Encoding UTF-8
Type Package
Title Multiblock Data Fusion in Statistics and Machine Learning
Version 0.8.3
Date 2022-10-29
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)
https://github.com/khliland/multiblock/
BugReports https://github.com/khliland/multiblock/issues/
Depends R (>= 3.5.0)
Imports ade4, car, FactoMineR, geigen, lme4, MASS, mixlm, plotrix, pls, plsVarSel, pracma, progress, rjive, Rcpp, RegularizedSCA, RGCCA, RSpectra, SSBtools
Suggests rmarkdown, knitr
LinkingTo Rcpp, RcppEigen
RoxygenNote 7.2.1
VignetteBuilder knitr
NeedsCompilation yes
Author Kristian Hovde Liland [aut, cre]
(<https://orcid.org/0000-0001-6468-9423>), Solve Sæbø [ctb], Stefan Schrunner [rev]
Maintainer Kristian Hovde Liland <kristian.liland@nmbu.no>
Repository CRAN
Date/Publication 2022-10-29 21:37:43 UTC

R topics documented:

asca ........................................... 3
asca_plots ................................... 4
asca_results ................................ 6
basic .......................................... 7
block.data.frame .............................. 8
candies ...................................... 9
cca ........................................... 10
complex ..................................... 11
compnames .................................... 11
disco ......................................... 12
dummycode ................................... 13
explvar ....................................... 13
gca ........................................... 14
gpa ........................................... 15
gsvd .......................................... 16
hogsved ....................................... 17
h pca ......................................... 18
ifa ........................................... 19
jive .......................................... 20
lpls ........................................... 21
lplslData .................................... 23
lplsl_results ................................ 24
maage ........................................ 26
mblpl ........................................ 28
mbrrda ....................................... 30
mcoa ......................................... 31
mcolors ...................................... 32
mfa ........................................... 33
multiblock ................................... 34
multiblock_plots .............................. 35
multiblock_results ............................ 38
pca ........................................... 40
pcagca ....................................... 41
popls ......................................... 42
potato ........................................ 44
rosa .......................................... 45
rosa_plots ................................... 47
rosa_results ................................ 48
sca ........................................... 51
simulated .......................... 52
smbpl ........................................ 53
sopls ........................................ 55
### Description
This is a quite general and flexible implementation of ASCA.

### Usage

```
asca(formula, data, subset, weights, na.action, family, pca.in = FALSE)
```

### Arguments
- **formula**: Model formula accepting a single response (block) and predictor names separated by + signs.
- **data**: The data set to analyse.
- **subset**: Subset of objects
- **weights**: Optional object weights.
- **na.action**: How to handle NAs (no action implemented).
- **family**: Error distributions and link function for Generalized Linear Models.
- **pca.in**: Compress response before ASCA (number of components).

### Details
ASCA is a method which decomposes a multivariate response according to one or more design variables. ANOVA is used to split variation into contributions from factors, and PCA is performed on the corresponding least squares estimates, i.e., $Y = X_1 B_1 + X_2 B_2 + \ldots + E = T_1 P_1' + T_2 P_2' + \ldots + E$. This version of ASCA encompasses LiMM-PCA, generalized ASCA and covariates ASCA. It includes confidence ellipsoids for the balanced fixed effect ASCA.

### Value
An `asca` object containing loadings, scores, explained variances, etc. The object has associated plotting (`asca_plots`) and result (`asca_results`) functions.
References


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 asca_results and asca_plots, respectively.

Examples

```r
# Load candies data
data(candies)

# Basic ASCA model with two factors
mod <- asca(assessment ~ candy + assessor, data=candies)
print(mod)

# ASCA model with interaction
mod <- asca(assessment ~ candy * assessor, data=candies)
print(mod)

# Result plotting for first factor
loadingplot(mod, scatter=TRUE, labels="names")
scoreplot(mod)

# ASCA model with compressed response using 5 principal components
mod.pca <- asca(assessment ~ candy + assessor, data=candies, pca.in=5)

# Mixed Model ASCA, random assessor
mod.mix <- asca(assessment ~ candy + (1|assessor), data=candies)
scoreplot(mod.mix)
```

### asca_plots

**ASCA Result Methods**

**Description**

Various plotting procedures for asca objects.
Usage

```r
## S3 method for class 'asca'
loadingplot(object, factor = 1, comps = 1:2, ...)

## S3 method for class 'asca'
scoreplot(
  object,
  factor = 1,
  comps = 1:2,
  pch.scores = 19,
  pch.projections = 1,
  gr.col = 1:nlevels(object$effects[[factor]]),
  ellipsoids,
  xlim,
  ylim,
  xlab,
  ylab,
  legendpos,
  ...
)
```

Arguments

- `object`: asca object.
- `factor`: integer/character for selecting a model factor.
- `comps`: integer vector of selected components.
- `...`: additional arguments to underlying methods.
- `pch.scores`: integer plotting symbol.
- `pch.projections`: integer plotting symbol.
- `gr.col`: integer vector of colours for groups.
- `ellipsoids`: character "confidence" or "data" ellipsoids for balanced fixed effect models.
- `xlim`: numeric x limits.
- `ylim`: numeric y limits.
- `xlab`: character x label.
- `ylab`: character y label.
- `legendpos`: character position of legend.

Details

Usage of the functions are shown using generics in the examples below. Plot routines are available as `scoreplot.asca` and `loadingplot.asca`.

Value

The plotting routines have no return.
References


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 asca_results.

<table>
<thead>
<tr>
<th>asca_results</th>
<th>ASCA Result Methods</th>
</tr>
</thead>
</table>

Description

Standard result computation and extraction functions for ASCA (asca).

Usage

```r
## S3 method for class 'asca'
print(x, ...)
## S3 method for class 'asca'
summary(object, ...)
## S3 method for class 'summary.asca'
print(x, digits = 2, ...)
## S3 method for class 'asca'
loadings(object, factor = 1, ...)
## S3 method for class 'asca'
scores(object, factor = 1, ...)
projections(object, ...)
## S3 method for class 'asca'
projections(object, factor = 1, ...)
```
Arguments

- **x**: asca object.
- ... additional arguments to underlying methods.
- **object**: asca object.
- **digits**: integer number of digits for printing.
- **factor**: integer/character for selecting a model factor.

Details

Usage of the functions are shown using generics in the examples in asca. Explained variances are available (block-wise and global) through blockexpl and print.rosaexpl. Object printing and summary are available through: print.asca and summary.asca. Scores and loadings have their own extensions of scores() and loadings() through scores.asca and loadings.asca. Special to ASCA is that scores are on a factor level basis, while back-projected samples have their own function in projections.asca.

Value

Returns depend on method used, e.g. projections.sopls returns projected samples, scores.sopls return scores, while print and summary methods return the object invisibly.

References


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 asca_plots.
Description

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

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

See Also

Overviews of available methods, \texttt{multiblock}, and methods organised by main structure: \texttt{basic, unsupervised, asca, supervised} and \texttt{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))

---

\texttt{block.data.frame} \hspace{1cm} \textit{Block-wise indexable data.frame}

Description

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

Usage

\texttt{block.data.frame}(X, block inds = \texttt{NULL}, to.matrix = \texttt{TRUE})

Arguments

\texttt{X} \hspace{1cm} Either a single \texttt{data.frame} to index or a list of matrices/data.frames
\texttt{block inds} \hspace{1cm} Named list of indexes if \texttt{X} is a single \texttt{data.frame}, otherwise \texttt{NULL}.
\texttt{to.matrix} \hspace{1cm} logical indicating if input list elements should be converted to matrices.
candies

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

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. \( \max(\text{cor}(X_1 \times a, X_2 \times b)) \). 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


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])
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

```r
# 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
```

comnames

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

```r
comnames(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 comnames from the `pls` package to work with multiblock objects.
Value

A character vector of component names.

Description

This is a wrapper for the RegularizedSCA::DISCOsca function 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


See Also

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

Examples

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

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 deficiency 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)

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)

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 = 2, svd = TRUE, tol = 10^-12, corrs = TRUE, ...)

Arguments

X  list of input blocks.
ncomp  integer number of components to extract.
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.
Value

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

References


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.
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


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))

hgsvd

Higher Order Generalized SVD - HOGSVD

Description

This is a simple implementation for computing HOGSVD

Usage

hgsvd(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

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

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

```r
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

```r
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

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

```r
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**

```r
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**


**See Also**

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

```r
# 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

```r
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


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

```r
# 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
```

---

**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**

```r
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**

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

---

**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**

```r
## 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.
- varsizex: 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 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))

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, ...)

måge

Måge plot

Måge plot for SO-PLS (sopls) cross-validation visualisation.
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 dis-
           playing.
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]
sowe <- soplgs("Global quality" ~ ., data=wine, ncomp=ncomp,
               max_comps=10, validation="CV", segments=10)
maage(sowe)

# 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(sowe)
maageSeq(sowe, 2)
maageSeq(sowe, c(2,1))
maageSeq(sowe, 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.
n.a. 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., $Z_{[i]} = X_{[i]} / \text{norm}(X_{[i]}, "F")^2 * \text{blockScale[i]}$.

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


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[['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


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


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 colors 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.
**mfa**

### Examples

```r
mcolors(5)
```

---

**Multiple Factor Analysis - MFA**

### Description

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

### Usage

```r
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 method 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


### 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)
}

Description

A collection of methods for analysis of data sets with more than two blocks of data.

Unsupervised 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 (h pca)
• 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)

Design based methods:

• ASCA - Anova Simultaneous Component Analysis (asca)

Supervised methods:

• MB-PLS - Multiblock Partial Least Squares (mbpls)
• sMB-PLS - Sparse Multiblock Partial Least Squares (sm bpls)
• 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)

Complex methods:
• L-PLS - Partial Least Squares in L configuration (lpls)
• SO-PLS-PM - Sequential and Orthogonalised PLS Path Modelling (sopls_pm)

**Single- and two-block methods:**
• 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)

**Datasets:**
• Sensory assessment of candies (candies)
• Sensory, rheological, chemical and spectroscopic analysis of potatoes (potato)
• Data simulated to have certain characteristics (simulated)
• Wines of Val de Loire (wine)

**Utility functions:**
• Block-wise indexable data.frame (block.data.frame)
• Dummy-code a vector (dummycode)

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

---

**multiblock_plots**

*Plot Functions for Multiblock Objects*

**Description**

Plotting procedures for multiblock objects.

**Usage**

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

### loadingplot

```r
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

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

## S3 method for class 'multiblock'

### biplot

```r
biplot(x,
       block = 0,
       comps = 1:2,
       which = c("x", "y", "scores", "loadings"),
       var.axes = FALSE,
       xlabs,
       ylabs,
       main,
       ...}
```

### corrplot

```r
corrplot(object, ...)
```

## Default S3 method:

```r
corrplot(object, ...)
```
## multiblock_plots

### S3 method for class 'mvr'

```r
corrplot(object, ...)
```

### S3 method for class 'multiblock'

```r
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.
- **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").
multiblock_results

**Description**

Standard result computation and extraction functions for multiblock objects.

**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**

```r
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)))
```
Usage

```r
## 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()` through `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

```r
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))
```
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


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 method 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


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.pca <- pca(potList)

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

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

---

### 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**

```r
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.
- **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`.

Value

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

References


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[['Var 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")

---

**Description**

A dataset containing 9 blocks of measurements on 26 potatoes. Original dataset can be found at [http://models.life.ku.dk/Texture_Potatoes](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**

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


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.
Description

Various plotting procedures for *rosa* objects.

Usage

```r
## 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


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)
...)

## 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 use apply.
comps An integer vector giving the exact components to apply.
type 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/\texttt{FALSE}) or component-wise (\texttt{TRUE}) explained variance should be printed.
classes  A character vector of class labels.
LQ  A character indicating if \textquote{max} (maximum score value), \textquote{lda} or \textquote{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: \texttt{predict.rosa}, \texttt{coef.rosa}, \texttt{print.rosa} and \texttt{summary.rosa}. Explained variances are available (block-wise and global) through \texttt{blockexpl} and \texttt{print.rosaexpl}. Scores and loadings have their own extensions of \texttt{scores()} and \texttt{loadings()} through \texttt{scores.rosa} and \texttt{loadings.rosa}. Finally, there is work in progress on classification support through \texttt{rosa.classify}.

Value

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

References


See Also

Overviews of available methods, \texttt{multiblock}, and methods organised by main structure: \texttt{basic}, \texttt{unsupervised}, \texttt{asca}, \texttt{supervised} and \texttt{complex}. Common functions for computation and extraction of results and plotting are found in \texttt{rosa_results} and \texttt{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)
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 implementation 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


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

```r
# Object linked data
data(potato)
potList <- as.list(potato[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
smbpls

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 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

sopls

Sequential and Orthogonalized PLS (SO-PLS)

Description

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


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))
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.
segments Optional number of segments or list of segments for cross-validation. (See plsp::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.).
sopls_plots

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).
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 logical or integer/character. Whether to plot the X correlation loadings, optionally which block(s). Defaults to TRUE.
sopls_plots

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.

Examples

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

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

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

# Default plot from first block
scoreplot(s)

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

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

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

# Default plot from first block
loadingplot(s, 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  
\textit{Result functions for SO-PLS models}

\section*{Description}
Standard result functions for SO-PLS (\texttt{sopls}).

\section*{Usage}

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

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

\texttt{## S3 method for class 'sopls'}
\texttt{print(x, ... )}

\texttt{## S3 method for class 'sopls'}
\texttt{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, ...)

### 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**  
A logical indicating if coefficients for the intercept should be included (default = FALSE).
A sopls object.

A character indicating if summary should include all, validation or training.

The number of digits used for printing.

Gap between columns when printing.

A character vector of class labels.

A character indicating if 'max' (maximum score value), 'lda' or 'qda' should be used when classifying.

A character indicating if 'train', 'CV' or 'test' results should be displayed.

A logical indicating if results for individual responses should be displayed.

A list of data blocks.

An object holding the CV-predicted values (sopls, matrix or list of vectors)

A numeric of true response values for CVANOVA.

A logical indicating if absolute (TRUE) or squared (FALSE) residuals should be computed.

An integer vector giving the exact components to apply.

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 classification support through classify.sopls.

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.

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

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)

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 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 criterion. 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


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)
# mamageSeq(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))
Description

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

Usage

```r
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 should 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


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 (h pca)
• MCOA - Multiple Co-Inertia Analysis (m coa)
• 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

```r
# 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

```r
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

Index

asca, 3, 4, 6–8, 10–12, 15–20, 23–25, 28, 29, 31–35, 38–40, 42, 43, 46, 48, 50, 51, 55, 57, 60, 63, 66–68, 70
asca_plots, 3, 4, 7
asca_results, 3, 4, 6
barplot.rosa (rosa_plots), 47
basic, 4, 6, 7, 7, 8, 10–12, 15–20, 23–25, 28, 29, 31–33, 35, 38–40, 42, 43, 46, 48, 50, 51, 55, 57, 60, 63, 66–68, 70
biplot.multiblock (multiblock_plots), 35
biplot.sopls (sopls_plots), 57
block.data_frame, 8, 35
blockexpl (rosa_results), 48
candies, 9, 35
cca, 8, 10, 35
classify (sopls_results), 61
coef.rosa (rosa_results), 48
coef.sopls (sopls_results), 61
complex, 4, 6–8, 10, 11, 11, 12, 15–20, 23–25, 28, 29, 31–33, 35, 38–40, 42, 43, 46, 48, 50, 51, 55, 57, 60, 63, 66–68, 70
compnames, 11
corrplot (multiblock_plots), 35
corrplot.sopls (sopls_plots), 57
covar.sopls (sopls_results), 61
disco, 12, 34, 70
dummycode, 13, 35
explvar, 13
gca, 14, 34, 43, 69
gpa, 15, 34, 69
gsvd, 8, 16, 35
hogsvid, 17, 34, 70
hpca, 18, 34, 70
ifa, 8, 19, 35
image.rosa (rosa_plots), 47
jive, 20, 34, 70
loadingplot.asca (asca_plots), 4
loadingplot.multiblock (multiblock_plots), 35
loadingplot.sopls (sopls_plots), 57
loadings.asca (asca_results), 6
loadings.multiblock (multiblock_results), 38
loadings.rosa (rosa_results), 48
loadings.sopls (sopls_results), 57
loadingweightplot (multiblock_plots), 35
lpls, 11, 21, 23, 24, 35
lpls_results, 23, 24
lplscv (lpls_results), 24
lplsd, 23
maage, 26, 57
maageSeq (maage), 26
mbpl.s, 28, 34, 53, 68
mbrrd, 30, 34, 68
mcoa, 31, 34, 70
mcolor, 32
mf, 33, 34, 69
multiblock, 4, 6–8, 10–12, 15–20, 23–25, 28, 29, 31–33, 34, 35, 38–40, 42, 43, 46, 48, 50, 51, 55, 57, 60, 63, 66–68, 70
multiblock_plots, 10, 12, 15–20, 22, 29, 30, 32, 33, 35, 39, 40, 42, 43, 51, 54, 67, 68, 70
multiblock_results, 10, 12, 15–20, 22, 29, 30, 32, 33, 38, 38, 40, 42, 43, 51, 54, 67, 68, 70
par, 37, 59
pca, 8, 35, 40
pcagca, 34, 41, 69
pcp (sopls_results), 61

72
pcr, 8, 35
plot.cvanova (sopls_results), 61
plot.lpls (lpls_results), 24
pls, 43
plsr, 8, 29, 35
points, 37, 59
popls, 34, 42, 68
potato, 35, 44
predict.lpls (lpls_results), 24
predict.rosa (rosa_results), 48
predict.sopls (sopls_results), 61
print.asca (asca_results), 6
print.cvanova (sopls_results), 61
print.multiprob (multiprob_results), 38
print.rosa (rosa_results), 48
print.rosaexpl (rosa_results), 48
print.SO_TDI (SO_TDI), 64
print.SO_TDI_multiple (SO_TDI), 64
print.sopls (sopls_results), 61
print.summary.asca (asca_results), 6
projections (asca_results), 6
R2.sopls (sopls_results), 61
RMSEP.sopls (sopls_results), 61
rosa, 34, 45, 47, 48, 68, 69
rosa.classify (rosa_results), 48
rosa_plots, 46, 47, 50
rosa_results, 46, 48, 48, 50
sca, 34, 51, 69
scoreplot.asca (asca_plots), 4
scoreplot.multiprob
 (multiprob_plots), 35
scoreplot.sopls (sopls_plots), 57
scores.asca (asca_results), 6
scores.multiprob (multiprob_results), 38
scores.rosa (rosa_results), 48
scores.sopls (sopls_plots), 57
simulated, 35, 52
smbpls, 34, 53, 68
SO_TDI, 57, 64
sopls, 26, 34, 55, 61, 68, 69
sopls_plots, 57, 57, 63
sopls_pm, 11, 35
sopls_pm (SO_TDI), 64
sopls_pm_multiple (SO_TDI), 64
sopls_results, 57, 60, 61

statis, 34, 67, 70
summary.asca (asca_results), 6
summary.cvanova (sopls_results), 61
summary.multiprob
 (multiprob_results), 38
summary.rosa (rosa_results), 48
summary.sopls (sopls_results), 61
supervised, 4, 6–8, 10–12, 15–20, 23–25, 28,
  29, 31–33, 35, 38–40, 42, 43, 46, 48,
  50, 51, 55, 57, 60, 63, 66–68, 68, 70
unique_combos, 69
unsupervised, 4, 6–8, 10–12, 15–20, 23–25,
  28, 29, 31–33, 35, 38–40, 42, 43, 46,
  48, 50, 51, 55, 57, 60, 63, 66–68, 69,
  70
wine, 35, 70