Package 'R.ComDim'

May 13, 2026

Type Package

Title Common Dimensions (ComDim) Multi-Block Analysis

Version 1.0.0

Description Common Dimensions (ComDim) is a multi-block method that simultaneously considers multiple data tables to find latent components that are common to all the tables as well as those specific to each data table, along with the contribution of each table to each component. See Jouan-Rimbaud Bouveresse and Rutledge (2024) <doi:10.1002/cem.3454>, Boccard and Rutledge (2013) <doi:10.1016/j.aca.2013.01.022>, and Puig-Castellví et al. (2021) <doi:10.1016/j.chemolab.2021.104422>.

License MIT + file LICENSE

Depends methods

Imports pracma, utils, ggplot2, ConsensusOPLS

Suggests gridExtra, imputeLCMD, ica, MultiAssayExperiment, SummarizedExperiment, ropls

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Collate 'R.ComDim-package.R' 'ComDim-classes.R' 'AddMetadata.R' 'ComDim_Exploratory.R' 'ComDim_PCA.R' 'ComDim_PLS.R' 'ComDim_OPLS.R' 'ComDim_y.R' 'ExpandMultiBlock.R' 'FilterSamplesMultiBlock.R' 'NAInfRemoveMultiBlock.R' 'MakeComDimLoadingsTable.R' 'MakeComDimScoresTable.R' 'MultiAssayExperiment2MultiBlock.R' 'MultiBlock.R' 'MultiBlock2MultiAssayExperiment.R' 'MultiBlock2Matrix.R' 'NormalizeMultiBlock.R' 'PredictMultiBlock.R' 'ProcessMultiBlock.R' 'SelectFeaturesRW.R' 'SimulateMultiBlock.R' 'SplitRW.R' 'SummarizedExperiment2MultiBlock.R' 'data-KEGG_table_metabolites.R' 'data-RNAseq3.R' 'data-extra.R' 'data-gcms.R' 'data-intra.R' 'data-lcms.R' 'data-lipids.R' 'data-metadata_RNAseq3.R' 'data-metadata_lipids.R' 'data-mirnaseq.R' 'data-rnaseq.R'

NeedsCompilation no

Author Francesc Puig-Castellví [aut, cre]

Maintainer Francesc Puig-Castellví <puig.francesc@gmail.com>

Repository CRAN

Date/Publication 2026-05-13 07:40:07 UTC

Contents

AddMetadata *AddMetadata*

Description

Adds or overwrites batch and/or metadata information for one or more blocks of an existing MultiBlock object.

Usage

```
AddMetadata(MB, block = NULL, batches = NULL, metadata = NULL)
```

Arguments

MB	A MultiBlock object.
block	The name of the block to modify (character). If NULL and the MultiBlock contains a single block, that block is used automatically. To modify several blocks at once, pass a character vector of block names.
batches	A numeric vector of batch labels, one per sample in the target block. Replaces any existing Batch entry for that block.
metadata	A data.frame with one row per sample in the target block. Replaces any existing Metadata entry for that block.

Value

The updated MultiBlock.

See Also

MultiBlock, FilterSamplesMultiBlock

Examples

```
b1 <- matrix(rnorm(500), 10, 50) # 10 samples, 50 variables
b2 <- matrix(rnorm(800), 10, 80) # 10 samples, 80 variables
batch_b1 <- rep(1, 10)
meta_b1 <- data.frame(condition = rep(c("A", "B"), 5))
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
# Add batch information to block 'b1':
mb <- AddMetadata(mb, block = "b1", batches = batch_b1)
# Add (or overwrite) metadata for block 'b1':
mb <- AddMetadata(mb, block = "b1", metadata = meta_b1)
```

blockNames *blockNames*

Description

Return the block names of a MultiBlock object.

Usage

```
blockNames(x, ...)

## S4 method for signature 'MultiBlock'
blockNames(x, slot = "Data")
```

Arguments

x	A MultiBlock object.
...	Not used. Present for S4 generic dispatch compatibility.
slot	A string: "Data" (default), "Batch", or "Metadata", indicating which slot to retrieve names from.

Value

A character vector with the block names of the requested slot.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
blockNames(mb) # c("b1", "b2")
blockNames(mb, "Data") # same
```

blockNames<- *blockNames<-*

Description

Set the block names of a MultiBlock object. Renames the Data and Variables slots, and updates Batch and Metadata names to stay consistent.

Usage

```
blockNames(x) <- value

## S4 replacement method for signature 'MultiBlock'
blockNames(x) <- value
```

Arguments

x	A MultiBlock object.
value	A character vector of new block names (same length as the number of blocks).

Value

The updated MultiBlock object.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
blockNames(mb) <- c("block1", "block2")
blockNames(mb) # c("block1", "block2")
```

ComDim-classes *MultiBlock object*

Description

Object of the type `MultiBlock`, to use as input for ComDim analyses.

The output of a ComDim analysis.

Slots

`Samples` vector with the sample names. If this data is not available, the slot will be filled with integers.

`Data` A list with the data-blocks.

`Variables` A character vector with the variable names. If this data is not available, the slot will be filled with integers.

`Batch` A list with the vectors with the batch information for each data-block. Optional.

`Metadata` A list with the samples metadata.

`Method` The algorithm used in the core of the ComDim analysis (ex. PCA, PLS,...)

`ndim` The number of components.

`Q.scores` The Global scores.

`T.scores` The Local scores.

`P.loadings` The Loadings

`Saliences` The Saliences

`R2X` The explained variance of the MultiBlock.

`R2Y` For regression or discriminant models, the explained variance of the Y-block.

`Q2` For regression or discriminant models, the predicted variance of the Y-block.

`DQ2` For discriminant models, the predicted discriminant variance of the Y-block.

`Singular` The singular values.

`Mean` The mean values for each variable in the MultiBlock.

`Norm` The norm values for each variable in the MultiBlock.

`PLS.model` For ComDim analyses using PLS as the core algorithm, contains the W, B, B0 and Y matrices.

`cv` For ComDim_KOPLS, it contains the index of the samples used during the cross-validation.

`Prediction` A list with the predicted Y, the decision rule used for sample classification, the sensitivity, the specificity, and a confusion matrix.

`Metadata` A list with the per-sample metadata for each block.

`variable.block` A vector with the same length as the P.loadings, indicating the block each variable belongs to.

`runtime` The used running time.

`ComDim_Exploratory` *ComDim_Exploratory - Extending matrix decomposition methods to multi-block data.*

Description

Extends any matrix decomposition method used for exploratory purposes to the multi-block field. The user provides a function (`FUN`) to compute the scores from the salience-weighted concatenated blocks; these scores are then used to derive the global scores, local scores, and loadings following the traditional ComDim-PCA framework.

Usage

```
ComDim_Exploratory(
  MB = MB,
  ndim = NULL,
  FUN = FUN,
  normalise = FALSE,
  threshold = 1e-10,
  loquace = FALSE,
  method = "FUN",
  ...
)
```

Arguments

`MB`	A MultiBlock object.
`ndim`	Number of Common Dimensions.
`FUN`	The function used as the core of the ComDim analysis. It must accept a matrix `W` (salience-weighted concatenated blocks) and `ndim` as its first two arguments, and return a matrix of scores with one column per component.

`normalise`	To apply block normalisation. FALSE == no (default), TRUE == yes.
`threshold`	The threshold limit to stop the iterations. Iterations stop when the change in the global score vector is below this value (1e-10 as default).
`loquace`	To display the calculation times. TRUE == yes, FALSE == no (default).
`method`	A string label identifying the decomposition method used (default: 'FUN').
`...`	Additional arguments passed to `FUN`.

Value

A `ComDim` object. Slots for supervised analysis (`R2Y`, `Q2`, `DQ2`, `VIP`, `VIP.block`, `PLS.model`, `cv`, `Prediction`) are empty. The populated slots are:

`Method` The label supplied via the `method` argument.

`ndim` Number of Common Dimensions extracted.

`Q.scores` Global scores matrix ($n \times ndim$). Column names are `CC1`, `CC2`, etc.; row names are sample names. Each column $\mathbf{q}_a$ is a unit-norm consensus score derived from the dominant left direction of FUN applied to the salience-weighted concatenated blocks $\mathbf{W} = [\sqrt{\lambda_1}\mathbf{X}_1 \mid \cdots \mid \sqrt{\lambda_B}\mathbf{X}_B]$.

`T.scores` Named list of block-specific local scores matrices ($n \times ndim$ each). For block b and component a: local loading $\mathbf{p}_{ba} = \mathbf{X}_b'\mathbf{q}_a$ and local score $\mathbf{t}_{ba} = \mathbf{X}_b\,\mathbf{p}_{ba}(\mathbf{p}_{ba}'\mathbf{p}_{ba})^{-1}$.

`P.loadings` Global loadings matrix ($p_{tot} \times ndim$). Column a is $\mathbf{P}_a = \mathbf{X}'\mathbf{q}_a$, where $\mathbf{X}$ is the mean-centred (and optionally normalised) concatenated blocks.

`Saliences` Block salience (weight) matrix ($ntable \times ndim$, row names = block names). Entry (b, a) is $\lambda_{ba} = \mathbf{q}_a'\mathbf{X}_b\mathbf{X}_b'\mathbf{q}_a$, the variance of block b captured by global score a.

`R2X` Proportion of multi-block variance captured by each component (named vector, length $ndim$). Let $\mathbf{s}_a$ be the score vector returned by FUN for component a (before unit-normalisation to obtain $\mathbf{q}_a$), so that $sv_a = \|\mathbf{s}_a\|$; then

$$R2X_a = sv_a^4 / \sum_k sv_k^4 = Singular_a^2 / \sum_k Singular_k^2.$$

`Singular` Squared L2 norms of the FUN score vectors: $Singular_a = sv_a^2 = \|\mathbf{s}_a\|^2$, used to derive `R2X`.

`Mean` List with `MeanMB`: named list of column-mean vectors per block, used for mean-centring.

`Norm` List with `NormMB`: Frobenius norms used for block normalisation (all ones when `normalise = FALSE`).

`variable.block` Character vector (length p_{tot}) indicating the block name of each row in `P.loadings`.

`runtime` Total computation time in seconds.

References

Jouan-Rimbaud Bouveresse D, Rutledge DN (2024). A synthetic review of some recent extensions of ComDim. *Journal of Chemometrics*, 38(5), e3454. doi:10.1002/cem.3454

Examples

```
b1 <- matrix(rnorm(500), 10, 50) # 10 samples, 50 variables
b2 <- matrix(rnorm(800), 10, 80) # 10 samples, 80 variables
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))

if (requireNamespace("ica", quietly = TRUE)) {
  fun.ICA <- function(W, ndim, ...) {
    # W is the concatenated MB.
    # ndim is the number of components.
    result <- ica::ica(W, ndim)
    # The function must return the source estimates (analogous to PCA scores).
    return(result$S)
  }
  resultsICA <- ComDim_Exploratory(mb,
    ndim = 2,
    FUN = fun.ICA,
    method = "ICA"
  )
}
```

ComDim_OPLS *ComDim_OPLS*

Description

Finding common dimensions in multi-block datasets using OPLS. Also known as ConsensusO-PLS (ComDim-OPLS) for multiblock structures: orthogonal components uncorrelated with Y are extracted from all blocks simultaneously before the predictive components are computed.

Usage

```
ComDim_OPLS(
  MB = MB,
  y = y,
  ndim = 1,
  nort = 1,
  method = c("OPLS-DA", "OPLS-R"),
  decisionRule = c("fixed", "max")[2],
  normalise = FALSE,
  loquace = FALSE,
  cv.k = 7
)
```

Arguments

MB	A MultiBlock object.
y	The Y-block. A class vector or dummy matrix for OPLS-DA, or a numeric matrix/vector for OPLS-R.

ndim	Number of predictive Common Dimensions. Default is 1.
nort	Maximum number of orthogonal Common Dimensions. Default is 1. The actual number used is determined by ConsensusOPLS cross-validation and may be less than this value.
method	'OPLS-DA' for discriminant analysis or 'OPLS-R' for regression.
decisionRule	Only used if method is 'OPLS-DA'. If 'fixed', samples are assigned to the class with Y-hat above 1/nclasses. If 'max', samples are assigned to the class with the highest Y-hat.
normalise	To apply block normalisation. FALSE == no (default), TRUE == yes.
loquace	To display the calculation times. TRUE == yes, FALSE == no (default).
cv.k	Number of folds for k-fold cross-validation (default 7). Set to 0 to skip CV output. ConsensusOPLS always performs internal CV to select the optimal number of orthogonal components; when cv.k >= 2 the resulting Q2 and DQ2 reflect that cross-validation.

Details

This function is a wrapper around ConsensusOPLS. The core kernel-OPLS extraction is delegated to that package; all ComDim output slots (local scores, loadings, VIP, sensitivity, confusion matrix, etc.) are computed from the returned model objects.

Value

A ComDim object. All slots are populated. Key slots:

Method "OPLS-DA" or "OPLS-R".

ndim Number of predictive Common Dimensions.

Q.scores Predictive global scores matrix ($n \times ndim$).

T.scores Named list of block-specific predictive local scores.

P.loadings Global predictive loadings.

Saliences Predictive block salience matrix ($ntable \times ndim$).

Orthogonal List with orthogonal component outputs: nort, Q.scores, T.scores, P.loadings.ort, Saliences.ort.

R2X Named vector (length $ndim + nort$) of X-variance fractions.

R2Y Named vector (length $ndim + nort$) of Y-variance fractions.

Q2 Cross-validated Q2 per class/response (when cv.k >= 2; otherwise training-set fit).

DQ2 (OPLS-DA only) Cross-validated discriminant Q2 per class.

VIP Global total VIP (named vector, length p_{tot}).

VIP.block Named list (one data.frame per block) with columns p, o, tot.

PLS.model KOPLS regression objects: W, B, B0, Y.

cv Cross-validation results when cv.k >= 2: k, Ypred, Q2, DQ2.

Prediction Training-set predictions: Y.pred; for OPLS-DA also decisionRule, trueClass, predClass, Sensitivity, Specificity, confusionMatrix.

`Mean` List with `MeanMB` and `MeanY`.

`Norm` List with `NormMB`, `FrobNorms`, `RVweights`.

`variable.block` Block membership of each variable.

`runtime` Total computation time in seconds.

References

Boccard J, Rutledge DN (2013). A consensus OPLS-DA strategy for multiblock Omics data fusion. *Analytica Chimica Acta*, 769, 30–39. doi:10.1016/j.aca.2013.01.022

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
y <- rep(c("A", "B"), 5)
results <- ComDim_OPLS(mb, y, ndim = 1, nort = 1, method = "OPLS-DA")
```

`ComDim_PCA` *ComDim_PCA*

Description

Finding common dimensions in multi-block datasets.

Usage

```
ComDim_PCA(
  MB = MB,
  ndim = NULL,
  normalise = FALSE,
  threshold = 1e-10,
  loquace = FALSE,
  CompMethod = "Normal",
  Partitions = 1
)
```

Arguments

`MB`	A MultiBlock object.
`ndim`	Number of Common Dimensions.
`normalise`	To apply normalisation. FALSE == no (default), TRUE == yes.
`threshold`	The threshold limit to stop the iterations. If the "difference of fit" < threshold (1e-10 as default).
`loquace`	To display the calculation times. TRUE == yes, FALSE == no (default).

`CompMethod` To speed-up the analysis for really big MultiBlocks. 'Normal' (default), 'Kernel', 'PCT', 'Tall' or 'Wide'.

`Partitions` To speed-up the analysis for really big MultiBlocks. This parameter is used if CompMethod is 'Tall' or 'Wide'.

Value

A `ComDim` object. Slots for supervised analysis (`R2Y`, `Q2`, `DQ2`, `VIP`, `VIP.block`, `PLS.model`, `cv`, `Prediction`) are empty. The populated slots are:

`Method` `"PCA"`.

`ndim` Number of Common Dimensions extracted.

`Q.scores` Global scores matrix ($n \times ndim$). Column names are `CC1`, `CC2`, etc.; row names are sample names. Each column $\mathbf{q}_a$ is a unit-norm consensus score, the dominant left singular vector of the salience-weighted concatenated blocks $\mathbf{W} = [\sqrt{\lambda_1}\mathbf{X}_1 \mid \cdots \mid \sqrt{\lambda_B}\mathbf{X}_B]$.

`T.scores` Named list of block-specific local scores matrices ($n \times ndim$ each). For block b and component a: local loading $\mathbf{p}_{ba} = \mathbf{X}_b'\mathbf{q}_a$ and local score $\mathbf{t}_{ba} = \mathbf{X}_b\,\mathbf{p}_{ba}(\mathbf{p}_{ba}'\mathbf{p}_{ba})^{-1}$.

`P.loadings` Global loadings matrix ($p_{tot} \times ndim$). Column a is $\mathbf{P}_a = \mathbf{X}'\mathbf{q}_a$, where $\mathbf{X}$ is the mean-centred (and optionally normalised) concatenated blocks.

`Saliences` Block salience (weight) matrix ($ntable \times ndim$, row names = block names). Entry (b, a) is $\lambda_{ba} = \mathbf{q}_a'\mathbf{X}_b\mathbf{X}_b'\mathbf{q}_a$, the variance of block b captured by global score a.

`R2X` Proportion of multi-block inertia captured by each component (named vector, length $ndim$). Let d_a be the leading singular value of $\mathbf{W}$ for component a (stored as $Singular_a = d_a^2$); then

$$R2X_a = Singular_a^2 / \sum_k Singular_k^2 = d_a^4 / \sum_k d_k^4.$$

`Singular` Squared leading singular values of $\mathbf{W}$, one per component: $Singular_a = d_a^2$.

`Mean` List with `MeanMB`: named list of column-mean vectors per block, used for mean-centring.

`Norm` List with `NormMB`: Frobenius norms used for block normalisation (all ones when `normalise = FALSE`).

`variable.block` Character vector (length p_{tot}) indicating the block name of each row in `P.loadings`.

`runtime` Total computation time in seconds.

References

Jouan-Rimbaud Bouveresse D, Rutledge DN (2024). A synthetic review of some recent extensions of ComDim. *Journal of Chemometrics*, 38(5), e3454. doi:10.1002/cem.3454

Original MATLAB implementation: https://github.com/DNRutledge/ComDim/

Examples

```
# Example 1: two data blocks.
b1 <- matrix(rnorm(500), 10, 50) # 10 samples, 50 variables
b2 <- matrix(rnorm(800), 10, 80) # 10 samples, 80 variables
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
```

```
results <- ComDim_PCA(mb, 2)

# Example 2: two data blocks, each with different replicate number
b1 <- matrix(rnorm(500), 10, 50)
batch_b1 <- rep(1, 10)
b2 <- matrix(rnorm(2400), 30, 80)
batch_b2 <- c(rep(1, 10), rep(2, 10), rep(3, 10))
mb <- MultiBlock(
  Samples = list(
    b1 = paste0("samples_", 1:10),
    b2 = rep(paste0("samples_", 1:10), 3)
  ),
  Data = list(b1 = b1, b2 = b2),
  Batch = list(b1 = batch_b1, b2 = batch_b2),
  ignore.size = TRUE
)
rw <- SplitRW(mb)
results <- ComDim_PCA(rw, 2)
```

ComDim_PLS *ComDim_PLS*

Description

Finding common dimensions in multi-block datasets using PLS.

Usage

```
ComDim_PLS(
  MB = MB,
  y = y,
  ndim = NULL,
  method = c("PLS-DA", "PLS-R"),
  decisionRule = c("fixed", "max")[2],
  normalise = FALSE,
  scale.y = FALSE,
  threshold = 1e-10,
  loquace = FALSE,
  CompMethod = "Normal",
  Partitions = 1,
  cv.k = 7
)
```

Arguments

MB	A MultiBlock object.
y	The Y-block to use in the PLS model as dependent data. A class vector or a dummy matrix.

ndim	Number of Common Dimensions.
method	PLS-DA or PLS-R.
decisionRule	Only used if method is set to PLS-DA. If 'fixed', samples are assigned to the class with Y-hat above 1/nclasses. If 'max', samples are assigned to the class with the highest Y-hat.
normalise	To apply block normalisation. FALSE == no (default), TRUE == yes. When TRUE each block is mean-centred and then divided by its Frobenius norm so that all blocks have unit total inertia entering the ComDim loop. Has no effect on the Y-block; use scale.y for that.
scale.y	Logical (default FALSE). When TRUE and method = 'PLS-R', each column of Y is mean-centred and scaled to unit variance before the PLS fit. The stored B, B0, and Ypred are back-transformed to the original Y scale, so downstream outputs (R2Y, Q2, predictions) are always in the original units. Ignored when method = 'PLS-DA' (dummy Y should not be scaled).
threshold	The threshold limit to stop the iterations. If the "difference of fit" < threshold (1e-10 as default).
loquace	To display the calculation times. TRUE == yes, FALSE == no (default).
CompMethod	To speed up the analysis for really big multi-blocks. 'Normal' (default), 'Kernel', 'PCT', 'Tall' or 'Wide'.
Partitions	To speed up the analysis for really big multi-blocks. This parameter is used if CompMethod is 'Tall' or 'Wide'.
cv.k	Number of folds for k-fold cross-validation (default 7). Set to 0 to skip CV. When cv.k >= 2, Q2 and DQ2 in the output reflect cross-validated predictive ability; otherwise they reflect training-set fit (R2).

Value

A ComDim object. All slots are populated. Key slots:

Method "PLS-DA" or "PLS-R".

ndim Number of Common Dimensions extracted.

Q.scores Global consensus PLS scores ($n \times ndim$). Each column $\mathbf{q}_a$ (unit-norm) is the dominant left singular vector from the NIPALS PLS applied to the salience-weighted concatenated blocks.

T.scores Named list of block-specific local scores ($n \times ndim$ each). Local loading $\mathbf{p}_{ba} = \mathbf{X}_b'\mathbf{q}_a$; local score $\mathbf{t}_{ba} = \mathbf{X}_b\,\mathbf{p}_{ba}(\mathbf{p}_{ba}'\mathbf{p}_{ba})^{-1}$.

P.loadings Global loadings ($p_{tot} \times ndim$): $\mathbf{P} = \mathbf{X}'\mathbf{Q}$, where $\mathbf{X}$ is the mean-centred (and optionally normalised) concatenated blocks.

Saliences Block salience matrix ($ntable \times ndim$): $\lambda_{ba} = \mathbf{q}_a'\mathbf{X}_b\mathbf{X}_b'\mathbf{q}_a$.

R2X Proportion of X variance captured by each component (named vector, length $ndim$). Let $\mathbf{t}_a$ be the NIPALS PLS X-score vector for component a on the salience- weighted blocks; then

$$R2X_a = \|\mathbf{t}_a\|^4 \Big/ \sum_k \|\mathbf{t}_k\|^4.$$

`R2Y` Cumulative Y-variance explained (named vector, length $ndim$). $R2Y_a$ is the R^2 from an OLS regression of $\mathbf{Y}$ on the first a global scores with an intercept:

$$R2Y_a = 1 - RSS_a/TSS_Y,$$

where RSS_a is the residual SS when predicting $\mathbf{Y}$ from $[1, \mathbf{q}_1, \ldots, \mathbf{q}_a]$. **Note:** $R2Y_a$ is cumulative — it reflects the total Y-variance explained by the first a components together, not the marginal contribution of component a alone.

`Q2` Predictive Q2 per response column (PLS-R) or per class (PLS-DA), named accordingly:

$$Q2 = 1 - PRESS/TSS_Y,$$

where $PRESS = \sum_i (\hat{y}_i - y_i)^2$ and $TSS_Y = \sum_i (y_i - \bar{y})^2$. When `cv.k >= 2`, $\hat{y}_i$ are out-of-sample cross-validated predictions; otherwise training-set predictions are used (i.e. Q2 = R2Y for the full model).

`DQ2` (PLS-DA only) Discriminant Q2 per class. Only penalising residuals contribute to the sum:

$$DQ2 = 1 - PRESSD/TSS_Y,$$

where $PRESSD$ sums $\hat{y}_i^2$ for class-0 samples with $\hat{y}_i > 0$, and $(\hat{y}_i - 1)^2$ for class-1 samples with $\hat{y}_i < 1$. Same cross-validation logic as `Q2`.

`Singular` Squared L2 norm of the NIPALS PLS score vector per component ($\|\mathbf{t}_a\|^2$), used to derive `R2X`.

`VIP` Global VIP scores (named numeric vector, length p_{tot}) using the Wold formula:

$$VIP_j = \sqrt{p_{tot} \cdot \frac{\sum_a s_a \tilde{w}_{j,a}^2}{\sum_a s_a}},$$

where $s_a = \|\mathbf{t}_a\|^2 \|\mathbf{q}_a\|^2$, $\tilde{w}_{j,a} = w_{j,a}/\|\mathbf{w}_a\|$ is the L2-normalised j-th element of the a-th NIPALS weight vector, and p_{tot} is the total number of variables.

`VIP.block` Named list (one `data.frame` per block) with columns `p` (per-block predictive VIP computed with block size p_b instead of p_{tot}) and `tot` (= `p` for PLS; included for consistency with OPLS output). Row names are variable names.

`PLS.model` List with: `W` (NIPALS X weight matrix, $p_{tot} \times ndim$); `B` (regression coefficients, $p_{tot} \times ncol(Y)$, $\mathbf{B} = \mathbf{W}(\mathbf{P}'\mathbf{W})^{-1}\mathbf{Q}'$, in original Y units); `B0` (intercept vector, length $ncol(Y)$, $\mathbf{B}_0 = \bar{\mathbf{y}} - \bar{\mathbf{x}}\mathbf{B}$); `Y` (original response matrix as supplied). Training-set Y predictions: $\hat{\mathbf{Y}} = \mathbf{XB} + \mathbf{B}_0$.

`cv` Cross-validation results when `cv.k >= 2` (empty list otherwise): `k` (number of folds), `fold` (sample-to-fold vector), `Ypred` ($n \times ncol(Y)$ matrix of out-of-sample predictions), `Q2` (CV Q2 per class/response), `DQ2` (mean CV DQ2 across classes, PLS-DA only), `DQ2.perclass` (CV DQ2 per class, PLS-DA only).

`Prediction` Training-set predictions: `Y.pred` ($n \times ncol(Y)$); for PLS-DA also `decisionRule`, `trueClass` (character vector), `predClass` (data.frame), `Sensitivity` and `Specificity` (named per class), `confusionMatrix` (named list of 2x2 matrices, one per class).

`Mean` List with `MeanMB` (column means per block), `MeanY` (column means of Y before any scaling), and `ScaleY` (column SDs of Y; all ones when `scale.y = FALSE`).

`Norm` List with `NormMB`: Frobenius norms for block normalisation.

`variable.block` Character vector (length p_{tot}) mapping each row of `P.loadings` and each element of `VIP` to its block.

`runtime` Total computation time in seconds.

References

Jouan-Rimbaud Bouveresse D, Rutledge DN (2024). A synthetic review of some recent extensions of ComDim. *Journal of Chemometrics*, 38(5), e3454. doi:10.1002/cem.3454

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
batch_b1 <- rep(1, 10)
b2 <- matrix(rnorm(2400), 30, 80)
batch_b2 <- c(rep(1, 10), rep(2, 10), rep(3, 10))
mb <- MultiBlock(
  Samples = list(
    b1 = paste0("samples_", 1:10),
    b2 = rep(paste0("samples_", 1:10), 3)
  ),
  Data = list(b1 = b1, b2 = b2),
  Batch = list(b1 = batch_b1, b2 = batch_b2),
  ignore.size = TRUE
)
rw <- SplitRW(mb)
y <- scale(1:10, center = TRUE)
results.plsr <- ComDim_PLS(rw, y, 2, method = "PLS-R")
groups <- c(rep("A", 5), rep("B", 5))
results.plsda <- ComDim_PLS(rw, y = groups, 2, method = "PLS-DA")
```

`ComDim_y` *ComDim_y - Extending PLS-like supervised methods to multi-block data.*

Description

Extends any PLS-like method used for regression or discriminant purposes to the multi-block field. The user provides a function (`FUN`) that computes one predictive component from the salience-weighted concatenated blocks; global scores, local scores, and loadings are then derived following the traditional ComDim-PLS framework. Optionally, orthogonal components returned by FUN (e.g. from an O-PLS wrapper) are captured. VIP scores and k-fold cross-validation are also supported.

Usage

```
ComDim_y(
  MB = MB,
  y = y,
  ndim = NULL,
  FUN = FUN,
  nort = 0L,
  type = c("regression", "discriminant")[1],
  decisionRule = c("fixed", "max")[2],
  normalise = FALSE,
```

```
  scale.y = FALSE,
  threshold = 1e-10,
  loquace = FALSE,
  method = "FUN",
  cv.k = 7,
  ...
)
```

Arguments

MB	A MultiBlock object.
y	The response: a numeric vector or matrix for regression (`type = 'regression'`), or a class vector or dummy matrix for discriminant analysis (`type = 'discriminant'`).
ndim	Number of predictive Common Dimensions. If `NULL`, defaults to the number of blocks.
FUN	The function used as the core of the ComDim analysis. It must accept `W` (salience-weighted concatenated blocks, n x p matrix), `y` (response), and `ndim` (number of components to compute) as its first three named arguments, and return a named list with at least: **scores** X scores vector (length n) for the current component. **P** X loadings vector (length p) for the current component. **W** X weights vector (length p) for the current component. **Q** Y loadings vector for the current component. **U** Y scores vector (length n) for the current component. Optional return fields: **y** y as used internally by FUN (e.g. after centring/scaling), so that `ComDim_y` can detect automatic y-transformation and back-transform the Q loadings for correct B and B0 computation. **orthoscores** A numeric vector (length n) or single-column matrix of orthogonal X-scores. **Required** when `nort = 1`. Only the first column is used. FUN does not need to change its behaviour between ort and predictive phases; if it always returns `orthoscores`, the framework uses it only during the ort phase and ignores it during the predictive phase.
nort	Number of orthogonal Common Dimensions to extract before the predictive loop. Default `0` (no orthogonalization, pure PLS-like). Only `nort = 0` or `nort = 1` are accepted; for multiple orthogonal components use `ComDim_OPLS()`. When `nort = 1` the function follows the same two-phase architecture as `ComDim_OPLS`: the orthogonal component is removed from the (lambda-weighted) concatenated blocks first, then predictive components are computed from the orthogonally-deflated blocks. `FUN` must return `output$orthoscores` (a numeric vector of length n, or a matrix whose first column is used) when called with `ndim = 1`. Cross-validation is automatically skipped when `nort > 0`.
type	`'regression'` (default) or `'discriminant'`.
decisionRule	Only used when `type = 'discriminant'`. If `'fixed'`, samples are assigned to the class whose predicted score exceeds `1/nclasses`; if `'max'`, the class with the highest predicted score is chosen. Default `'max'`.

normalise	To apply block normalisation. FALSE == no (default), TRUE == yes.
scale.y	Logical (default FALSE). When TRUE and type = 'regression', each column of Y is mean-centred and scaled to unit variance before being passed to FUN. The stored B, B0, and Ypred are back-transformed to the original Y scale, so downstream outputs (R2Y, Q2, predictions) are always in the original units. Ignored when type = 'discriminant' (dummy Y should not be scaled).
threshold	Convergence threshold: iterations stop when the change in the global score vector falls below this value (default 1e-10).
loquace	Display computation time at each step. TRUE == yes, FALSE == no (default).
method	A string label identifying the method (default: 'FUN').
cv.k	Number of folds for k-fold cross-validation (default 7). Set to 0 to skip CV. When cv.k >= 2, the Q2 and DQ2 slots in the output reflect cross-validated predictive ability; otherwise they reflect training-set fit. CV is skipped when nort > 0.
...	Additional arguments passed to FUN.

Value

A ComDim object with the following slots:

Method The label supplied via the method argument.

ndim Number of predictive Common Dimensions extracted.

Q.scores Global consensus scores matrix ($n \times ndim$). Each column $\mathbf{q}_a$ (unit-norm) is derived from the dominant left direction of FUN applied to the salience-weighted concatenated blocks.

T.scores Named list of block-specific local scores ($n \times ndim$ each). Local loading $\mathbf{p}_{ba} = \tilde{\mathbf{X}}_b' \mathbf{q}_a$ (computed on the ort-deflated block when nort > 0); local score $\mathbf{t}_{ba} = \tilde{\mathbf{X}}_b \, \mathbf{p}_{ba} (\mathbf{p}_{ba}' \mathbf{p}_{ba})^{-1}$.

P.loadings Global loadings ($p_{tot} \times ndim$): $\mathbf{P} = \tilde{\mathbf{X}}' \mathbf{Q}$, where $\tilde{\mathbf{X}}$ is the (optionally ort-deflated) mean-centred concatenated blocks.

Saliences Block salience matrix ($ntable \times ndim$): $\lambda_{ba} = \mathbf{q}_a' \tilde{\mathbf{X}}_b \tilde{\mathbf{X}}_b' \mathbf{q}_a$.

R2X Proportion of X variance captured by each predictive component (named vector, length $ndim$). Let $\mathbf{t}_a$ be the X-score vector returned by FUN for component a:

$$R2X_a = \|\mathbf{t}_a\|^4 \Big/ \sum_k \|\mathbf{t}_k\|^4.$$

When nort > 0, the denominator also includes the orthogonal $\|\mathbf{t}_{ort,k}\|^4$ terms, and the orthogonal R2X fractions are stored separately in Orthogonal$R2X.

R2Y Cumulative Y-variance explained (named vector, length $ndim$):

$$R2Y_a = 1 - RSS_a / TSS_Y,$$

where RSS_a is the residual SS from an OLS regression of $\mathbf{Y}$ on $[1, \mathbf{q}_1, \ldots, \mathbf{q}_a]$. **Note:** $R2Y_a$ is cumulative – the total Y-variance explained by the first a components together, not the marginal contribution of component a alone.

`Q2` Predictive Q2 per response column (regression) or per class (discriminant), named accordingly:

$$Q2 = 1 - PRESS/TSS_Y,$$

where $PRESS = \sum_i (\hat{y}_i - y_i)^2$. When `cv.k >= 2` and `nort = 0`: cross-validated (out-of-sample) predictions are used; otherwise training-set predictions. CV is automatically skipped when `nort > 0`.

`DQ2` (Discriminant mode only) Discriminant Q2 per class, using only penalising residuals:

$$DQ2 = 1 - PRESSD/TSS_Y,$$

where $PRESSD$ sums $\hat{y}_i^2$ for class-0 samples with $\hat{y}_i > 0$, and $(\hat{y}_i - 1)^2$ for class-1 samples with $\hat{y}_i < 1$. Same cross-validation logic as `Q2`.

`Singular` Squared L2 norm of the FUN X-score vector per component ($\|\mathbf{t}_a\|^2$), used to derive `R2X`.

`VIP` Global total VIP (named vector, length p_{tot}): concatenation of `VIP.block[[b]]$tot` across blocks. When `nort = 0`, uses the Wold formula; when `nort = 1`, tot combines predictive and orthogonal VIPs (see `VIP.block`).

`VIP.block` Named list (one `data.frame` per block). When `nort = 0`: columns `p` and `tot` (= `p`), using the Wold formula:

$$VIPp_j = \sqrt{p_b \cdot \frac{\sum_a s_a \tilde{w}_{j,a}^2}{\sum_a s_a}},$$

where $s_a = \|\mathbf{t}_a\|^2 \|\mathbf{q}_a\|^2$ and $\tilde{w}_{j,a} = w_{j,a}/\|\mathbf{w}_a\|$ is the L2-normalised j-th element of the a-th weight vector. When `nort = 1`: columns `p` (Wold, same as above), `o` (orthogonal VIP, loadings-based: $VIPo_j = \sqrt{p_b \cdot \sum_a s_{oa} \tilde{P}_{o,j,a}^2 / \sum_a s_{oa}}$, where $s_{oa} = \|\mathbf{q}_{ort}[,a]\|^2$ and $\tilde{\mathbf{P}}_o$ is the column-L2-normalised block-slice of the ort loadings), and `tot` ($VIPtot_j = \sqrt{(VIPp_j^2 + VIPo_j^2)/2}$). Row names are variable names.

`PLS.model` List with: `W` (X weight matrix collected from FUN, $p_{tot} \times ndim$); `B` (regression coefficients, $\mathbf{B} = \mathbf{W}(\mathbf{P}'\mathbf{W})^{-1}\mathbf{Q}'$, in original Y units); `B0` (intercept, $\mathbf{B}_0 = \bar{\mathbf{y}} - \bar{\tilde{\mathbf{x}}}\mathbf{B}$); `Y` (original response matrix as supplied). Training-set predictions: $\hat{\mathbf{Y}} = \tilde{\mathbf{X}}\mathbf{B} + \mathbf{B}_0$.

`cv` Cross-validation results when `cv.k >= 2` and `nort = 0` (empty list otherwise): `k`, `fold` (sample-to-fold vector), `Ypred` ($n \times ncol(Y)$ out-of-sample predictions), `Q2` (CV Q2 per class/response), `DQ2` (mean CV DQ2, discriminant only), `DQ2.perclass` (CV DQ2 per class, discriminant only).

`Orthogonal` When `nort > 0`: list with `nort`, `Q.scores` (global ort scores, $n \times nort$, unit-norm), `T.scores` (block ort local scores, $n \times nort$ each), `P.loadings.ort` (ort loadings, $p_{tot} \times nort$), `Saliences.ort` ($ntable \times nort$), and `R2X` (orthogonal X-variance fractions, $R2X_{ort,a} = \|\mathbf{t}_{ort,a}\|^4/total$). Empty list when `nort = 0`.

`Prediction` Training-set predictions: `Y.pred` ($n \times ncol(Y)$); for discriminant analysis also `decisionRule`, `trueClass`, `predClass` (data.frame), `Sensitivity` and `Specificity` (per class), `confusionMatrix` (named list of 2x2 matrices).

`Mean` List with `MeanMB` (column means per block), `MeanY` (column means of Y), and `ScaleY` (column SDs of Y; all ones when `scale.y = FALSE`).

`Norm` List with `NormMB`: Frobenius norms for block normalisation.

`variable.block` Character vector (length p_{tot}) mapping each row of `P.loadings` and each element of `VIP` to its block.

`runtime` Total computation time in seconds.

Examples

```
b1 <- matrix(rnorm(500), 10, 50) # 10 samples, 50 variables
b2 <- matrix(rnorm(800), 10, 80) # 10 samples, 80 variables
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))

## Example 1: ComDim-PLS (regression) ------------------------------------
# Single-step NIPALS PLS wrapper (one predictive component per call).
# Note: 'tx' is used instead of 't' to avoid shadowing base::t().
fun.PLS <- function(W, y, ndim, ...) {
  output <- list()
  w <- t(W) %*% y / as.numeric(t(y) %*% y) # X weight (u = y, 1 step)
  w <- w / sqrt(sum(w^2)) # L2 normalise
  tx <- W %*% w # X score
  p <- t(W) %*% tx / as.numeric(t(tx) %*% tx) # X loading
  q <- t(y) %*% tx / as.numeric(t(tx) %*% tx) # Y loading
  u <- y %*% q / as.numeric(t(q) %*% q) # Y score
  output$scores <- as.vector(tx)
  output$P <- as.vector(p)
  output$W <- as.vector(w)
  output$Q <- as.vector(q)
  output$U <- as.vector(u)
  return(output)
}

y <- c(1, 1, 1, 1, 1, 5, 5, 5, 10, 10)
resultsPLS <- ComDim_y(mb,
  y = y, ndim = 2,
  type = "regression",
  FUN = fun.PLS,
  method = "PLS",
  cv.k = 0
)

## Example 2: ComDim-OPLS-DA (discriminant, nort = 1) ----------------------
# Thin wrapper around OPLS_NIPALS_DNR(), the package's NIPALS OPLS engine.
# All inputs (W, y, and any extra args such as 'threshold') are forwarded
# directly via '...'. Use this pattern when nort > 0; for nort = 0 the
# simpler PLS wrapper in Example 1 is sufficient (no orthoscores needed).
fun.OPLS <- function(W, y, ndim, ...) {
  res <- OPLS_NIPALS_DNR(W = W, y = y, ...)
  list(
    scores      = as.vector(res$t_pred),
    P           = as.vector(res$p),
    W           = as.vector(res$w_pred),
    Q           = as.vector(res$q),
    U           = as.vector(res$u),
    orthoscores = matrix(res$t_ort, ncol = 1)
  )
}

groups <- c(rep("A", 5), rep("B", 5))
resultsOPLS <- ComDim_y(mb,
```

```
  y = groups, ndim = 1,
  nort = 1,
  type = "discriminant",
  FUN = fun.OPLS,
  method = "OPLS-DA",
  cv.k = 0
)

## Example 3 (not run): ComDim-OPLS-DA via ropls ----------------------------
# Wrapping ropls::opls is also possible. Key points:
#   - Use orthoI = 1 (fixed) instead of NA so the output is predictable.
#   - Always return output$orthoscores; ComDim_y ignores it in phases
#     where ort has already been removed.
#   - Expand the single ropls Q loading to match the ncol(y_dummy) width.

if (requireNamespace("ropls", quietly = TRUE)) {
fun.OPLSDA.ropls <- function(W, y, ndim, ...) {
  output <- list()
  # Convert dummy matrix to ropls-compatible -1/+1 vector
  Y <- c(-1, 1)[apply(y, 1, function(x) match(1, x))]
  result <- tryCatch(
    ropls::opls(
      x = W, y = Y, predI = 1, orthoI = 1,
      fig.pdfC = "none", info.txtC = "none"
    ),
    error = function(e) {
      ropls::opls(
        x = W, y = Y, predI = 1, orthoI = 0,
        fig.pdfC = "none", info.txtC = "none"
      )
    }
  )
  output$scores <- result@scoreMN[, 1]
  output$P <- result@loadingMN[, 1]
  output$W <- result@weightMN[, 1]
  output$U <- result@uMN[, 1]
  # Expand the single ropls Q loading to match the 2-column dummy matrix:
  # loadings for class1 and class2 are antisymmetric in binary PLS-DA.
  output$Q <- c(-result@cMN[, 1], result@cMN[, 1])
  output$y <- result@suppLs$yModelMN # internal y (for scaling detection)
  # Orthogonal scores (used during the ort pre-loop when nort > 0)
  if (!is.null(result@orthoScoreMN) && ncol(result@orthoScoreMN) > 0) {
    output$orthoscores <- result@orthoScoreMN   # n x k matrix; col jj used for jj-th ort
  } else {
    output$orthoscores <- matrix(0, nrow = nrow(W), ncol = 1)
  }
  return(output)
}

b1_r <- matrix(rnorm(8 * 30), 8, 30)
b2_r <- matrix(rnorm(8 * 20), 8, 20)
mb_r <- MultiBlock(Data = list(b1 = b1_r, b2 = b2_r))
resultsOPLSDA <- ComDim_y(mb_r,
```

```
  y = c(rep("NI", 4), rep("OFF", 4)),
  ndim = 1,
  nort = 1,
  type = "discriminant",
  FUN = fun.OPLSDA.ropls,
  method = "OPLS-DA(ropls)",
  cv.k = 0
)
}
```

ExpandMultiBlock *ExpandMultiBlock*

Description

Splits data into several blocks, allowing variables to appear in more than one block simultaneously. Each variable is duplicated into every block to which it is assigned according to the metadata mapping table.

Usage

```
ExpandMultiBlock(data = NULL, metadata = NULL, minblock = 0, loquace = TRUE)
```

Arguments

data	A data.frame or matrix with samples in rows and variables in columns.
metadata	A 2-column data.frame describing how variables are assigned to blocks. The first column gives the block name; the second column gives the variable name, and must match the column names of data. A variable may appear in multiple rows (and therefore in multiple blocks).
minblock	Integer. Blocks with fewer than minblock variables are discarded. Use 0 (default) to keep all blocks regardless of size.
loquace	Logical. If TRUE (default), a warning is printed listing column names in data that have no matching entry in the second column of metadata.

Details

For each row in metadata that matches a variable in data, the variable's values are copied into the corresponding block column. Column names in the resulting expanded matrix are formed as <block>.<variable>. Variables with all-NA values after expansion are removed. If no matches exist between data column names and metadata, NULL is returned with a warning. If minblock filtering removes all blocks, NULL is returned.

Value

A MultiBlock object whose blocks are defined by the first column of metadata, or NULL if no valid blocks could be constructed.

See Also

MultiBlock, ProcessMultiBlock

Examples

```
data(mouse_ds)
lipidsMB <- ExpandMultiBlock(data = lipids, metadata = metadata_lipids,
  minblock = 0, loquace = FALSE)
```

---

extra *Extracellular metabolites in growth medium*

Description

Extracellular metabolites in growth medium

Usage

```
data(mouse_ds)
```

Format

An object of class `matrix` (inherits from `array`) with 12 rows and 298 columns.

Author(s)

Radic Shechter et al. (2021) Molecular Systems Biology 17:e10141 (doi:10.15252/msb.202010141)

Source

doi:10.15252/msb.202010141

Examples

```
data(mouse_ds)
# MB <- MultiBlock(Data = list(RNAseq3 = RNAseq3,
#   lipids = lipids, intra = intra, extra = extra))
```

FilterSamplesMultiBlock *FilterSamplesMultiBlock*

Description

Retain a subset of samples in a MultiBlock object.

Usage

```
FilterSamplesMultiBlock(MB, samples = sampleNames(MB))
```

Arguments

MB	A MultiBlock object.
samples	A vector of sample names to keep. Names not found in the MultiBlock are silently ignored; the order of the retained samples follows the order given here.

Value

The MultiBlock object restricted to the requested samples. The Batch and Metadata slots are also subsetted and reordered to match.

See Also

MultiBlock, AddMetadata, ProcessMultiBlock

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
rownames(b1) <- rownames(b2) <- paste0("sample_", 1:10)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
mb <- FilterSamplesMultiBlock(mb, samples = paste0("sample_", 1:5))
```

gcms *GCMS data on cell differentiation*

Description

GCMS data on cell differentiation

Usage

```
data(dataset3)
```

Format

An object of class `matrix` (inherits from `array`) with 36 rows and 15 columns.

Author(s)

Cabrero et al. (2019) Scientific data 6:256 (doi:10.1038/s4159701902027)

Source

Metabolights

Examples

```
data(dataset3)
mb_d3 <- MultiBlock(Data = list(gcms = gcms))
```

`intra` *Intracellular metabolites in growth medium*

Description

Intracellular metabolites in growth medium

Usage

```
data(mouse_ds)
```

Format

An object of class `matrix` (inherits from `array`) with 12 rows and 230 columns.

Author(s)

Radic Shechter et al. (2021) Molecular Systems Biology 17:e10141 (doi:10.15252/msb.202010141)

Source

doi:10.15252/msb.202010141

Examples

```
data(mouse_ds)
# MB <- MultiBlock(Data = list(RNAseq3 = RNAseq3,
#   lipids = lipids, intra = intra, extra = extra))
```

KEGG_table_metabolites *Metadata for the metabolites in growth medium*

Description

Metadata for the metabolites in growth medium

Usage

```
data(mouse_ds)
```

Format

An object of class `data.frame` with 687 rows and 2 columns.

Author(s)

Radic Shechter et al. (2021) Molecular Systems Biology 17:e10141 (doi:10.15252/msb.202010141)

Source

doi:10.15252/msb.202010141

Examples

```
data(mouse_ds)
#' intraMB <- ExpandMB(data = intra, metadata = KEGG_table_metabolites,
#'   minblock = 10, loquace = FALSE)
```

lcms *LCMS data on cell differentiation*

Description

LCMS data on cell differentiation

Usage

```
data(dataset3)
```

Format

An object of class `matrix` (inherits from `array`) with 36 rows and 44 columns.

Author(s)

Cabrero et al. (2019) Scientific data 6:256 (doi:10.1038/s4159701902027)

Source

Metabolights

Examples

```
data(dataset3)
#' mb_d3 <- MultiBlock(Data = list(lcms = lcms))
```

---

lipids *Lipid profiles of cell extracts*

Description

Lipid profiles of cell extracts

Usage

```
data(mouse_ds)
```

Format

An object of class `matrix` (inherits from `array`) with 12 rows and 437 columns.

Author(s)

Radic Shechter et al. (2021) Molecular Systems Biology 17:e10141 (doi:10.15252/msb.202010141)

Source

doi:10.15252/msb.202010141

Examples

```
data(mouse_ds)
# MB <- MultiBlock(Data = list(RNAseq3 = RNAseq3,
#   lipids = lipids, intra = intra, extra = extra))
```

MakeComDimLoadingsTable
MakeComDimLoadingsTable

Description

Creates a long (tidy) data frame with the local P-loadings from a ComDim model, suitable for use with **ggplot2**.

Usage

```
MakeComDimLoadingsTable(model, blocks = NULL, dim = NULL, dim.ort = NULL)
```

Arguments

`model`	The output from a ComDim analysis (a `ComDim` object).
`blocks`	The blocks from which loadings will be extracted. A vector of integers (block indices) or block names. When omitted, all blocks are included.
`dim`	Integer vector of predictive component indices to include. When omitted, all predictive components in the model are included.
`dim.ort`	Integer vector of orthogonal component indices to include. When `NULL`, all orthogonal components present in the model are included. When `0`, no orthogonal components are included.

Value

A long data frame with one row per variable–component combination, containing the following columns:

`variable.id` Variable name (factor).

`variable.id.number` Position of the variable across all blocks (factor).

`block.id` Integer index of the block.

`block.name` Name of the block (factor).

`dim` Component number.

`value` Loading value (from `P.loadings` or `Orthogonal$P.loadings.ort`).

See Also

`MakeComDimScoresTable`, `ComDim_PCA`

Examples

```
b1 <- matrix(rnorm(500), 10, 50) # 10 rows and 50 columns
b2 <- matrix(rnorm(800), 10, 80) # 10 rows and 80 columns
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
model <- ComDim_PCA(mb, ndim = 2)
tbl <- MakeComDimLoadingsTable(model)
```

MakeComDimScoresTable *MakeComDimScoresTable*

Description

Creates a long (tidy) data frame with the global and/or local scores from a ComDim model, suitable for use with **ggplot2**.

Usage

```
MakeComDimScoresTable(
  model,
  blocks = NULL,
  dim = NULL,
  dim.ort = NULL,
  include = c("Q.scores", "T.scores", "Q.scores.ort", "T.scores.ort")[1:2]
)
```

Arguments

`model`	The output from a ComDim analysis (a `ComDim` object).
`blocks`	The blocks from which local scores will be extracted. A vector of integers (block indices) or block names. When omitted, all blocks are included. Only relevant when `"T.scores"` or `"T.scores.ort"` are in `include`.
`dim`	Integer vector of predictive component indices to include. When omitted, all predictive components in the model are included.
`dim.ort`	Integer vector of orthogonal component indices to include. When `NULL`, all orthogonal components present in the model are included. When `0`, no orthogonal components are included.
`include`	Character vector selecting which score types to include. Accepted values (case-insensitive) are `"Q.scores"` (global predictive scores), `"T.scores"` (local predictive scores), `"Q.scores.ort"` (global orthogonal scores), and `"T.scores.ort"` (local orthogonal scores). Defaults to `c("Q.scores", "T.scores")`.

Value

A long data frame with one row per sample–component–score-type combination, containing the following columns:

`sample.id` Sample name (factor).

`sample.id.number` Integer position of the sample (factor).

`block.id` Block index, or `"Global"` / `"Global.ort"` for Q scores.

`block.name` Block name, or `"Global"` / `"Global.ort"` for Q scores (factor).

`dim` Component number.

`scores.type` One of `"Global"`, `"Global.ort"`, `"Local"`, or `"Local.ort"` (factor).

`scores.type.dim` Concatenation of scores type and component number, e.g. `"Q.scores1"` (factor).

`value` Score value.

See Also

`MakeComDimLoadingsTable`, `ComDim_PCA`

Examples

```
b1 <- matrix(rnorm(500), 10, 50) # 10 rows and 50 columns
b2 <- matrix(rnorm(800), 10, 80) # 10 rows and 80 columns
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
model <- ComDim_PCA(mb, ndim = 2)
tbl <- MakeComDimScoresTable(model)
```

---

metadata_lipids *Metadata for the lipid profiles of cell extracts*

Description

Metadata for the lipid profiles of cell extracts

Usage

```
data(mouse_ds)
```

Format

An object of class `data.frame` with 437 rows and 2 columns.

Author(s)

Radic Shechter et al. (2021) Molecular Systems Biology 17:e10141 (doi:10.15252/msb.202010141)

Source

doi:10.15252/msb.202010141

Examples

```
data(mouse_ds)
#' lipidsMB <- ExpandMB(data = lipids, metadata = metadata_lipids,
#'   minblock = 0, loquace = FALSE)
```

metadata_RNAseq3 *Metadata for the RNAseq data of cell extracts*

Description

Metadata for the RNAseq data of cell extracts

Usage

```
data(mouse_ds)
```

Format

An object of class `data.frame` with 16890 rows and 2 columns.

Author(s)

Radic Shechter et al. (2021) Molecular Systems Biology 17:e10141 (doi:10.15252/msb.202010141)

Source

SEQOUT

Examples

```
data(mouse_ds)
#' RNAseqMB <- ExpandMB(data = RNAseq3, metadata = metadata_RNAseq3,
#'   minblock = 500, loquace = FALSE)
```

mirnaseq *miRNAseq data on cell differentiation*

Description

miRNAseq data on cell differentiation

Usage

```
data(dataset3)
```

Format

An object of class `matrix` (inherits from `array`) with 36 rows and 469 columns.

Author(s)

Cabrero et al. (2019) Scientific data 6:256 (doi:10.1038/s4159701902027)

Source

NCBI

Examples

```
data(dataset3)
#' mb_d3 <- MultiBlock(Data = list(mirnaseq = mirnaseq))
```

`MultiAssayExperiment2MultiBlock`
MultiAssayExperiment2MultiBlock

Description

Converts a MultiAssayExperiment into a MultiBlock. Samples are first intersected across all experiments so that only common samples are retained.

Usage

```
MultiAssayExperiment2MultiBlock(se, colData_samplenames = NULL, Batch = NULL)
```

Arguments

`se`	A `MultiAssayExperiment` object.
`colData_samplenames`	Character string giving the name of the column in `colData(se)` that holds the sample names used for matching. When `NULL`, `colData` metadata is not stored in the resulting MultiBlock.
`Batch`	Character string giving the name of the column in `colData(se)` to use as the Batch variable. Requires `colData_samplenames` to be set; an error is raised if `Batch` is supplied but `colData_samplenames` is `NULL` or the column is not found in `colData`.

Details

Columns (samples) are intersected across all experiments via `intersectColumns` before conversion. Each experiment in the `MultiAssayExperiment` becomes one block in the MultiBlock (rows = samples, columns = features). Sample names are taken from the `colData` row names of `se`. Metadata from `colData` is stored only for the first block; subsequent blocks share the same sample order but carry no additional metadata. If `Batch` is specified, the corresponding column is extracted from `colData`, removed from the metadata, and stored as the `Batch` slot of the MultiBlock.

Value

A `MultiBlock` object with one block per experiment in `se`.

See Also

MultiBlock, MultiBlock2MultiAssayExperiment

Examples

```
if (requireNamespace("MultiAssayExperiment", quietly = TRUE)) {
library(MultiAssayExperiment)
mae <- MultiAssayExperiment(
  experiments = ExperimentList(
   block1 = matrix(rnorm(50), nrow = 5, dimnames = list(paste0("s", 1:5), paste0("v", 1:10))),
   block2 = matrix(rnorm(30), nrow = 5, dimnames = list(paste0("s", 1:5), paste0("w", 1:6)))
  )
)
mb <- MultiAssayExperiment2MultiBlock(mae)
}
```

MultiBlock *MultiBlock*

Description

Creates a MultiBlock object from a named list of data blocks.

Usage

```
MultiBlock(
  Samples = NULL,
  Data,
  Variables = NULL,
  Batch = NULL,
  Metadata = NULL,
  ignore.names = FALSE,
  ignore.size = FALSE
)
```

Arguments

Samples	A vector of sample names shared across all blocks (optional). When omitted, sample names are taken from the row names of each block. If no row names exist and all blocks have the same number of rows, samples are numbered as integers. Use ignore.names or ignore.size for more flexible behaviour.
Data	A named list of matrices or data.frames (one entry per block).
Variables	A named list of variable-name vectors, one per block (optional). When omitted, column names are taken from each block; if absent, variables are numbered as integers.
Batch	A named list of batch vectors, one per block (optional).

`Metadata`	A named list of metadata data.frames, one per block (optional).
`ignore.names`	If TRUE, sample names are not checked across blocks. All blocks must have the same number of rows unless `ignore.size` is also TRUE.
`ignore.size`	If TRUE (only meaningful when `ignore.names = TRUE`), blocks with different row counts are accepted. The resulting MultiBlock stores a per-block Samples list and is not directly suitable for ComDim (use `SplitRW()` first).

Value

A MultiBlock object.

References

Puig-Castellví F, Jouan-Rimbaud Bouveresse D, Mazéas L, Chapleur O, Rutledge DN (2021). Rearrangement of incomplete multi-omics datasets combined with ComDim for evaluating replicate cross-platform variability and batch influence. *Chemometrics and Intelligent Laboratory Systems*, 218, 104422. doi:10.1016/j.chemolab.2021.104422

Examples

```
b1 <- matrix(rnorm(500), 10, 50) # 10 samples, 50 variables
b2 <- matrix(rnorm(800), 10, 80) # 10 samples, 80 variables
# Minimal call: Samples and Variables are filled in automatically.
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))

# With explicit sample names (enables cross-block alignment):
rownames(b1) <- paste0("s", 1:10)
rownames(b2) <- paste0("s", 1:10)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))

# Blocks with different row counts (replicate design):
b3 <- matrix(rnorm(800), 30, 80)
batch_b3 <- c(rep(1, 10), rep(2, 10), rep(3, 10))
mb3 <- MultiBlock(
  Data = list(b3 = b3), Batch = list(b3 = batch_b3),
  ignore.names = TRUE, ignore.size = TRUE
)
```

`MultiBlock2Matrix` *MultiBlock2Matrix*

Description

Combines the blocks of a MultiBlock into a single matrix by column-binding the selected blocks. Useful for computing summary statistics across the whole MultiBlock (e.g. maximum value).

Usage

```
MultiBlock2Matrix(MB = MB, blocks = NULL, vars = NULL)
```

Arguments

MB	A MultiBlock object.
blocks	The blocks to combine. A vector of integers (block indices) or a vector of block names. When omitted, all blocks are included.
vars	The variables to keep. A list of the same length as blocks. Each element contains the indices (integer) or names (character) of the variables to retain from the corresponding block. When omitted, all variables in each block are included.

Details

Blocks are column-bound in the order given by blocks. Row names of the output matrix are the sample names of MB. Column names are the variable names; if the same variable name appears in more than one block, duplicates are disambiguated by appending .<block_name> to all occurrences of the repeated name, and a warning is issued.

Value

A numeric matrix with rows corresponding to samples and columns corresponding to the variables of the selected blocks concatenated in order.

See Also

MultiBlock, ProcessMultiBlock

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
# Combine all blocks into a matrix and compute a summary statistic:
mat <- MultiBlock2Matrix(mb)
max(mat)
# Combine only the first block, keeping variables 1-10:
mat <- MultiBlock2Matrix(mb, blocks = 1, vars = list(1:10))
```

MultiBlock2MultiAssayExperiment *MultiBlock2MultiAssayExperiment*

Description

Converts a MultiBlock into a MultiAssayExperiment. Each block becomes one experiment; sample names, batch information, and metadata are carried over into the colData of the result.

Usage

```
MultiBlock2MultiAssayExperiment(MB, MSEmetadata = NULL)
```

Arguments

MB	A MultiBlock object.
MSEmetadata	An optional list of unstructured metadata describing the overall content of the MultiAssayExperiment (stored in its metadata slot). Pass NULL (default) to omit experiment-level metadata.

Details

Each block in MB is transposed (features x samples) and stored as a named matrix in the ExperimentList. Row names are set to the variable names and column names to the sample names of the MultiBlock. The colData is constructed from MB@Samples; any Metadata and Batch information present in the MultiBlock is appended as additional columns. A sampleMap is generated mapping every sample to every experiment using the same primary and column names.

Value

A MultiAssayExperiment object with one experiment per block in MB.

See Also

MultiBlock, MultiAssayExperiment2MultiBlock

Examples

```
if (requireNamespace("MultiAssayExperiment", quietly = TRUE)) {
library(MultiAssayExperiment)
b1 <- matrix(rnorm(50), 5, 10, dimnames = list(paste0("s", 1:5), paste0("v", 1:10)))
b2 <- matrix(rnorm(30), 5,  6, dimnames = list(paste0("s", 1:5), paste0("w", 1:6)))
mb <- MultiBlock(Data = list(block1 = b1, block2 = b2))
mae <- MultiBlock2MultiAssayExperiment(mb)
mae <- MultiBlock2MultiAssayExperiment(mb, MSEmetadata = list(study = "example"))
}
```

NAInfRemoveMultiBlock *NAInfRemoveMultiBlock*

Description

Remove NA and Infinite values from a MultiBlock object in a single pass. Variables whose combined count of NA and Infinite values meets or exceeds minfrac * nrow are discarded first. Remaining Infinite values are then replaced according to inf.method; remaining NA values are imputed according to na.method.

Usage

```
NAInfRemoveMultiBlock(
  MB,
  blocks = NULL,
  minfrac = 0.5,
 na.method = c("none", "zero", "median", "discard", "fixed.value", "fixed.value.all",
    "fixed.noise", "random.noise", "QRILC")[8],
  inf.method = c("none", "fixed.noise", "random.noise")[3],
  constant = 0,
  factor.NA = 0.5,
  sd.noise = 0.3,
  tune.sigma = 1,
  showWarning = TRUE
)
```

Arguments

MB	The MultiBlock object.
blocks	Blocks to process: a vector of integers or block names (optional; all blocks are processed when omitted).
minfrac	Minimum fraction of valid (non-NA, finite) values required to retain a variable. Variables at or below this threshold are discarded. Default: 0.5.
na.method	Imputation method for remaining NA values after the minfrac filter. One of: 'none' (keep NAs), 'zero', 'median', 'discard' (drop columns still containing NAs), 'fixed.value', 'fixed.value.all', 'fixed.noise', 'random.noise', 'QRILC'. Default: 'random.noise'.
inf.method	Replacement method for remaining Infinite values after the minfrac filter. One of: 'none' (keep Inf), 'fixed.noise', 'random.noise'. Default: 'random.noise'.
constant	For na.method = 'fixed.value': value to replace NAs. For na.method = 'fixed.value.all': value added to every element (NAs are then set to this value). Default: 0.
factor.NA	Noise factor used by 'fixed.noise' and 'random.noise' for both NA and Inf imputation. For NA and -Inf: the replacement mean equals col_min * factor.NA. For +Inf: col_max / factor.NA. Default: 0.5.
sd.noise	Standard-deviation factor for 'random.noise': SD = abs(mean * sd.noise). Default: 0.3.
tune.sigma	Tuning scalar for na.method = 'QRILC'. Default: 1 (Gaussian complete-data assumption).
showWarning	If TRUE, emit a warning when variables are discarded by the minfrac filter. Default: TRUE.

Value

The processed MultiBlock object.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
b2[c(2, 3, 5), c(1, 2, 3)] <- NA
b2[c(1, 4), c(4, 5)] <- Inf
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
mb <- NAInfRemoveMultiBlock(mb, na.method = "zero", inf.method = "fixed.noise")
```

---

ncol,MultiBlock-method

ncol for MultiBlock

Description

Return the number of columns (variables) in each block of a MultiBlock.

Usage

```
## S4 method for signature 'MultiBlock'
ncol(x)
```

Arguments

x	A MultiBlock object.

Value

A named integer vector with the number of columns per block.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
ncol(mb) # c(b1 = 50, b2 = 80)
```

NormalizeMultiBlock *NormalizeMultiBlock*

Description

Normalize all blocks from a MultiBlock object. The ranknorm transform is based on that from the RNOmni package (doi:10.1111/biom.13214).

Usage

```
NormalizeMultiBlock(
  MB,
  blocks = NULL,
 method = c("none", "auto", "mean", "pareto", "norm", "geometric", "ranknorm")[5],
  infinite.as.NA = FALSE,
  constant = 0,
  offset = 3/8,
  showWarning = TRUE
)
```

Arguments

MB	The MultiBlock object.
blocks	Blocks to normalize. A vector of integers or block names (optional; all blocks are processed when omitted).
method	Normalization method. One of: 'none' (no transformation beyond optional Inf-to-NA conversion), 'auto' (auto-scaling), 'mean' (mean-centering), 'pareto' (mean-centered and divided by the square root of the SD), 'norm' (mean-centered and divided by its Frobenius norm), 'geometric' (geometric mean scaling), 'ranknorm' (rank-based inverse normal transform). Default: 'norm'.
infinite.as.NA	If TRUE, Infinite values are converted to NA before normalization. The presence of NA or Inf values will cause ComDim to stop; use NAInfRemoveMultiBlock to handle them.
constant	For 'geometric': value added to each element before computing the geometric mean (useful when data contain zeros). Default: 0.
offset	For 'ranknorm': offset in the Blom-type transform. Default: 3/8 (Blom transform).
showWarning	If TRUE, warn when all-NA variables are discarded. Default: TRUE.

Value

The normalized MultiBlock object.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
b2[c(2, 3, 5), c(1, 2, 3)] <- NA
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
mb <- NormalizeMultiBlock(mb, method = "auto")
```

nrow,MultiBlock-method *nrow for MultiBlock*

Description

Return the number of rows (samples) in each block of a MultiBlock. For a valid MultiBlock all blocks share the same number of rows, so the result is a named integer vector with one (identical) value per block.

Usage

```
## S4 method for signature 'MultiBlock'
nrow(x)
```

Arguments

x	A MultiBlock object.

Value

A named integer vector with the number of rows per block.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
nrow(mb) # c(b1 = 10, b2 = 10)
```

OPLS_NIPALS_DNR *OPLS_NIPALS_DNR*

Description

One NIPALS OPLS step on a (lambda-weighted) concatenated block matrix. Computes one predictive component and one orthogonal component in a single pass. This function is the recommended building block for constructing an OPLS-based `FUN` argument for `ComDim_y()` when `nort > 0`. For `nort = 0` (plain PLS) a simpler PLS wrapper is sufficient.

Usage

```
OPLS_NIPALS_DNR(W = W, y = y, threshold = 1e-10)
```

Arguments

W	Numeric matrix (n x p): the concatenated, lambda-weighted blocks as passed by ComDim_y().
y	Numeric matrix (n x q): the response block (dummy matrix for discriminant analysis, numeric matrix for regression). Only the first column drives the NIPALS u-score iteration; all columns are used to compute the Y-loading q.
threshold	Convergence threshold for the u-score update (default 1e-10).

Value

A named list:

t_pred Predictive X-score (length n).

w_pred Predictive X-weight (length p), L2-normalised.

p X-loading (length p).

q Y-loading (length q = ncol(y)).

u Y-score (length n).

t_ort Orthogonal X-score (length n).

w_ort Orthogonal X-weight (length p), L2-normalised.

p_ort Orthogonal X-loading (length p).

See Also

ComDim_y for the multi-block OPLS wrapper that uses this function.

PredictMultiBlock *PredictMultiBlock*

Description

Projects a new MultiBlock dataset into an existing ComDim model. Works with models produced by ComDim_PCA, ComDim_PLS, ComDim_OPLS, ComDim_y, and ComDim_Exploratory. The projection type (PCA-like, PLS-like, OPLS-like) is determined from the model structure rather than the method string, so custom method labels from ComDim_y are handled automatically.

Usage

```
PredictMultiBlock(MB = MB, y, model = model, normalise = FALSE, loquace = TRUE)
```

Arguments

MB	A MultiBlock object containing the new samples to project.
y	Response vector or dummy matrix (optional). When supplied for a supervised model, Q2, DQ2, and classification statistics are computed for the new samples.
model	A ComDim object (the calibration model).
normalise	If TRUE, each block is mean-centred using the training column means and divided by the training Frobenius norm. Must match the normalise setting used during calibration. Default FALSE.
loquace	If TRUE, print a message for each set of model elements that were projected. Default TRUE.

Value

The model ComDim object with updated slots:

- Q.scores — projected global scores (new samples x ndim).
- T.scores — projected local scores (per block).
- Orthogonal$Q.scores — projected global ort scores (if model has orthogonal components).
- Orthogonal$T.scores — projected local ort scores.
- Prediction$Y.pred — predicted Y (supervised models).
- Q2, DQ2, classification slots — when y is supplied for a supervised model.

ProcessMultiBlock *ProcessMultiBlock*

Description

Apply a custom function to transform a MultiBlock and/or select variables or blocks. When multiple operations are supplied, the order of execution is: vars subsetting, then FUN, then FUN.SelectVars, then FUN.SelectBlocks.

Usage

```
ProcessMultiBlock(
  MB = MB,
  blocks = NULL,
  vars = NULL,
  FUN = NULL,
  FUN.SelectVars = NULL,
  FUN.SelectBlocks = NULL
)
```

Arguments

MB	A MultiBlock object.
blocks	The blocks to process. A vector of integers or block names. When omitted, all blocks are processed.
vars	The variables to keep. A list with the same length as blocks. Each element contains the indices (integer) or names (character) of the variables to retain in that block. Subsetting is applied before any of the FUN arguments.
FUN	A function applied to each selected block's data matrix (samples x variables). It receives the matrix as its sole argument and must return a matrix of the same dimensions.
FUN.SelectVars	A function applied to each selected block's data matrix to determine which variables to keep. It receives the matrix as its sole argument and must return a logical vector of length equal to ncol of the matrix (TRUE = keep).
FUN.SelectBlocks	A function applied to each selected block's data matrix to determine whether the block should be retained. It receives the matrix as its sole argument and must return a single TRUE or FALSE.

Value

The processed MultiBlock object, with data matrices transformed and/or blocks/variables removed according to the supplied arguments. Blocks not listed in blocks are left unchanged.

See Also

MultiBlock, MultiBlock2Matrix

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
# Normalize each block to 0-100 range
BY100 <- function(x) 100 * (x - min(x)) / (max(x) - min(x))
mb <- ProcessMultiBlock(mb, FUN = BY100)
# Keep only variables with non-zero variance
mb <- ProcessMultiBlock(mb, FUN.SelectVars = function(x) apply(x, 2, var) > 0)
# Remove blocks where all values are below 0.5
mb <- ProcessMultiBlock(mb, FUN.SelectBlocks = function(x) max(x) >= 0.5)
```

---

rnaseq *RNAseq data on cell differentiation*

Description

RNAseq data on cell differentiation

Usage

```
data(dataset3)
```

Format

An object of class `matrix` (inherits from `array`) with 36 rows and 12762 columns.

Author(s)

Cabrero et al. (2019) Scientific data 6:256 (doi:10.1038/s4159701902027)

Source

NCBI

Examples

```
data(dataset3)
#' mb_d3 <- MultiBlock(Data = list(rnaseq = rnaseq))
```

`RNAseq3` *RNAseq data of cell extracts*

Description

RNAseq data of cell extracts

Usage

```
data(mouse_ds)
```

Format

An object of class `matrix` (inherits from `array`) with 12 rows and 9254 columns.

Author(s)

Radic Shechter et al. (2021) Molecular Systems Biology 17:e10141 (doi:10.15252/msb.202010141)

Source

SEQOUT

Examples

```
data(mouse_ds)
# MB <- MultiBlock(Data = list(RNAseq3 = RNAseq3,
#   lipids = lipids, intra = intra, extra = extra))
```

sampleNames *sampleNames*

Description

Return the sample names of a MultiBlock object.

Usage

```
sampleNames(x)

## S4 method for signature 'MultiBlock'
sampleNames(x)
```

Arguments

x	A MultiBlock object.

Value

A vector with the sample names.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
rownames(b1) <- paste0("s", 1:10)
mb <- MultiBlock(Data = list(b1 = b1))
sampleNames(mb) # "s1" ... "s10"
```

sampleNames<- *sampleNames<-*

Description

Set the sample names of a MultiBlock object.

Usage

```
sampleNames(x) <- value

## S4 replacement method for signature 'MultiBlock'
sampleNames(x) <- value
```

Arguments

x	A MultiBlock object.
value	A vector of new sample names. Must have the same length as the current number of samples.

Value

The updated MultiBlock object.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
mb <- MultiBlock(Data = list(b1 = b1))
sampleNames(mb) <- paste0("patient_", 1:10)
sampleNames(mb)
```

SelectFeaturesRW *SelectFeaturesRW*

Description

Finds the important variables presenting a coordinated response across all specified replicate-blocks for a given ComDim component.

Usage

```
SelectFeaturesRW(
  RW = RW,
  results = results,
  ndim = NULL,
  blocks = NULL,
  threshold_cor = 1,
  threshold_cov = 1,
  mean.RW = TRUE,
  plots = "NO"
)
```

Arguments

RW	The object used as input in the ComDim analysis.
results	The output object obtained in the ComDim analysis.
ndim	The number of the component for which the important variables are to be identified.
blocks	A vector with the indices or the names for the replicate blocks of the same data type.

`threshold_cor` The "times" parameter used to calculate the threshold in the following formula: cor(variable) > times * sd(cor(variables)). Minimal value that can be assigned to threshold_cor is 1.

`threshold_cov` The "times" parameter used to calculate the threshold in the following formula: cov(variable) > times * sd(cov(variables)). Minimal value that can be assigned to threshold_cov is 1.

`mean.RW` Logical value to indicate whether the RW data must be mean-centered (TRUE) or not (FALSE).

`plots` Parameter to indicate whether S-plots (covariance vs. correlation with the Q scores) must be produced. Possible values are `"NO"` for no plots, `"separated"` for displaying one S-plot per block individually (pausing between plots), and `"together"` to arrange all S-plots side by side in a single grid. In all plot variants, variables selected as important are highlighted in red and labelled with their variable name. Requires the `ggplot2` package; `"together"` additionally requires `gridExtra`.

Details

The function applies an S-plot approach to identify variables that are both strongly covarying and strongly correlated with the Q scores of the chosen component. For each block in `blocks`, covariance (`s1`) and correlation (`s2`) of every variable with the (pseudo-inverse-scaled) Q scores are computed. A variable is considered important in a block if its absolute covariance exceeds `threshold_cov * sd(s1)` *and* its absolute correlation exceeds `threshold_cor * sd(s2)`. Only variables that satisfy both criteria in *all* specified blocks simultaneously are returned. The sign of the local P-loadings is used to separate variables into positive and negative groups.

Value

A named list with two elements:

`$positive` Integer indices (named by variable name) of the important variables presenting a positive relationship with the Q scores (positive covariance, positive correlation, and positive local P-loading) across all specified blocks.

`$negative` Integer indices (named by variable name) of the important variables presenting a negative relationship with the Q scores (negative covariance, negative correlation, and negative local P-loading) across all specified blocks.

When `plots` is not `"NO"`, S-plots are also displayed as a side effect: each plot shows covariance on the x-axis and correlation on the y-axis, with selected variables highlighted in red.

See Also

SplitRW, ComDim_PCA

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
batch_b1 <- rep(1, 10)
b2 <- matrix(rnorm(800), 30, 80)
```

```
batch_b2 <- c(rep(1, 10), rep(2, 10), rep(3, 10))
mb <- MultiBlock(
  Data = list(b1 = b1, b2 = b2),
  Batch = list(b1 = batch_b1, b2 = batch_b2),
  ignore.names = TRUE, ignore.size = TRUE
)
rw <- SplitRW(mb)
results <- ComDim_PCA(rw, 2)
# Identify important variables for component 1 across replicate blocks 2, 3, and 4
features <- SelectFeaturesRW(RW = rw, results = results, ndim = 1, blocks = c(2, 3, 4))
# Use stricter thresholds and display S-plots side by side
features <- SelectFeaturesRW(RW = rw, results = results, ndim = 1, blocks = c(2, 3, 4),
                             threshold_cor = 2, threshold_cov = 2, plots = "together")
```

`SimulateMultiBlock` *SimulateMultiBlock*

Description

Generate a synthetic `MultiBlock` dataset built from a known number of orthogonal latent sources plus Gaussian noise. Useful for benchmarking and testing ComDim functions.

Usage

```
SimulateMultiBlock(
  n = 500L,
  p = 2000L,
  n_sources = 4L,
  noise = 0.05,
  n_blocks = 2L
)
```

Arguments

`n`	Number of samples. Default: `500`.
`p`	Total number of variables (split evenly across blocks). Must be divisible by `n_blocks` and by `n_sources`. Default: `2000`.
`n_sources`	Number of orthogonal latent sources. Default: `4`.
`noise`	Fraction of total variance attributed to noise, in (0, 1). Default: `0.05` (5 % noise).
`n_blocks`	Number of blocks to split the variables into. Default: `2`.

Details

The dataset is constructed as follows:

1. `n_sources` score vectors ($n \times$ n_sources) are drawn from a standard normal distribution and orthonormalised by QR decomposition.

2. Loading vectors (n_sources $\times$ p) are built so that each source loads primarily (SD = 1) on one equal-sized variable segment, with small cross-loadings (SD = 0.10) on the remaining variables.
3. The true signal $X = TP$ is computed.
4. Gaussian noise is added such that noise_var $=$ signal_var $\times$ noise$/(1 -$ noise$)$.
5. The p variables are split into `n_blocks` equal-width blocks, each assembled as a named element of the returned `MultiBlock`.

Value

A `MultiBlock` object with `n_blocks` blocks, each of size $n \times (p/\text{n\_blocks})$, named `"Block1"`, `"Block2"`, etc.

See Also

`ComDim_PCA`, `MultiBlock`

Examples

```
mb <- SimulateMultiBlock(n = 100, p = 200, n_sources = 4,
                         noise = 0.05, n_blocks = 2)
mb <- NormalizeMultiBlock(mb, method = 'norm')
res <- ComDim_PCA(mb, ndim = 4)
```

SplitRW *SplitRW*

Description

Splits a multi-block into a replicate-wise (RW) structure by expanding each block along its batch dimension. Each batch within each original block becomes a separate block in the output, enabling replicate-wise ComDim analysis.

Usage

```
SplitRW(
  MB = MB,
  checkSampleCorrespondence = FALSE,
  batchNormalisation = TRUE,
  showSampleCorrespondence = TRUE
)
```

Arguments

MB	A MultiBlock object built with MultiBlock(). Must contain Batch information; blocks without batch labels are treated as a single batch.
checkSampleCorrespondence	Logical. If FALSE (default), identical sample order and count are assumed across all batches and only the minimum batch size is used. If TRUE, sample names are intersected across all batches and only common samples are retained, which is safer when batches have different sizes or orderings.
batchNormalisation	Logical. If TRUE (default), each replicate block is divided by its Frobenius norm and by the square root of the number of replicates in its original block, to prevent blocks with more replicates from dominating the analysis.
showSampleCorrespondence	Logical. If TRUE (default), the matrix of sample names assigned to each replicate block is printed to the console via print(). Set to FALSE to suppress this output.

Details

Output block names follow the convention <original_block> when the original block has only one batch, or <original_block>_<batch_label> when it has multiple batches. The Metadata slot of each source block is also split and carried over to the corresponding replicate blocks. If the MultiBlock has no Batch information at all, the original object is returned unchanged with a warning.

Value

A MultiBlock object in which each block corresponds to one batch of one original data block (a replicate-wise structure ready for ComDim_PCA or similar).

See Also

MultiBlock, ComDim_PCA, SelectFeaturesRW

Examples

```
b1 <- matrix(rnorm(1500), 30, 50)
b2 <- matrix(rnorm(2400), 30, 80)
batch_b <- c(rep(1, 10), rep(2, 10), rep(3, 10))
# Generate the multi-block (mb) with 3 batches of 10 samples each
mb <- MultiBlock(
  Data = list(b1 = b1, b2 = b2),
  Batch = list(b1 = batch_b, b2 = batch_b),
  ignore.names = TRUE
)
rw <- SplitRW(mb)
```

SummarizedExperiment2MultiBlock
SummarizedExperiment2MultiBlock

Description

Converts a SummarizedExperiment into a MultiBlock. Each assay in the SummarizedExperiment becomes one block (rows = samples, columns = features).

Usage

```
SummarizedExperiment2MultiBlock(se, colData_samplenames = NULL, Batch = NULL)
```

Arguments

se	A SummarizedExperiment object.
colData_samplenames	Character string giving the name of the column in colData(se) that holds the sample names used for matching. When NULL, colData metadata is not stored in the resulting MultiBlock.
Batch	Character string giving the name of the column in colData(se) to use as the Batch variable. When supplied, the column is extracted from colData, removed from the metadata, and stored in the Batch slot of the MultiBlock. Requires at least one of colData_samplenames or a named colData; an error is raised if the column is not found.

Details

Each assay is transposed so that samples are in rows and features in columns. Sample order is aligned to colData(se) row names; samples not present in colData are removed. If an assay has no name, the block is labelled "X". Metadata from colData (excluding the Batch column if specified) is stored only for the first block; subsequent assays are appended as additional blocks with no extra metadata.

Value

A MultiBlock object with one block per assay in se.

See Also

MultiBlock, MultiAssayExperiment2MultiBlock, MultiBlock2MultiAssayExperiment

Examples

```
if (requireNamespace("SummarizedExperiment", quietly = TRUE)) {
library(SummarizedExperiment)
nrows <- 20; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
rownames(counts) <- paste0("feature", seq_len(nrows))
colnames(counts) <- paste0("sample",  seq_len(ncols))
se <- SummarizedExperiment(assays = list(counts = counts))
mb <- SummarizedExperiment2MultiBlock(se)
}
```

`variableNames` *variableNames*

Description

Return the variable names of a MultiBlock object.

Usage

```
variableNames(x, ...)

## S4 method for signature 'MultiBlock'
variableNames(x, block)
```

Arguments

`x`	A MultiBlock object.
`...`	Not used. Present for S4 generic dispatch compatibility.
`block`	Optional. A vector of block names or indices to retrieve. When omitted, variable names for all blocks are returned as a named list. When a single block is specified, a plain vector is returned.

Value

A named list of variable-name vectors (all blocks), or a single vector when exactly one block is requested.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))
variableNames(mb) # named list: b1 = 1:50, b2 = 1:80
variableNames(mb, "b1") # 1:50
variableNames(mb, 1:2) # same as variableNames(mb)
```

variableNames<- *variableNames<-*

Description

Set the variable names of a MultiBlock object. `value` must be a named list with one entry per block, where each entry is a vector of variable names whose length matches the number of columns in that block.

Usage

```
variableNames(x) <- value

## S4 replacement method for signature 'MultiBlock'
variableNames(x) <- value
```

Arguments

`x`	A MultiBlock object.
`value`	A named list of variable-name vectors, one per block.

Details

To update a single block, use standard list-replacement chaining: `variableNames(mb)[["b1"]] <- newNames`

Value

The updated MultiBlock object.

Examples

```
b1 <- matrix(rnorm(500), 10, 50)
b2 <- matrix(rnorm(800), 10, 80)
mb <- MultiBlock(Data = list(b1 = b1, b2 = b2))

# Replace all at once:
variableNames(mb) <- list(
  b1 = paste0("v", 1:50),
  b2 = paste0("v", 1:80)
)

# Replace a single block using chaining:
variableNames(mb)[["b1"]] <- paste0("feat_", 1:50)
```

Index