, or
- a full-length vector with one entry per block (non-predictors may be `0`). Positive integers set upper limits; negative integers force exact component counts. All predictor entries must be non-zero and have the same sign.

`transitive_closure`
: Logical. If `TRUE`, automatically add transitive closure to the relations. Default is `FALSE`.

`...`
: Additional arguments passed to underlying fitting functions.

Value

An object of class `path_effects`, which is a matrix with the following components for each path:

- `Tot`: Total effect
- `Un`: Unique effect
- `Co`: Common contribution
- `Ad`: Additional effect

Attributes include:

- `scaled`: Scaled contribution matrix
- `individual`: Individual-level contributions
- `boot`: Bootstrap replicates (if `boot > 0`)

See Also

`print.path_effects()`, `plot.path_effects()`

Examples

```
# Analysis of the mobile dataset
data(mobile)

# Define path structure (A->B, A->E, A->G, B->C, B->D, B->E, C->D, D->E,
#                        D->E, E->F, E->G, F->G)
paths <- matrix(c(1,2, 1,5, 1,7, 2,3, 2,4, 2,5, 3,4, 3,5, # Add 0,2, for A->C
                  4,5, 5,6, 5,7, 6,7),
                ncol=2, byrow=TRUE)

# Compute path effects with cross-validation using SO-PLS
pem <- path_effects(paths, mobile, validation="CV", segments=5,
                    segment.type="consecutive")

# Print results
print(pem)

# Plot all results
plot(pem)
```

```
# Print and plot single path
print(pem, "A","G")
plot(pem, from = "A", to = "G")

# Print and plot results per variable
print(pem, individual = TRUE)
plot(pem, individual = TRUE)

# Analysis of the NIR-Raman-PUFA data (emulsions)
data(emulsions)

# Standardise response
emulsions$PUFA <- scale(emulsions$PUFA)

# Define path structure (NIR->Raman, NIR->PUFA, Raman->PUFA)
paths_NRP <- matrix(c(1,2,1,3,2,3), ncol = 2, byrow = TRUE)

## Not run:  # Too time consuming
# Compute path effects with cross-validation using SO-PLS
pem_NRP <- path_effects(paths_NRP, emulsions, validation="CV",
                    segments = 5, segment.type="consecutive",
                    ncomp=c(16,15))

# Print results
print(pem_NRP)

# Plot all results
plot(pem_NRP)

## End(Not run)
# Reversed order of NIR and Raman (uncomment to run)
# paths_RNP <- matrix(c(2,1,2,3,1,3), ncol = 2, byrow = TRUE)
# pem_RNP <- path_effects(paths_RNP, emulsions, validation="CV",
#                    segments = 5, segment.type="consecutive", ncomp=c(16,15))
# print(pem_RNP)
```

---

pca *Principal Component Analysis - PCA*

Description

This is a wrapper for the `pls::PCR` function for computing PCA.

Usage

```
pca(X, scale = FALSE, ncomp = 1, ...)
```

Arguments

X	matrix of input data.
scale	logical indicating if variables should be standardised (default=FALSE).
ncomp	integer number of principal components to return.
...	additional arguments to pls:pcr.

Details

PCA is a method for decomposing a matrix into subspace components with sample scores and variable loadings. It can be formulated in various ways, but the standard formulation uses singular value decomposition to create scores and loadings. PCA is guaranteed to be the optimal way of extracting orthogonal subspaces from a matrix with regard to the amount of explained variance per component.

Value

multiblock object with scores, loadings, mean X values and explained variances. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

Pearson, K. (1901) On lines and planes of closest fit to points in space. Philosophical Magazine, 2, 559–572.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
X <- potato$Chemical

pca.pot  <- pca(X, ncomp = 2)
```

---

pcagca *PCA-GCA*

Description

PCA-GCA is a methods which aims at estimating subspaces of common, local and distinct variation from two or more blocks.

Usage

```
pcagca(
  X,
  commons = 2,
  auto = TRUE,
  auto.par = list(explVarLim = 40, rLim = 0.8),
  manual.par = list(ncomp = 0, ncommon = 0),
  tol = 10^-12
)
```

Arguments

X	list of input blocks
commons	numeric giving the highest number of blocks to combine when calculating local or common scores.
auto	logical indicating if automatic choice of complexities should be used.
auto.par	named list setting limits for automatic choice of complexities.
manual.par	named list for manual choice of blocks. The list consists of ncomp which indicates the number of components to extract from each block and ncommon which is the corresponding for choosing the block combinations (local/common). For the latter, use unique_combos(n_blocks, commons) to see order of local/common blocks. Component numbers will be reduced if simpler models give better predictions. See example.
tol	numeric tolerance for component inclusion (singular values).

Details

The name PCA-GCA comes from the process of first applying PCA to each block, then using GCA to estimate local and common components, and finally orthogonalising the block-wise scores on the local/common ones and re-estimating these to obtain distinct components. The procedure is highly similar to the supervised method PO-PLS, where the PCA steps are exchanged with PLS.

Value

multiblock object including relevant scores and loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results. Distinct components are marked as 'D(x), Comp c' for block x and component c while local and common components are marked as "C(x1, x2), Comp c", where x1 and x2 (and more) are block numbers.

References

Smilde, A., Måge, I., Naes, T., Hankemeier, T.,Lips, M., Kiers, H., Acar, E., and Bro, R.(2017). Common and distinct components in data fusion. Journal of Chemometrics, 31(7), e2900.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.pcagca <- pcagca(potList)

# Show origin and type of all components
lapply(pot.pcagca$blockScores,colnames)

# Basic multiblock plot
plot(scores(pot.pcagca, block=2), comps=1, labels="names")
```

---

plot.path_effects *Plot Path Effects*

Description

Visualisation of multiblock path effects decomposition.

Usage

```
## S3 method for class 'path_effects'
plot(
  x,
  from = NULL,
  to = NULL,
  scaled = FALSE,
  individual = FALSE,
  spectra = numeric(0),
  mar = c(0.5, 2.8, 1.8, 0.3),
  ...
)
```

Arguments

x	An object of class path_effects.
from	Optional source block name or index. If specified with to, plots only that block pair.
to	Optional target block name or index. If specified with from, plots only that block pair.

scaled	Logical. If TRUE, display scaled contributions. Default is FALSE.
individual	Logical. If TRUE, display individual-level effects. Default is FALSE.
spectra	Optional numeric vector for spectral data visualization. Default is empty.
mar	Margins for the plot. Default is c(0.5,2.8,1.8,0.3).
...	Additional arguments passed to plotting functions.

Value

Invisibly returns NULL.

See Also

path_effects(), print.path_effects()

---

popls *Parallel and Orthogonalised Partial Least Squares - PO-PLS*

Description

This is a basic implementation of PO-PLS with manual and automatic component selections.

Usage

```
popls(
  X,
  Y,
  commons = 2,
  auto = TRUE,
  auto.par = list(explVarLim = 40, rLim = 0.8),
  manual.par = list(ncomp = rep(0, length(X)), ncommon = list())
)
```

Arguments

X	list of input blocks
Y	matrix of response variable(s)
commons	numeric giving the highest number of blocks to combine when calculating local or common scores.
auto	logical indicating if automatic choice of complexities should be used.
auto.par	named list setting limits for automatic choice of complexities. See Details.
manual.par	named list for manual choice of blocks. The list consists of ncomp which indicates the number of components to extract from each block and ncommon which is the corresponding for choosing the block combinations (local/common). For the latter, use unique_combos(n_blocks, commons) to see order of local/common blocks. Component numbers will be reduced if simpler models give better predictions. See example.

Details

PO-PLS decomposes a set of input data blocks into common, local and distinct components through a process involving `pls` and `gca`. The `rLim` parameter is a lower bound for the GCA correlation when building common components, while explVarLim is the minimum explained variance for common components and unique components.

Value

A `multiblock` object with block-wise, local and common loadings and scores. Relevant plotting functions: `multiblock_plots` and result functions: `multiblock_results`.

References

- I Måge, BH Mevik, T Næs. (2008). Regression models with process variables and parallel blocks of raw material measurements. Journal of Chemometrics: A Journal of the Chemometrics Society 22 (8), 443-456
- I Måge, E Menichelli, T Næs (2012). Preference mapping by PO-PLS: Separating common and unique information in several data blocks. Food quality and preference 24 (1), 8-16

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`. Common functions for computation and extraction of results and plotting are found in `multiblock_results` and `multiblock_plots`, respectively.

Examples

```
data(potato)

# Automatic analysis
pot.po.auto <- popls(potato[1:3], potato[['Sensory']][,1])
pot.po.auto$explVar

# Manual choice of up to 5 components for each block and 1, 0, and 2 blocks,
# respectively from the (1,2), (1,3) and (2,3) combinations of blocks.
pot.po.man <- popls(potato[1:3], potato[['Sensory']][,1], auto=FALSE,
                manual.par = list(ncomp=c(5,5,5), ncommon=c(1,0,2)))
pot.po.man$explVar

# Score plot for local (2,3) components
plot(scores(pot.po.man,3), comps=1:2, labels="names")
```

potato *Sensory, rheological, chemical and spectroscopic analysis of potatoes.*

Description

A dataset containing 9 blocks of measurements on 26 potatoes. Original dataset can be found at http://models.life.ku.dk/Texture_Potatoes. This version has been pre-processed as follows (corresponding to Liland et al. 2016):

- Variables containing NaN have been removed.
- Chemical and Compression blocks have been scaled by standard deviations.
- NIR blocks have been subjected to SNV (Standard Normal Variate).

Usage

```
data(potato)
```

Format

A data.frame having 26 rows and 9 variables:

Chemical Matrix of chemical measurements

Compression Matrix of rheological compression data

NIRraw Matrix of near-infrared measurements of raw potatoes

NIRcooked Matrix of near-infrared measurements of cooked potatoes

CPMGraw Matrix of NMR (CPMG) measurements of raw potatoes

CPMGcooked Matrix of NMR (CPMG) measurements of cooked potatoes

FIDraw Matrix of NMR (FID) measurements of raw potatoes

FIDcooked Matrix of NMR (FID) measurements of cooked potatoes

Sensory Matrix of sensory assessments

References

- L.G.Thygesen, A.K.Thybo, S.B.Engelsen, Prediction of Sensory Texture Quality of Boiled Potatoes From Low-field1H NMR of Raw Potatoes. The Role of Chemical Constituents. LWT - Food Science and Technology 34(7), 2001, pp 469-477.
- Kristian Hovde Liland, Tormod Næs, Ulf Geir Indahl, ROSA – a fast extension of Partial Least Squares Regression for Multiblock Data Analysis, Journal of Chemometrics 30:11 (2016), pp. 651-662.

`predict.mbpls` *Predict Method for MBPLS*

Description

Prediction for the mbpls (MBPLS) model. New responses or scores are predicted using a fitted model and a data.frame or list containing matrices of observations.

Usage

```
## S3 method for class 'mbpls'
predict(
  object,
  newdata,
  ncomp = 1:object$ncomp,
  comps,
  type = c("response", "scores"),
  na.action = na.pass,
  ...
)
```

Arguments

`object`	an `mvr` object. The fitted model
`newdata`	a data frame. The new data. If missing, the training data is used.
`ncomp`, `comps`	vector of positive integers. The components to use in the prediction. See below.
`type`	character. Whether to predict scores or response values
`na.action`	function determining what should be done with missing values in `newdata`. The default is to predict `NA`. See `na.omit` for alternatives.
`...`	further arguments. Currently not used

Details

When `type` is `"response"` (default), predicted response values are returned. If `comps` is missing (or is `NULL`), predictions for `length(ncomp)` models with `ncomp[1]` components, `ncomp[2]` components, etc., are returned. Otherwise, predictions for a single model with the exact components in `comps` are returned. (Note that in both cases, the intercept is always included in the predictions. It can be removed by subtracting the `Ymeans` component of the fitted model.)

When `type` is `"scores"`, predicted score values are returned for the components given in `comps`. If `comps` is missing or `NULL`, `ncomps` is used instead.

Value

When `type` is `"response"`, a three dimensional array of predicted response values is returned. The dimensions correspond to the observations, the response variables and the model sizes, respectively.

When `type` is `"scores"`, a score matrix is returned.

Note

A warning message like ‘'newdata' had 10 rows but variable(s) found have 106 rows’ means that not all variables were found in the newdata data frame. This (usually) happens if the formula contains terms like yarn$NIR. Do not use such terms; use the data argument instead. See mvr for details.

Author(s)

Kristian Hovde Liland

See Also

mbpls

Examples

```
data(potato)
mb <- mbpls(Sensory ~ Chemical+Compression, data=potato, ncomp = 5, subset = 1:26 <= 18)
testdata <- subset(potato, 1:26 > 18)

# Predict response
yhat <- predict(mb, newdata = testdata)

# Predict scores and plot
scores <- predict(mb, newdata = testdata, type = "scores")
scoreplot(mb)
points(scores[,1], scores[,2], col="red")
legend("topright", legend = c("training", "test"), col=1:2, pch = 1)
```

---

preprocess *Preprocessing of block data*

Description

This is an interface to simplify preprocessing of one, a subset or all blocks in a multiblock object, e.g., a data.frame (see block.data.frame) or list. Several standard preprocessing methods are supplied in addition to letting the user supply it’s own function.

Usage

```
block.preprocess(
  X,
  block = 1:length(X),
 fun = c("autoscale", "center", "scale", "SNV", "EMSC", "Fro", "FroSq", "SingVal"),
  ...
)
```

Arguments

X	data.frame or list of data.
block	vector of block(s) to preprocess (integers or characters).
fun	character or function selecting which preprocessing to apply (see Details).
...	additional arguments to underlying functions.

Details

The fun parameter controls the type of preprocessing to be performed:

- autoscale: centre and scale each feature/variable.
- center: centre each feature/variable.
- scale: scale each feature/variable.
- SNV: Standard Normal Variate correction, i.e., centre and scale each sample across features/variables.
- EMSC: Extended Multiplicative Signal Correction defaulting to basic EMSC (2nd order polynomials). Further parameters are sent to EMSC::EMSC.
- Fro: Frobenius norm scaling of whole block.
- FroSq: Squared Frobenius norm scaling of whole block (sum of squared values).
- SingVal: Singular value scaling of whole block (first singular value).
- User defined: If a function is supplied, this will be applied to chosen blocks. Preprocessing can be done for all blocks or a subset. It can also be done in a series of operations to combine preprocessing techniques.

Value

The input multiblock object is preprocessed and returned.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
# Autoscale Chemical block
potato <- block.preprocess(potato, block = "Chemical", "autoscale")
# Apply SNV to NIR blocks
potato <- block.preprocess(potato, block = 3:4, "SNV")
# Centre Sensory block
potato <- block.preprocess(potato, block = "Sensory", "center")
# Scale all blocks to unit Frobenius norm
potato <- block.preprocess(potato, fun = "Fro")
```

```
# Effect of SNV
NIR <- (potato$NIRraw + rnorm(26)) * rnorm(26,1,0.2)
NIRc <- block.preprocess(list(NIR), fun = "SNV")[[1]]
old.par <- par(mfrow = c(2,1), mar = c(4,4,1,1))
matplot(t(NIR), type="l", main = "uncorrected", ylab = "")
matplot(t(NIRc), type="l", main = "corrected", ylab = "")
par(old.par)
```

`print.path_effects` *Print Path Effects*

Description

Prints a summary of path effects analysis with optional bootstrap standard errors.

Usage

```
## S3 method for class 'path_effects'
print(
  x,
  from = NULL,
  to = NULL,
  nsmall = 2,
  digits = 1,
  scaled = FALSE,
  individual = FALSE,
  boot = TRUE,
  ...
)
```

Arguments

`x`	An object of class `path_effects`.
`from`	Optional source block name or index. If specified with `to`, prints only that block pair.
`to`	Optional target block name or index. If specified with `from`, prints only that block pair.
`nsmall`	Number of decimal places to display. Default is 2.
`digits`	Number of significant digits. Default is 1.
`scaled`	Logical. If `TRUE`, display scaled contributions. Default is `FALSE`.
`individual`	Logical. If `TRUE`, display individual-level effects. Default is `FALSE`.
`boot`	Logical. If `TRUE` and bootstrap samples are available, display standard errors. Default is `TRUE`.
`...`	Additional arguments (currently unused).

Value

Invisibly returns `x`.

See Also

`path_effects()`, `plot.path_effects()`

`rosa` *Response Oriented Sequential Alternation - ROSA*

Description

Formula based interface to the ROSA algorithm following the style of the `pls` package.

Usage

```
rosa(
  formula,
  ncomp,
  Y.add,
  common.comp = 1,
  data,
  subset,
  na.action,
  scale = FALSE,
  weights = NULL,
  validation = c("none", "CV", "LOO"),
  internal.validation = FALSE,
  fixed.block = NULL,
  design.block = NULL,
  canonical = TRUE,
  ...
)
```

Arguments

`formula`	Model formula accepting a single response (block) and predictor block names separated by + signs.
`ncomp`	The maximum number of ROSA components.
`Y.add`	Optional response(s) available in the data set.
`common.comp`	Automatically create all combinations of common components up to length `common.comp` (default = 1).
`data`	The data set to analyse.
`subset`	Expression for subsetting the data before modelling.
`na.action`	How to handle NAs (no action implemented).

scale	Optionally scale predictor variables by their individual standard deviations.
weights	Optional object weights.
validation	Optional cross-validation strategy "CV" or "LOO".
internal.validation	Optional cross-validation for block selection process, "LOO", "CV3", "CV5", "CV10" (CV-number of segments), or vector of integers (default = FALSE).
fixed.block	integer vector with block numbers for each component (0 = not fixed) or list of length <= ncomp (element length 0 = not fixed).
design.block	integer vector containing block numbers of design blocks
canonical	logical indicating if canonical correlation should be use when calculating loading weights (default), enabling B/W maximization, common components, etc. Alternatively (FALSE) a PLS2 strategy, e.g. for spectra response, is used.
...	Additional arguments for cvseg or rosa.fit

Details

ROSA is an opportunistic method sequentially selecting components from whichever block explains the response most effectively. It can be formulated as a PLS model on concatenated input block with block selection per component. This implementation adds several options that are not described in the literature. Most importantly, it opens for internal validation in the block selection process, making this more robust. In addition it handles design blocks explicitly, enables classification and secondary responses (CPLS), and definition of common components.

Value

An object of classes rosa and mvr having several associated printing (rosa_results) and plotting methods (rosa_plots).

References

Liland, K.H., Næs, T., and Indahl, U.G. (2016). ROSA - a fast extension of partial least squares regression for multiblock data analysis. Journal of Chemometrics, 30, 651–662, doi:10.1002/cem.2824.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in rosa_results and rosa_plots, respectively.

Examples

```
data(potato)
mod <- rosa(Sensory[,1] ~ ., data = potato, ncomp = 10, validation = "CV", segments = 5)
summary(mod)

# For examples of ROSA results and plotting see
# ?rosa_results and ?rosa_plots.
```

rosa_plots *Plotting functions for ROSA models*

Description

Various plotting procedures for rosa objects.

Usage

```
## S3 method for class 'rosa'
image(
  x,
  type = c("correlation", "residual", "order"),
  ncomp = x$ncomp,
  col = mcolors(128),
  legend = TRUE,
  mar = c(5, 6, 4, 7),
  las = 1,
  ...
)

## S3 method for class 'rosa'
barplot(
  height,
  type = c("train", "CV"),
  ncomp = height$ncomp,
  col = mcolors(ncomp),
  ...
)
```

Arguments

x	A rosa object
type	An optional character for selecting the plot type. For image.rosa the options are: "correlation" (default), "residual" or "order". For barplot.rosa the options indicate: explained variance should be based on training data ("train") or cross-validation ("CV").
ncomp	Integer to control the number of components to plot (if fewer than the fitted number of components).
col	Colours used for the image and bar plot, defaulting to mcolors(128).
legend	Logical indicating if a legend should be included (default = TRUE) for image.rosa.
mar	Figure margins, default = c(5,6,4,7) for image.rosa.
las	Axis text direction, default = 1 for image.rosa.
...	Additional parameters passed to loadingplot, image, axis, color.legend, or barplot.
height	A rosa object.

Details

Usage of the functions are shown using generics in the examples below. `image.rosa` makes an image plot of each candidate score's correlation to the winner or the block-wise response residual. These plots can be used to find alternative block selection for tweaking the ROSA model. `barplot.rosa` makes barplot of block and component explained variances. `loadingweightsplot` is an adaptation of `pls::loadingplot` to plot loading weights.

Value

No return.

References

Liland, K.H., Næs, T., and Indahl, U.G. (2016). ROSA - a fast extension of partial least squares regression for multiblock data analysis. Journal of Chemometrics, 30, 651–662, doi:10.1002/cem.2824.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results in rosa_results.

Examples

```
data(potato)
mod <- rosa(Sensory[,1] ~ ., data = potato, ncomp = 5)
image(mod)
barplot(mod)
loadingweightplot(mod)
```

`rosa_results` *Result functions for ROSA models*

Description

Standard result computation and extraction functions for ROSA (rosa).

Usage

```
## S3 method for class 'rosa'
predict(
  object,
  newdata,
  ncomp = 1:object$ncomp,
  comps,
  type = c("response", "scores"),
  na.action = na.pass,
```

```
  ...
)

## S3 method for class 'rosa'
coef(object, ncomp = object$ncomp, comps, intercept = FALSE, ...)

## S3 method for class 'rosa'
print(x, ...)

## S3 method for class 'rosa'
summary(
  object,
  what = c("all", "validation", "training"),
  digits = 4,
  print.gap = 2,
  ...
)

blockexpl(object, ncomp = object$ncomp, type = c("train", "CV"))

## S3 method for class 'rosaexpl'
print(x, digits = 3, compwise = FALSE, ...)

rosa.classify(object, classes, newdata, ncomp, LQ)

## S3 method for class 'rosa'
scores(object, ...)

## S3 method for class 'rosa'
loadings(object, ...)
```

Arguments

object	A rosa object.
newdata	Optional new data with the same types of predictor blocks as the ones used for fitting the object.
ncomp	An integer giving the number of components to apply (cummulative).
comps	An integer vector giving the exact components to apply (subset).
type	For blockexpl: Character indicating which type of explained variance to compute (default = "train", alternative = "CV").
na.action	Function determining what to do with missing values in newdata.
...	Additional arguments. Currently not implemented.
intercept	A logical indicating if coefficients for the intercept should be included (default = FALSE).
x	A rosa object.
what	A character indicating if summary should include all, validation or training.

digits	The number of digits used for printing.
print.gap	Gap between columns when printing.
compwise	Logical indicating if block-wise (default/FALSE) or component-wise (TRUE) explained variance should be printed.
classes	A character vector of class labels.
LQ	A character indicating if 'max' (maximum score value), 'lda' or 'qda' should be used when classifying.

Details

Usage of the functions are shown using generics in the examples below. Prediction, regression coefficients, object printing and summary are available through: predict.rosa, coef.rosa, print.rosa and summary.rosa. Explained variances are available (block-wise and global) through blockexpl and print.rosaexpl. Scores and loadings have their own extensions of scores() and loadings() throught scores.rosa and loadings.rosa. Finally, there is work in progress on classifcation support through rosa.classify.

When type is "response" (default), predicted response values are returned. If comps is missing (or is NULL), predictions for length(ncomp) models with ncomp[1] components, ncomp[2] components, etc., are returned. Otherwise, predictions for a single model with the exact components in comps are returned. (Note that in both cases, the intercept is always included in the predictions. It can be removed by subtracting the Ymeans component of the fitted model.)

If comps is missing (or is NULL), coef()[,,ncomp[i]] are the coefficients for models with ncomp[i] components, for $i = 1, \ldots, length(ncomp)$. Also, if intercept = TRUE, the first dimension is $nxvar + 1$, with the intercept coefficients as the first row.

If comps is given, however, coef()[,,comps[i]] are the coefficients for a model with only the component comps[i], i.e., the contribution of the component comps[i] on the regression coefficients.

Value

Returns depend on method used, e.g. predict.rosa returns predicted responses or scores depending on inputs, coef.rosa returns regression coefficients, blockexpl returns an object of class rosaexpl containing block-wise and component-wise explained variance contained in a matrix with attributes.

References

Liland, K.H., Næs, T., and Indahl, U.G. (2016). ROSA - a fast extension of partial least squares regression for multiblock data analysis. Journal of Chemometrics, 30, 651–662, doi:10.1002/cem.2824.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in rosa_results and rosa_plots, respectively.

Examples

```
data(potato)
mod <- rosa(Sensory[,1] ~ ., data = potato, ncomp = 5, subset = 1:20)
testset <- potato[-(1:20),]; testset$Sensory <- testset$Sensory[,1,drop=FALSE]
predict(mod, testset, ncomp=5)
dim(coef(mod, ncomp=5)) # <variables x responses x components>
print(mod)
summary(mod)
blockexpl(mod)
print(blockexpl(mod), compwise=TRUE)
```

---

sca *Simultaneous Component Analysis - SCA*

Description

This is a basic implementation of the SCA-P algorithm (least restricted SCA) with support for both sample- and variable-linked modes.

Usage

```
sca(X, ncomp = 2, scale = FALSE, samplelinked = "auto", ...)
```

Arguments

`X`	`list` of input blocks.
`ncomp`	`integer` number of components to extract.
`scale`	`logical` indicating autoscaling of features (default = FALSE).
`samplelinked`	`character/logical` indicating if blocks are linked by samples (TRUE) or variables (FALSE). Using 'auto' (default), this will be determined automatically.
`...`	additional arguments (not used).

Details

SCA, in its original variable-linked version, calculates common loadings and block-wise scores. There are many possible constraints and specialisations. This implementations uses PCA as the backbone, thus resulting in deterministic, ordered components. A parameter controls the linking mode, but if left untouched an attempt is made at automatically determining variable or sample linking.

Value

`multiblock` object including relevant scores and loadings. Relevant plotting functions: `multiblock_plots` and result functions: `multiblock_results`.

References

Levin, J. (1966) Simultaneous factor analysis of several gramian matrices. Psychometrika, 31(3), 413–419.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
# Object linked data
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.sca    <- sca(potList)
plot(scores(pot.sca), labels="names")

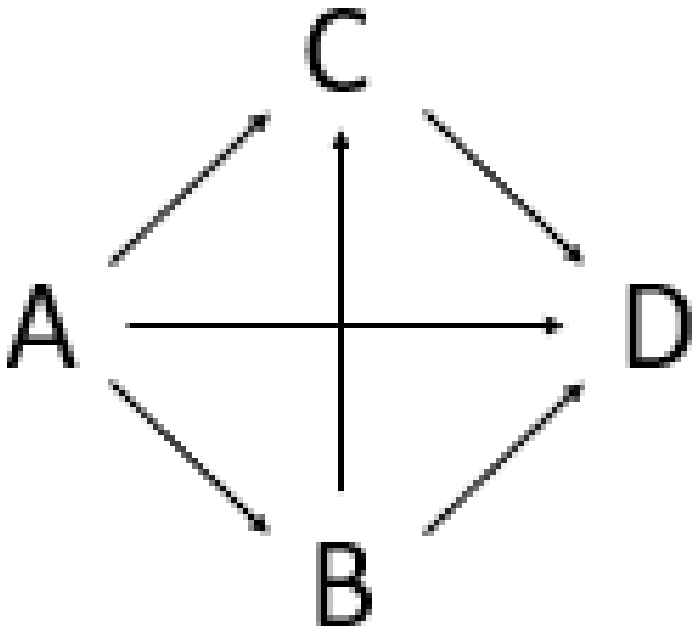
# Variable linked data
data(candies)
candyList <- lapply(1:nlevels(candies$candy),function(x)candies$assessment[candies$candy==x,])
pot.sca    <- sca(candyList, samplelinked = FALSE)
pot.sca
```

---

simulated *Data simulated to have certain characteristics.*

Description

A dataset containing simulated data for 4 connected events where A is the starting point and D is the end point. This can be described as a directed acyclic graph (sketched below, moving left->right).



Subpaths include: ABD, AD, ABCD, ACD

Usage

```
data(simulated)
```

Format

A list of matrices having 200 rows and 10 variables:

A Simulated matrix A

B Simulated matrix B ...

References

Tormod Næs, Rosaria Romano, Oliver Tomic, Ingrid Måge, Age Smilde, Kristian Hovde Liland, Sequential and orthogonalized PLS (SO-PLS) regression for path analysis: Order of blocks and relations between effects. Journal of Chemometrics, In Press

---

`smbpls` *Sparse Multiblock Partial Least Squares - sMB-PLS*

Description

sMB-PLS is an adaptation of MB-PLS (`mbpls`) that enforces sparseness in loading weights when computing PLS components in the global model.

Usage

```
smbpls(
  formula,
  data,
  subset,
  na.action,
  X = NULL,
  Y = NULL,
  ncomp = 1,
  scale = FALSE,
  shrink = NULL,
  truncation = NULL,
  trunc.width = 0.95,
  blockScale = c("sqrtnvar", "ssq", "none"),
  ...
)
```

Arguments

formula	Model formula accepting a single response (block) and predictor block names separated by + signs.
data	The data set to analyse.
subset	Expression for subsetting the data before modelling.
na.action	How to handle NAs (no action implemented).
X	list of input blocks. If X is supplied, the formula interface is skipped.
Y	matrix of responses.
ncomp	integer number of PLS components.
scale	logical for autoscaling inputs (default = FALSE).
shrink	numeric scalar indicating degree of L1-shrinkage/Soft-Thresholding (optional), 0 <= shrink < 1.
truncation	character indicating type of truncation (optional) "Lenth" uses asymmetric confidence intervals to determine outlying loading weights. "quantile" uses a quantile plot approach to determining outliers.
trunc.width	numeric indicating confidence of "Lenth type" confidence interval or quantile in "quantile plot" approach. Default = 0.95.
blockScale	Either a character indicating type of block scaling or a numeric vector of block weights (see Details).
...	additional arguments to pls::plsr.

Details

Two versions of sparseness are supplied: Soft-Threshold PLS, also known as Sparse PLS, and Truncation PLS. The former uses L1 shrinkage of loading weights, while the latter comes in two flavours, both estimating inliers and outliers. The "Lenth" method uses asymmetric confidence intervals around the median of a loading weigh vector to estimate inliers. The "quantile" method

uses a quantile plot approach to estimate outliers as deviations from the estimated quantile line. As with ordinary MB-PLS scaled input blocks (1/sqrt(ncol)) are used.

Block weighting is performed after scaling all variables and is by default `"sqrtnvar"`: 1/sqrt(ncol(X[[i]])) in each block. Alternatives are `"ssq"`: 1/norm(X[[i]], "F")^2 and `"none"`: 1/1. Finally, if a `numeric` vector is supplied, it will be used to scale the blocks after `"ssq"` scaling, i.e., Z[[i]] = X[[i]] / norm(X[[i]], "F")^2 * blockScale[i].

Value

`multiblock`, `mvr` object with super-scores, super-loadings, block-scores and block-loading, and the underlying `mvr` (PLS) object for the super model, with all its result and plot possibilities. Relevant plotting functions: `multiblock_plots` and result functions: `multiblock_results`.

References

- Sæbø, S.; Almøy, T.; Aarøe, J. & Aastveit, A. ST-PLS: a multi-directional nearest shrunken centroid type classifier via PLS Journal of Chemometrics: A Journal of the Chemometrics Society, Wiley Online Library, 2008, 22, 54-62.
- Lê Cao, K.; Rossouw, D.; Robert-Granié, C. & Besse, P. A sparse PLS for variable selection when integrating omics data Statistical applications in genetics and molecular biology, 2008, 7.
- Liland, K.; Høy, M.; Martens, H. & Sæbø, S. Distribution based truncation for variable selection in subspace methods for multivariate regression Chemometrics and Intelligent Laboratory Systems, 2013, 122, 103-111.
- Karaman, I.; Nørskov, N.; Yde, C.; Hedemann, M.; Knudsen, K. & Kohler, A. Sparse multiblock PLSR for biomarker discovery when integrating data from LC–MS and NMR metabolomics Metabolomics, 2015, 11, 367-379.

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`.

Examples

```
data(potato)

# Truncation MB-PLS
# Loading weights inside 60% confidence intervals around the median are set to 0.
tmb <- smbpls(Sensory ~ Chemical+Compression, data=potato, ncomp = 5,
              truncation = "Lenth", trunc.width = 0.6)

# Alternative XY-interface
tmb.XY <- smbpls(X=potato[c('Chemical','Compression')], Y=potato[['Sensory']], ncomp = 5,
              truncation = "Lenth", trunc.width = 0.6)
identical(tmb, tmb.XY)
scoreplot(tmb, labels="names") # Exploiting mvr object structure from pls package
loadingweightplot(tmb, labels="names")
```

```
# Soft-Threshold / Sparse MB-PLS
# Loading weights are subtracted by 60% of maximum value.
smb <- smbpls(X=potato[c('Chemical','Compression')], Y=potato[['Sensory']],
              ncomp = 5, shrink = 0.6)
print(smb)
scoreplot(smb, labels="names") # Exploiting mvr object structure from pls package
loadingweightplot(smb, labels="names")

# Emphasis may be different for blocks
smb <- smbpls(X=potato[c('Chemical','Compression')], Y=potato[['Sensory']],
              ncomp = 5, shrink = 0.6, blockScale = c(1, 10))
```

SO_TDI *Total, direct, indirect and additional effects in SO-PLS-PM.*

Description

SO-PLS-PM is the use of SO-PLS for path-modelling. This particular function is used to compute effects (explained variances) in sub-paths of the directed acyclic graph.

Usage

```
sopls_pm(
  X,
  Y,
  ncomp,
  max_comps = min(sum(ncomp), 20),
  sel.comp = "opt",
  computeAdditional = FALSE,
  sequential = FALSE,
  B = NULL,
  k = 10,
  type = "consecutive",
  simultaneous = TRUE
)

## S3 method for class 'SO_TDI'
print(x, showComp = TRUE, heading = "SO-PLS path effects", digits = 2, ...)

sopls_pm_multiple(
  X,
  ncomp,
  max_comps = min(sum(ncomp), 20),
  sel.comp = "opt",
  computeAdditional = FALSE,
  sequential = FALSE,
  B = NULL,
  k = 10,
```

```
  type = "consecutive"
)

## S3 method for class 'SO_TDI_multiple'
print(x, heading = "SO-PLS path effects", digits = 2, ...)
```

Arguments

X	A list of input blocks (of type matrix).
Y	A matrix of response(s).
ncomp	An integer vector giving the number of components per block or a single integer for common number of components.
max_comps	Maximum total number of components.
sel.comp	A character or integer vector indicating the type ("opt" - minimum error / "chi" - chi-squared reduced) or exact number of components in selections.
computeAdditional	A logical indicating if additional components should be computed.
sequential	A logical indicating if sequential component optimization should be applied.
B	An integer giving the number of bootstrap replicates for variation estimation.
k	An integer indicating number of cross-validation segments (default = 10).
type	A character for selecting type of cross-validation segments (default = "consecutive").
simultaneous	logical indicating if simultaneous orthogonalisation on intermediate blocks should be performed (default = TRUE).
x	An object of type SO_TDI.
showComp	A logical indicating if components should be shown in print (default = TRUE).
heading	A character giving the heading of the print.
digits	An integer for selecting number of digits in print.
...	Not implemented

Details

sopls_pm computes 'total', 'direct', 'indirect' and 'additional' effects for the 'first' versus the 'last' input block by cross-validated explained variances. 'total' is the explained variance when doing regression of 'first' -> 'last'. 'indirect' is the the same, but controlled for the intermediate blocks. 'direct' is the left-over part of the 'total' explained variance when subtracting the 'indirect'. Finally, 'additional' is the added explained variance of 'last' for each block following 'first'.

sopls_pm_multiple is a wrapper for sopls_pm that repeats the calculation for all pairs of blocks from 'first' to 'last'. Where sopls_pm has a separate response, Y, signifying the 'last' block, sopls_pm_multiple has multiple 'last' blocks, depending on sub-path, thus collects the response(s) from the list of blocks X.

When sel.comp = "opt", the number of components for all models are optimized using cross-validation within the ncomp and max_comps supplied. If sel.comp is "chi", an optimization is also

performed, but parsimonious solutions are sought through a chi-square chriterion. When setting sel.comp to a numeric vector, exact selection of number of components is performed.

When setting B to a number, e.g. 200, the procedures above are repeated B times using bootstrapping to estimate standard deviations of the cross-validated explained variances.

Value

An object of type `SO_TDI` containing total, direct and indirect effects, plus possibly additional effects and standard deviations (estimated by bootstrapping).

References

- Menichelli, E., Almøy, T., Tomic, O., Olsen, N. V., & Næs, T. (2014). SO-PLS as an exploratory tool for path modelling. Food quality and preference, 36, 122-134.
- Næs, T., Romano, R., Tomic, O., Måge, I., Smilde, A., & Liland, K. H. (2020). Sequential and orthogonalized PLS (SO-PLS) regression for path analysis: Order of blocks and relations between effects. Journal of Chemometrics, e3243.

See Also

Overviews of available methods, `multiblock`, and methods organised by main structure: `basic`, `unsupervised`, `asca`, `supervised` and `complex`.

Examples

```
# Single path for the potato data:
data(potato)
pot.pm <- sopls_pm(potato[1:3], potato[['Sensory']], c(5,5,5), computeAdditional=TRUE)
pot.pm

# Corresponding SO-PLS model:
# so <- sopls(Sensory ~ ., data=potato[c(1,2,3,9)], ncomp=c(5,5,5), validation="CV", segments=10)
# maageSeq(pot.so, compSeq = c(3,2,4))

# All path in the forward direction for the wine data:
data(wine)
pot.pm.multiple <- sopls_pm_multiple(wine, ncomp = c(4,2,9,8))
pot.pm.multiple
```

`sopls` *Sequential and Orthogonalized PLS (SO-PLS)*

Description

Function for computing standard SO-PLS based on the interface of the `pls` package.

Usage

```
sopls(
  formula,
  ncomp,
  max_comps = min(sum(ncomp), 20),
  data,
  subset,
  na.action,
  scale = FALSE,
  validation = c("none", "CV", "LOO"),
  sequential = FALSE,
  segments = 10,
  sel.comp = "opt",
  progress = TRUE,
  ...
)
```

Arguments

formula	Model formula accepting a single response (block) and predictor block names separated by + signs.
ncomp	Numeric vector of components per block or scalar of overall maximum components.
max_comps	Maximum total number of components from all blocks combined (<= sum(ncomp)).
data	The data set to analyse.
subset	Expression for subsetting the data before modelling.
na.action	How to handle NAs (no action implemented).
scale	Logical indicating if variables should be scaled.
validation	Optional cross-validation strategy "CV" or "LOO".
sequential	Logical indicating if optimal components are chosen sequentially or globally (default=FALSE).
segments	Optional number of segments or list of segments for cross-validation. (See [pls::cvsegments()]).
sel.comp	Character indicating if sequential optimal number of components should be chosen as minimum RMSECV ('opt', default) or by Chi-square test ('chi').
progress	Logical indicating if a progress bar should be displayed while cross-validating.
...	Additional arguments to underlying methods.

Details

SO-PLS is a method which handles two or more input blocks by sequentially performing PLS on blocks against a response and orthogonalising the remaining blocks on the extracted components. Component number optimisation can either be done globally (best combination across blocks) or sequentially (determine for one block, move to next, etc.).

Value

An sopls, mvr object with scores, loadings, etc. associated with printing (sopls_results) and plotting methods (sopls_plots).

References

Jørgensen K, Mevik BH, Næs T. Combining designed experiments with several blocks of spectroscopic data. Chemometr Intell Lab Syst. 2007;88(2): 154–166.

See Also

SO-PLS result functions, sopls_results, SO-PLS plotting functions, sopls_plots, SO-PLS Måge plot, maage, and SO-PLS path-modelling, SO_TDI. Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex.

Examples

```
data(potato)
so <- sopls(Sensory ~ Chemical + Compression, data=potato, ncomp=c(10,10),
            max_comps=10, validation="CV", segments=10)
summary(so)

# Scatter plot matrix with two first components from Chemical block
# and 1 component from the Compression block.
scoreplot(so, comps=list(1:2,1), ncomp=2, block=2)

# Result functions and more plots for SO-PLS
# are found in ?sopls_results and ?sopls_plots.
```

---

sopls_plots *Scores, loadings and plots for sopls objects*

Description

Extraction of scores and loadings and adaptation of scoreplot, loadingplot and biplot from package pls for sopls objects.

Usage

```
## S3 method for class 'sopls'
loadings(object, ncomp = "all", block = 1, y = FALSE, ...)

## S3 method for class 'sopls'
scores(object, ncomp = "all", block = 1, y = FALSE, ...)

## S3 method for class 'sopls'
scoreplot(
  object,
```

```
  comps = 1:2,
  ncomp = NULL,
  block = 1,
  labels,
  identify = FALSE,
  type = "p",
  xlab,
  ylab,
  ...
)

## S3 method for class 'sopls'
loadingplot(
  object,
  comps = 1:2,
  ncomp = NULL,
  block = 1,
  scatter = TRUE,
  labels,
  identify = FALSE,
  type,
  lty,
  lwd = NULL,
  pch,
  cex = NULL,
  col,
  legendpos,
  xlab,
  ylab,
  pretty.xlabels = TRUE,
  xlim,
  ...
)

## S3 method for class 'sopls'
corrplot(
  object,
  comps = 1:2,
  ncomp = NULL,
  block = 1,
  labels = TRUE,
  col = 1:5,
  plotx = TRUE,
  ploty = TRUE,
  ...
)

## S3 method for class 'sopls'
```

```
biplot(
  x,
  comps = 1:2,
  ncomp = "all",
  block = 1,
  which = c("x", "y", "scores", "loadings"),
  var.axes = FALSE,
  xlabs,
  ylabs,
  main,
  ...
)
```

Arguments

object	sopls object
ncomp	integer vector giving components from all blocks before block (see next argument).
block	integer indicating which block to extract components from.
y	logical extract Y loadings/scores instead of X loadings/scores (default = FALSE).
...	further arguments sent to the underlying plot function(s)
comps	integer vector giving components, within block, to plot (see Details regarding combination of blocks).
labels	character indicating if "names" or "numbers" should be plot symbols (optional).
identify	logical for activating identify to interactively identify points.
type	character for selecting type of plot to make. Defaults to "p" (points) for scatter plots and "l" (lines) for line plots.
xlab	character text for x labels.
ylab	character text for y labels.
scatter	logical indicating if a scatterplot of loadings should be made (default = TRUE).
lty	Vector of line type specifications (see par for details).
lwd	numeric vector of line width specifications.
pch	Vector of point specifications (see points for details).
cex	numeric vector of plot size expansions (see par for details).
col	integer vector of symbol/line colours (see par for details).
legendpos	character indicating legend position (if scatter is FALSE), e.g. legendpos = "topright".
pretty.xlabels	logical indicating if xlabels should be more nicely plotted (default = TRUE).
xlim	numeric vector of length two, with the x limits of the plot (optional).
plotx	locical or integer/character. Whether to plot the X correlation loadings, optionally which block(s). Defaults to TRUE.

ploty	logical. Whether to plot the Y correlation loadings. Defaults to TRUE.
x	sopls object
which	character for selecting type of biplot ("x" = default, "y", "scores", "loadings").
var.axes	logical indicating if second axes of a biplot should have arrows.
xlabs	character vector for labelling first set of biplot points (optional).
ylabs	character vector for labelling second set of biplot points (optional).
main	character for setting the main title of a plot.

Details

If comps is supplied as a list for scoreplot, it is assumed that its elements refer to each of the blocks up to block number block. For instance comps = list(1, 0, 1:2) will select 1 component from the first block, no components from the second block and the first two components from the last block. This must be matched by ncomp, specifying how many components were selected before block number block.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results are found in sopls_results.

#' @return The score and loading functions return scores and loadings, while plot functions have no return (except use of 'identify').

Examples

```
data(potato)
so <- sopls(Sensory ~ Chemical + Compression + NIRraw, data=potato, ncomp=c(5,5,5))

# Loadings
loadings(so, ncomp=c(3), block=2)[, 1:3]

# Scores
scores(so, block=1)[, 1:4]

# Default plot from first block
scoreplot(so)

# Second block with names
scoreplot(so, ncomp=c(3), block=2, labels="names")

# Scatterplot matrix
scoreplot(so, ncomp=c(3,2), block=3, comps=1:3)

# Combination of blocks (see Details)
scoreplot(so, ncomp=c(3,2), block=3, comps=list(1,0,1))

# Default plot from first block
loadingplot(so, scatter=TRUE)
```

```
# Second block with names
loadingplot(so, ncomp=c(3), block=2, labels="names", scatter=TRUE)

# Scatterplot matrix
loadingplot(so, ncomp=c(3,2), block=3, comps=1:3, scatter=TRUE)

# Correlation loadings
corrplot(so, block=2, ncomp=1)

# Default plot from first block
biplot(so)
```

`sopls_results` *Result functions for SO-PLS models*

Description

Standard result functions for SO-PLS (`sopls`).

Usage

```
## S3 method for class 'sopls'
predict(
  object,
  newdata,
  ncomp = object$ncomp,
  type = c("response", "scores"),
  na.action = na.pass,
  ...
)

## S3 method for class 'sopls'
coef(object, ncomp = object$ncomp, intercept = FALSE, ...)

## S3 method for class 'sopls'
print(x, ...)

## S3 method for class 'sopls'
summary(
  object,
  what = c("all", "validation", "training"),
  digits = 4,
  print.gap = 2,
  ...
)

classify(object, ...)
```

```
## S3 method for class 'sopls'
classify(object, classes, newdata, ncomp, LQ = "LDA", ...)

## S3 method for class 'sopls'
R2(object, estimate, newdata, ncomp = "all", individual = FALSE, ...)

## S3 method for class 'sopls'
RMSEP(object, estimate, newdata, ncomp = "all", individual = FALSE, ...)

pcp(object, ...)

## S3 method for class 'sopls'
pcp(object, ncomp, ...)

## Default S3 method:
pcp(object, X, ...)

cvanova(pred, ...)

## Default S3 method:
cvanova(pred, true, absRes = TRUE, ...)

## S3 method for class 'sopls'
cvanova(pred, comps, absRes = TRUE, ...)

## S3 method for class 'cvanova'
print(x, ...)

## S3 method for class 'cvanova'
summary(object, ...)

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

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

Arguments

object	A sopls object.
newdata	Optional new data with the same types of predictor blocks as the ones used for fitting the object.
ncomp	An integer vector giving the exact components to apply.
type	A character for predict indicating if responses or scores should be predicted (default = "response", or "scores"), for summary indicating which type of explained variance to compute (default = "train", alternative = "CV").
na.action	Function determining what to do with missing values in newdata.

...	Additional arguments. Currently not implemented.
intercept	A logical indicating if coefficients for the intercept should be included (default = FALSE).
x	A sopls object.
what	A character indicating if summary should include all, validation or training.
digits	The number of digits used for printing.
print.gap	Gap between columns when printing.
classes	A character vector of class labels.
LQ	A character indicating if 'max' (maximum score value), 'lda' or 'qda' should be used when classifying.
estimate	A character indicating if 'train', 'CV' or 'test' results should be displayed.
individual	A logical indicating if results for individual responses should be displayed.
X	A list of data blocks.
pred	An object holding the CV-predicted values (sopls, matrix or list of vectors)
true	A numeric of true response values for CVANOVA.
absRes	A logical indicating if absolute (TRUE) or squared (FALSE) residuals should be computed.
comps	An integer vector giving the exact components to apply.

Details

The parameter ncomp controls which components to apply/extract, resulting in the sequence of components leading up to the specific choice, i.e. ncomp = c(2,2,1) results in the sequence 1,0,0; 2,0,0; 2,1,0; 2,2,0; 2,2,1. Usage of the functions are shown using generics in the examples below. Prediction, regression coefficients, object printing and summary are available through: predict.sopls, coef.sopls, print.sopls and summary.sopls. Explained variances and RMSEP are available through R2.sopls and RMSEP.sopls. Principal components of predictions are available through pcp.sopls. Finally, there is work in progress on classifcation support through classify.sopls.

Value

Returns depend on method used, e.g. predict.sopls returns predicted responses or scores depending on inputs, coef.sopls return regression coefficients, while print and summary methods return the object invisibly.

References

Jørgensen K, Mevik BH, Næs T. Combining designed experiments with several blocks of spectroscopic data. Chemometr Intell Lab Syst. 2007;88(2): 154–166.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for plotting are found in sopls_plots.

Examples

```
data(potato)
mod <- sopls(Sensory[,1] ~ ., data = potato[c(1:3,9)], ncomp = 5, subset = 1:20)
testset <- potato[-(1:20),]; testset$Sensory <- testset$Sensory[,1,drop=FALSE]
predict(mod, testset, ncomp=c(2,1,2))
dim(coef(mod, ncomp=c(3,0,1))) # <variables x responses x components>
R2(mod, ncomp = c(4,1,2))
print(mod)
summary(mod)

# PCP from sopls object
modMulti <- sopls(Sensory ~ ., data = potato[c(1:3,9)], ncomp = 5, validation = "CV", segment = 5)
(PCP <- pcp(modMulti, c(2,1,2)))
scoreplot(PCP)

# PCP from matrices
preds <- modMulti$validation$Ypred[,,"2,1,2"]
PCP_default <- pcp(preds, potato[1:3])

# CVANOVA
modCV <- sopls(Sensory[,1] ~ ., data = potato[c(1:3,9)], ncomp = 5, validation = "CV", segment = 5)
summary(cva <- cvanova(modCV, "2,1,2"))
plot(cva)
```

`statis` *Structuration des Tableaux à Trois Indices de la Statistique - STATIS*

Description

This is a wrapper for the `ade4::statis` function for computing STATIS.

Usage

```
statis(X, ncomp = 3, scannf = FALSE, tol = 1e-07, ...)
```

Arguments

`X`	`list` of input blocks.
`ncomp`	`integer` number of components to extract.
`scannf`	`logical` indicating if eigenvalue bar plot shoulde be displayed.
`tol`	`numeric` eigenvalue threshold tolerance.
`...`	additional arguments (not used).

Details

STATIS is a method, related to MFA, for analysing two or more blocks. It also decomposes the data into a low-dimensional subspace but uses a different scaling of the individual blocks.

Value

multiblock object including relevant scores and loadings. Relevant plotting functions: multiblock_plots and result functions: multiblock_results.

References

Lavit, C.; Escoufier, Y.; Sabatier, R.; Traissac, P. (1994). The ACT (STATIS method). Computational Statistics & Data Analysis. 18: 97

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(candies)
candyList <- lapply(1:nlevels(candies$candy),function(x)candies$assessment[candies$candy==x,])
can.statis <- statis(candyList)
plot(scores(can.statis), labels="names")
```

---

supervised *Supervised Multiblock Methods*

Description

Collection of supervised multiblock methods:

- MB-PLS - Multiblock Partial Least Squares (mbpls)
- sMB-PLS - Sparse Multiblock Partial Least Squares (smbpls)
- SO-PLS - Sequential and Orthogonalized PLS (sopls)
- PO-PLS - Parallel and Orthogonalized PLS (popls)
- ROSA - Response Oriented Sequential Alternation (rosa)
- mbRDA - Multiblock Redundancy Analysis (mbrda)

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

```
data(potato)
mb <- mbpls(Sensory ~ Chemical + Compression, data=potato, ncomp = 5)
print(mb)

# Convert data.frame with AsIs objects to list of matrices
potatoList <- lapply(potato, unclass)
mbr <- mbrda(Sensory ~ Chemical + Compression, data=potatoList, ncomp = 10)
print(mbr)
scoreplot(mbr, labels="names")
```

---

unique_combos *Unique combinations of blocks*

Description

Compute a list of all possible block combinations where the number of blocks in each combination is limited by parameters `min_level` and `max_level`.

Usage

```
unique_combos(n_block, max_level, min_level = 2)
```

Arguments

`n_block`	`integer` number of input blocks.
`max_level`	`integer` maximum number of blocks per combination.
`min_level`	`integer` minimum number of blocks per combination.

Details

This function is used for minimal redundancy implementations of `rosa` and `sopls` and for lookups into computed components.

Value

A list of unique block combinations.

Examples

```
unique_combos(3, 2)
```

unsupervised *Unsupervised Multiblock Methods*

Description

Collection of unsupervised multiblock methods:

- SCA - Simultaneous Component Analysis (sca)
- GCA - Generalized Canonical Analysis (gca)
- GPA - Generalized Procrustes Analysis (gpa)
- MFA - Multiple Factor Analysis (mfa)
- PCA-GCA (pcagca)
- DISCO - Distinctive and Common Components with SCA (disco)
- HPCA - Hierarchical Principal component analysis (hpca)
- MCOA - Multiple Co-Inertia Analysis (mcoa)
- JIVE - Joint and Individual Variation Explained (jive)
- STATIS - Structuration des Tableaux à Trois Indices de la Statistique (statis)
- HOGSVD - Higher Order Generalized SVD (hogsvd)

Details

Original documentation of STATIS: statis. JIVE, STATIS and HOGSVD assume variable linked matrices/data.frames, while SCA handles both links.

See Also

Overviews of available methods, multiblock, and methods organised by main structure: basic, unsupervised, asca, supervised and complex. Common functions for computation and extraction of results and plotting are found in multiblock_results and multiblock_plots, respectively.

Examples

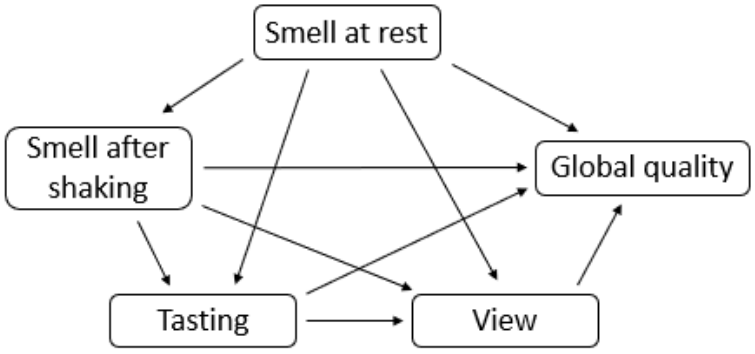
```
# Object linked data
data(potato)
potList <- as.list(potato[c(1,2,9)])
pot.sca    <- sca(potList)

# Variable linked data
data(candies)
candyList <- lapply(1:nlevels(candies$candy),function(x)candies$assessment[candies$candy==x,])
can.statis <- statis(candyList)
plot(can.statis$statis)
```

`wine` *Wines of Val de Loire*

Description

This dataset contains sensory assessment of 21 wines. The assessments are grouped according to the tasting process and thus have a natural ordering with a all blocks pointing forward to all remaining blocks in the process.



Usage

```
data(wine)
```

Format

A data.frame having 21 rows and 5 variables:

Smell at rest Matrix of sensory assessments

View Matrix of sensory assessments

Smell after shaking Matrix of sensory assessments

Tasting Matrix of sensory assessments

Global quality Matrix of sensory assessments

References

Escofier B, Pages L. Analyses Factorielles Simples and Multiples. Paris: Dunod; 1988.

Index

∗ **multivariate**
    mvrVal, 37
∗ **regression**
    mvrVal, 37

asca, *3*, *5–7*, *13–18*, *21*, *23–25*, *27*, *29–31*, *35*, *37*, *43*, *45*, *47*, *51*, *54*, *56*, *58*, *60*, *63*, *66*, *68*, *71*, *74*, *76*, *78*

barplot.rosa *(*rosa_plots*)*, 55
basic, 3, *3*, *5–7*, *13–18*, *21*, *23–25*, *27*, *29–31*, *35*, *37*, *43*, *45*, *47*, *51*, *54*, *56*, *58*, *60*, *63*, *66*, *68*, *71*, *74*, *76*, *78*
biplot.multiblock *(*multiblock_plots*)*, 33
biplot.sopls *(*sopls_plots*)*, 68
block.data.frame, 4
block.preprocess *(*preprocess*)*, 50
blockexpl *(*rosa_results*)*, 56

candies, 4
cca, *3*, 5
classify *(*sopls_results*)*, 72
coef.rosa *(*rosa_results*)*, 56
coef.sopls *(*sopls_results*)*, 72
complex, *3*, *5*, 6, *6*, *7*, *13–18*, *21*, *23–25*, *27*, *29–31*, *35*, *37*, *43*, *45*, *47*, *51*, *54*, *56*, *58*, *60*, *63*, *66*, *68*, *71*, *74*, *76*, *78*
compnames, 6
corrplot *(*multiblock_plots*)*, 33
corrplot.sopls *(*sopls_plots*)*, 68
cvanova *(*sopls_results*)*, 72

data.frame, *11*
disco, 7, *78*
DISCOsca, 8
dummycode, 9

emulsions, 10
explvar, 10
extended.model.frame, 11

gca, 12, *47*, *78*
gpa, 13, *78*
gsvd, *3*, 14

hogsvd, 15, *78*
hpca, 16, *78*

ifa, *3*, 17
image.rosa *(*rosa_plots*)*, 55

jive, 18, *78*

loadingplot.multiblock *(*multiblock_plots*)*, 33
loadingplot.sopls *(*sopls_plots*)*, 68
loadings.multiblock *(*multiblock_results*)*, 36
loadings.rosa *(*rosa_results*)*, 56
loadings.sopls *(*sopls_plots*)*, 68
loadingweightplot *(*multiblock_plots*)*, 33
lpls, *6*, 19, *21*, *23*
lpls_results, 21, *21*
lplsCV *(*lpls_results*)*, 21
lplsData, 23

maage, 24, *68*
maageSeq *(*maage*)*, 24
mbpls, 26, *39*, *50*, *61*, *76*
mbrda, 28, *76*
mcoa, 29, *78*
mcolors, 30
mfa, 31, *78*
mobile, 32
model.frame, *11*
MSEP.mbpls *(*mvrVal*)*, 37
multiblock, *3*, *5–7*, *13–18*, *21*, *23–25*, *27*, *29–31*, *35*, *37*, *43*, *45*, *47*, *51*, *54*, *56*, *58*, *60*, *63*, *66*, *68*, *71*, *74*, *76*, *78*
multiblock_plots, *5*, *7*, *12–18*, *20*, *27–31*, 33, *37*, *43–45*, *47*, *51*, *59*, *60*, *63*, *76*, *78*

multiblock_results, *5*, *7*, *12–18*, *20*, *27–31*, *35*, 36, *43–45*, *47*, *51*, *59*, *60*, *63*, *76*, *78*
mvr, *50*
mvrVal, 37
mvrValstats.mbpls *(*mvrVal*)*, 37

na.omit, *49*

par, *35*, *70*
path_effects, 40
path_effects(), *46*, *53*
pca, *3*, 42
pcagca, 43, *78*
pcp *(*sopls_results*)*, 72
pcr, *3*
plot.cvanova *(*sopls_results*)*, 72
plot.lpls *(*lpls_results*)*, 21
plot.path_effects, 45
plot.path_effects(), *41*, *53*
pls::plsr(), *40*
plsr, *3*, *27*
points, *35*, *70*
popls, 46, *76*
potato, 48
predict.lpls *(*lpls_results*)*, 21
predict.mbpls, 49
predict.rosa *(*rosa_results*)*, 56
predict.sopls *(*sopls_results*)*, 72
preprocess, 50
print.cvanova *(*sopls_results*)*, 72
print.multiblock *(*multiblock_results*)*, 36
print.path_effects, 52
print.path_effects(), *41*, *46*
print.rosa *(*rosa_results*)*, 56
print.rosaexpl *(*rosa_results*)*, 56
print.SO_TDI *(*SO_TDI*)*, 64
print.SO_TDI_multiple *(*SO_TDI*)*, 64
print.sopls *(*sopls_results*)*, 72

R2.mbpls *(*mvrVal*)*, 37
R2.sopls *(*sopls_results*)*, 72
residuals.sopls *(*sopls_results*)*, 72
RMSEP.mbpls *(*mvrVal*)*, 37
RMSEP.sopls *(*sopls_results*)*, 72
rosa, 53, *55*, *56*, *76*, *77*
rosa.classify *(*rosa_results*)*, 56
rosa_plots, *54*, 55, *58*
rosa_results, *54*, 56, *56*, *58*

sca, 59, *78*
scoreplot.multiblock *(*multiblock_plots*)*, 33
scoreplot.sopls *(*sopls_plots*)*, 68
scores.multiblock *(*multiblock_results*)*, 36
scores.rosa *(*rosa_results*)*, 56
scores.sopls *(*sopls_plots*)*, 68
simulated, 60
smbpls, 61, *76*
SO_TDI, 64, *68*
sopls, *24*, 66, *72*, *76*, *77*
sopls_plots, 68, *68*, *74*
sopls_pm, *6*
sopls_pm *(*SO_TDI*)*, 64
sopls_pm_multiple *(*SO_TDI*)*, 64
sopls_results, *68*, *71*, 72
statis, 75, *78*
summary.cvanova *(*sopls_results*)*, 72
summary.multiblock *(*multiblock_results*)*, 36
summary.rosa *(*rosa_results*)*, 56
summary.sopls *(*sopls_results*)*, 72
supervised, *3*, *5–7*, *13–18*, *21*, *23–25*, *27*, *29–31*, *35*, *37*, *43*, *45*, *47*, *51*, *54*, *56*, *58*, *60*, *63*, *66*, *68*, *71*, *74*, 76, *76*, *78*

unique_combos, 77
unsupervised, *3*, *5–7*, *13–18*, *21*, *23–25*, *27*, *29–31*, *35*, *37*, *43*, *45*, *47*, *51*, *54*, *56*, *58*, *60*, *63*, *66*, *68*, *71*, *74*, *76*, 78, *78*

wine, 79