Package 'multivarious'

January 22, 2026

Title Extensible Data Structures for Multivariate Analysis

Version 0.3.1

Description Provides a set of basic and extensible data structures and functions for multivariate analysis, including dimensionality reduction techniques, projection methods, and preprocessing functions. The aim of this package is to offer a flexible and user-friendly framework for multivariate analysis that can be easily extended for custom requirements and specific data analysis tasks.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports rlang, chk, glmnet, corpcor, Matrix, rsvd, svd, pls, irlba, RSpectra, proxy, matrixStats, ggplot2, ggrepel, future.apply, tibble, dplyr, crayon, MASS, methods, cli, withr, assertthat, future, geigen, PRIMME, GPArotation, lifecycle

Suggests covr, randomForest, testthat, magrittr, knitr, rmarkdown

URL https://bbuchsbaum.github.io/multivarious/

VignetteBuilder knitr

Config/Needs/website albersdown

NeedsCompilation no

Author Bradley Buchsbaum [aut, cre] (ORCID: <https://orcid.org/0000-0002-1108-4866>)

Maintainer Bradley Buchsbaum <brad.buchsbaum@gmail.com>

Depends R (>= 4.2.0)

Repository CRAN

Date/Publication 2026-01-21 23:00:03 UTC

Contents

---

add_node *add a pre-processing stage*

Description

add a pre-processing stage

Usage

```
add_node(x, step, ...)
```

Arguments

x	the processing pipeline
step	the pre-processing step to add
...	extra args

Value

a new pre-processing pipeline with the added step

`add_node.prepper` *Add a pre-processing node to a pipeline*

Description

Add a pre-processing node to a pipeline

Usage

```
## S3 method for class 'prepper'
add_node(x, step, ...)
```

Arguments

`x`	A `prepper` pipeline
`step`	The pre-processing step to add
`...`	Additional arguments

`apply_rotation` *Apply rotation*

Description

Apply a specified rotation to the fitted model

Usage

```
apply_rotation(x, rotation_matrix, ...)
```

Arguments

`x`	A model object, possibly created using the `pca()` function.
`rotation_matrix`	`matrix` reprsenting the rotation.
`...`	extra args

Value

A modified object with updated components and scores after applying the specified rotation.

apply_transform *apply a pre-processing transform*

Description

apply a pre-processing transform

Usage

```
apply_transform(x, X, colind, ...)
```

Arguments

x	the pre_processor
X	the data matrix
colind	column indices
...	extra args

Value

the transformed data

biplot.pca *Biplot for PCA Objects (Enhanced with ggrepel)*

Description

Creates a 2D biplot for a pca object, using **ggplot2** and **ggrepel** to show both sample scores (observations) and variable loadings (arrows).

Usage

```
## S3 method for class 'pca'
biplot(
  x,
  y = NULL,
  dims = c(1, 2),
  scale_arrows = 2,
  alpha_points = 0.6,
  point_size = 2,
  point_labels = NULL,
  var_labels = NULL,
  arrow_color = "red",
  text_color = "red",
  repel_points = TRUE,
```

```
  repel_vars = FALSE,
  ...
)
```

Arguments

x	A pca object returned by pca.
y	(ignored) Placeholder to match biplot(x, y, ...) signature.
dims	A length-2 integer vector specifying which principal components to plot on the x and y axes. Defaults to c(1, 2).
scale_arrows	A numeric factor to scale the variable loadings (arrows). Default is 2.
alpha_points	Transparency level for the sample points. Default is 0.6.
point_size	Size for the sample points. Default is 2.
point_labels	Optional character vector of labels for the sample points. If NULL, rownames of the scores matrix are used if available; otherwise numeric indices.
var_labels	Optional character vector of variable names (columns in the original data). If NULL, rownames of x\$v are used if available; otherwise "Var1", "Var2", etc.
arrow_color	Color for the loading arrows. Default is "red".
text_color	Color for the variable label text. Default is "red".
repel_points	Logical; if TRUE, repel sample labels using geom_text_repel. Default is TRUE.
repel_vars	Logical; if TRUE, repel variable labels using geom_text_repel. Default is FALSE.
...	Additional arguments passed on to ggplot2 or ggrepel functions (if needed).

Details

This function constructs a scatterplot of the PCA scores (observations) on two chosen components and overlays arrows for the loadings (variables). The arrow length and direction indicate how each variable contributes to those principal components. You can control arrow scaling with scale_arrows.

If your pca object includes an $explained_variance field (e.g., proportion of variance per component), those values will appear in the axis labels. Otherwise, the axes are labeled simply as "PC1", "PC2", etc.

Note: If you do not have **ggrepel** installed, you can set repel_points=FALSE and repel_vars=FALSE, or install **ggrepel**.

Value

A ggplot object.

Examples

```
data(iris)
X <- as.matrix(iris[,1:4])
pca_res <- pca(X, ncomp=2)

# Enhanced biplot with repelled text
biplot(pca_res, repel_points=TRUE, repel_vars=TRUE)
```

`bi_projector` *Construct a bi_projector instance*

Description

A bi_projector offers a two-way mapping from samples (rows) to scores and from variables (columns) to components. Thus, one can project from D-dimensional input space to d-dimensional subspace. And one can project (project_vars) from n-dimensional variable space to the d-dimensional component space. The singular value decomposition is a canonical example of such a two-way mapping.

Usage

```
bi_projector(v, s, sdev, preproc = prep(pass()), classes = NULL, ...)
```

Arguments

`v`	A matrix of coefficients with dimensions `nrow(v)` by `ncol(v)` (columns = components)
`s`	The score matrix
`sdev`	The standard deviations of the score matrix
`preproc`	(optional) A pre-processing pipeline, default is prep(pass())
`classes`	(optional) A character vector specifying the class attributes of the object, default is NULL
`...`	Extra arguments to be stored in the `projector` object.

Value

A bi_projector object

Examples

```
X <- matrix(rnorm(200), 10, 20)
svdfit <- svd(X)

p <- bi_projector(svdfit$v, s = svdfit$u %*% diag(svdfit$d), sdev=svdfit$d)
```

bi_projector_union *A Union of Concatenated* bi_projector *Fits*

Description

This function combines a set of bi_projector fits into a single bi_projector instance. The new instance's weights and associated scores are obtained by concatenating the weights and scores of the input fits.

Usage

```
bi_projector_union(fits, outer_block_indices = NULL)
```

Arguments

fits	A list of bi_projector instances with the same row space. These instances will be combined to create a new bi_projector instance.
outer_block_indices	An optional list of indices for the outer blocks. If not provided, the function will compute the indices based on the dimensions of the input fits.

Value

A new bi_projector instance with concatenated weights, scores, and other properties from the input bi_projector instances.

Examples

```
X1 <- matrix(rnorm(5*5), 5, 5)
X2 <- matrix(rnorm(5*5), 5, 5)

bpu <- bi_projector_union(list(pca(X1), pca(X2)))
```

block_indices *get block_indices*

Description

extract the list of indices associated with each block in a multiblock object

Usage

```
block_indices(x, ...)
```

Arguments

x	the object
...	extra args

Value

a list of block indices

---

block_indices.multiblock_projector *Extract the Block Indices from a Multiblock Projector*

Description

Extract the Block Indices from a Multiblock Projector

Usage

```
## S3 method for class 'multiblock_projector'
block_indices(x, i, ...)
```

Arguments

x	A multiblock_projector object.
i	Ignored.
...	Ignored.

Value

The list of block indices.

---

block_lengths *get block_lengths*

Description

extract the lengths of each block in a multiblock object

Usage

```
block_lengths(x)
```

Arguments

x	the object

Value

the block lengths

`bootstrap` *Bootstrap Resampling for Multivariate Models*

Description

Perform bootstrap resampling on a multivariate model to estimate the variability of components and scores.

Usage

```
bootstrap(x, nboot, ...)

## S3 method for class 'plsc'
bootstrap(x, nboot = 500, ...)
```

Arguments

`x`	A fitted model object, such as a `projector`, that has been fit to a training dataset.
`nboot`	An integer specifying the number of bootstrap resamples to perform.
`...`	Additional arguments to be passed to the specific model implementation of `bootstrap`.

Value

A list containing the bootstrap resampled components and scores for the model.

`bootstrap_pca` *Fast, Exact Bootstrap for PCA Results from* `pca` *function*

Description

Performs bootstrap resampling for Principal Component Analysis (PCA) based on the method described by Fisher et al. (2016), optimized for high-dimensional data (p » n). This version is specifically adapted to work with the output object generated by the provided `pca` function (which returns a `bi_projector` object of class 'pca').

Usage

```
bootstrap_pca(
  x,
  nboot = 100,
  k = NULL,
  parallel = FALSE,
  cores = NULL,
  seed = NULL,
  epsilon = 1e-15,
  ...
)
```

Arguments

x	An object of class 'pca' as returned by the provided pca function. It's expected to contain loadings (v), scores (s), singular values (sdev), left singular vectors (u), and pre-processing info (preproc).
nboot	The number of bootstrap resamples to perform. Must be a positive integer (default: 100).
k	The number of principal components to bootstrap (default: all components available in the fitted PCA model x). Must be less than or equal to the number of components in x.
parallel	Logical flag indicating whether to use parallel processing via the future framework (default: FALSE). Requires the future.apply package and a configured future backend (e.g., future::plan(future::multisession)).
cores	The number of cores to use for parallel processing if parallel = TRUE (default: future::availableCores()). This is used if no future plan is set.
seed	An integer value for the random number generator seed for reproducibility (default: NULL, no seed is set).
epsilon	A small positive value added to standard deviations before division to prevent division by zero or instability (default: 1e-15).
...	Additional arguments (currently ignored).

Details

This function implements the fast bootstrap PCA algorithm proposed by Fisher et al. (2016), adapted for the output structure of the provided pca function. The pca function returns an object containing:

- v: Loadings (coefficients, p x k) - equivalent to V in SVD Y = U D V'. Note the transpose difference from prcomp.
- s: Scores (n x k) - calculated as U %*% D.
- sdev: Singular values (vector of length k) - equivalent to d.
- u: Left singular vectors (n x k).

The bootstrap algorithm works by resampling the *subjects* (rows) and recomputing the SVD on a low-dimensional representation. Specifically, it computes the SVD of the resampled matrix `D U' P^b`, where `Y = U D V'` is the SVD of the original (pre-processed) data, and `P^b` is a resampling matrix operating on the subjects (columns of U').

The SVD of the resampled low-dimensional matrix is `svd(D U' P^b) = A^b S^b (R^b)'`. The bootstrap principal components (loadings) are then calculated as `V^b = V A^b`, and the bootstrap scores are `Scores^b = R^b S^b`.

Z-scores are provided as `mean / sd`.

Important Note: The algorithm assumes the data `Y` used for the *original* SVD (`Y = U D V'`) was appropriately centered (or pre-processed according to `x$preproc`). The bootstrap samples are generated based on the components derived from this pre-processed data.

Value

A `list` object of class `bootstrap_pca_result` containing:

`E_Vb`	Matrix (p x k) of the estimated bootstrap means of the principal components (loadings V^b = coefficients).
`sd_Vb`	Matrix (p x k) of the estimated bootstrap standard deviations of the principal components (loadings V^b).
`z_loadings`	Matrix (p x k) of the bootstrap Z-scores for the loadings, calculated as `E_Vb / sd_Vb`.
`E_Scores`	Matrix (n x k) of the estimated bootstrap means of the principal component scores (S^b).
`sd_Scores`	Matrix (n x k) of the estimated bootstrap standard deviations of the principal component scores (S^b).
`z_scores`	Matrix (n x k) of the bootstrap Z-scores for the scores, calculated as `E_Scores / sd_Scores`.
`E_Ab`	Matrix (k x k) of the estimated bootstrap means of the internal rotation matrices A^b.
`Ab_array`	Array (k x k x nboot) containing all the bootstrap rotation matrices A^b.
`Scores_array`	Array (n x k x nboot) containing all the bootstrap score matrices (S^b, with NAs for non-sampled subjects).
`nboot`	The number of bootstrap samples used (successful ones).
`k`	The number of components bootstrapped.
`call`	The matched call to the function.

References

Fisher, Aaron, Brian Caffo, Brian Schwartz, and Vadim Zipunnikov. 2016. "Fast, Exact Bootstrap Principal Component Analysis for P > 1 Million." *Journal of the American Statistical Association* 111 (514): 846–60. doi:10.1080/01621459.2015.1062383.

Examples

```
# Simulate data (p=50, n=20)
set.seed(123)
p_dim <- 50
n_obs <- 20
Y_mat <- matrix(rnorm(p_dim * n_obs), nrow = p_dim, ncol = n_obs)
# Transpose for pca function input (n x p)
X_mat <- t(Y_mat)

# Perform PCA using the provided pca function
# Use center() pre-processing
pca_res <- pca(X_mat, ncomp = 5, preproc = center(), method = "fast")

# Run bootstrap on the pca result
boot_res <- bootstrap_pca(pca_res, nboot = 5, k = 5, seed = 456)

# Explore results
print(dim(boot_res$z_loadings)) # p x k Z-scores for loadings (coefficients)
print(dim(boot_res$z_scores))   # n x k Z-scores for scores
```

---

bootstrap_plsc *Bootstrap inference for PLSC loadings*

Description

Provides bootstrap ratios (mean / sd) for X and Y loadings to assess stability, mirroring common practice in Behavior PLSC.

Usage

```
bootstrap_plsc(
  x,
  X,
  Y,
  nboot = 500,
  comps = ncomp(x),
  seed = NULL,
  parallel = FALSE,
  epsilon = 1e-09,
  ...
)
```

Arguments

x	A fitted plsc object.
X	Original X block.
Y	Original Y block.

nboot	Number of bootstrap samples (default 500).
comps	Number of components to bootstrap (default: ncomp(x)).
seed	Optional integer seed for reproducibility.
parallel	Use future.apply for parallelization (default FALSE).
epsilon	Small positive constant to stabilize division for ratios.
...	Additional arguments (currently unused).

---

center *center a data matrix*

Description

remove mean of all columns in matrix

Usage

```
center(preproc = prepper(), cmeans = NULL)
```

Arguments

preproc	the pre-processing pipeline
cmeans	optional vector of precomputed column means

Value

a prepper list

---

classifier *Construct a Classifier*

Description

Create a classifier from a given model object (e.g., projector). This classifier can generate predictions for new data points.

Usage

```
classifier(x, colind, ...)

## S3 method for class 'projector'
classifier(
  x,
  colind = NULL,
  labels,
  new_data = NULL,
  knn = 1,
  global_scores = TRUE,
  ...
)
```

Arguments

x	projector
colind	...
...	extra args
labels	...
new_data	...
knn	...
global_scores	...

Value

A classifier function that can be used to make predictions on new data points.

See Also

Other classifier: `classifier.multiblock_biprojector()`, `rf_classifier.projector()`

Examples

```
# Assume proj is a fitted projector object
# Assume lbls are labels and dat is new data
# classifier(proj, labels = lbls, new_data = dat, knn = 3)
```

---

classifier.discriminant_projector

Create a k-NN classifier for a discriminant projector

Description

Create a k-NN classifier for a discriminant projector

Usage

```
## S3 method for class 'discriminant_projector'
classifier(x, colind = NULL, knn = 1, ...)
```

Arguments

x	the discriminant projector object
colind	an optional vector specifying the column indices of the components
knn	the number of nearest neighbors (default=1)
...	extra arguments

Value

a classifier object

Examples

```
# Assume dp is a fitted discriminant_projector object
# classifier(dp, knn = 5) # Basic example
```

---

classifier.multiblock_biprojector

Multiblock Bi-Projector Classifier

Description

Constructs a k-Nearest Neighbors (k-NN) classifier based on a fitted `multiblock_biprojector` model object. The classifier uses the projected scores as the feature space for k-NN.

Usage

```
## S3 method for class 'multiblock_biprojector'
classifier(
  x,
  colind = NULL,
  labels,
  new_data = NULL,
  block = NULL,
  global_scores = TRUE,
  knn = 1,
  ...
)
```

Arguments

x	A fitted multiblock_biprojector object.
colind	An optional numeric vector specifying column indices from the original data space. If provided when global_scores=FALSE, these indices are used to perform a partial projection for the reference scores. If provided when global_scores=TRUE, this value is stored but does not affect the reference scores (which remain global); however, it may influence the default projection behavior during prediction unless overridden there. See predict.classifier.
labels	A factor or vector of class labels for the training data.
new_data	An optional data matrix used to generate reference scores when global_scores=FALSE, or when global_scores=TRUE but colind or block is also provided (overriding global_scores). Must be provided if global_scores=FALSE.
block	An optional integer specifying a predefined block index. Used for partial projection if global_scores=FALSE or if new_data is also provided. Cannot be used simultaneously with colind.
global_scores	Logical. **DEPRECATED** This argument is deprecated and its behavior has changed. Reference scores are now determined automatically: • If new_data is NULL: Uses the globally projected scores stored in x (scores(x)). • If new_data is provided: Always projects new_data to generate reference scores (using partial_project/project_block if colind/block are given, project otherwise).
knn	The integer number of nearest neighbors (k) for the k-NN algorithm (default: 1).
...	Additional arguments (currently ignored).

Details

Users can specify whether to use the globally projected scores stored within the model (global_scores = TRUE) or to generate reference scores by projecting provided new_data (global_scores = FALSE). Partial projections based on colind or block can be used when global_scores = FALSE or when new_data is provided alongside colind/block. Prediction behavior is further controlled by arguments passed to predict.classifier.

Value

An object of class multiblock_classifier, which also inherits from classifier.

See Also

Other classifier: classifier(), rf_classifier.projector()

coef.composed_projector *Get Coefficients of a Composed Projector*

Description

Calculates the effective coefficient matrix that maps from the original input space (of the first projector) to the final output space (of the last projector). This is done by multiplying the coefficient matrices of all projectors in the sequence.

Usage

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

Arguments

object	A composed_projector object.
...	Currently unused.

Value

A matrix representing the combined coefficients.

coef.cross_projector *Extract coefficients from a cross_projector object*

Description

Extract coefficients from a cross_projector object

Usage

```
## S3 method for class 'cross_projector'
coef(object, source = c("X", "Y"), ...)
```

Arguments

object	the model fit
source	the source of the data (X or Y block), either "X" or "Y"
...	extra args

Value

the coefficients

coef.multiblock_projector *Coefficients for a Multiblock Projector*

Description

Extracts the components (loadings) for a given block or the entire projector.

Usage

```
## S3 method for class 'multiblock_projector'
coef(object, block, ...)
```

Arguments

object	A multiblock_projector object.
block	Optional block index. If missing, returns loadings for all variables.
...	Additional arguments.

Value

A matrix of loadings.

colscale *scale a data matrix*

Description

normalize each column by a scale factor.

Usage

```
colscale(preproc = prepper(), type = c("unit", "z", "weights"), weights = NULL)
```

Arguments

preproc	the pre-processing pipeline
type	the kind of scaling, unit norm, z-scoring, or precomputed weights
weights	optional precomputed weights

Value

a prepper list

components *get the components*

Description

Extract the component matrix of a fit.

Usage

```
components(x, ...)
```

Arguments

x	the model fit
...	extra args

Value

the component matrix

compose_partial_projector *Compose Multiple Partial Projectors*

Description

Creates a `composed_partial_projector` object that applies partial projections sequentially. If multiple projectors are composed, the column indices (colind) used at each stage must be considered.

This infix operator provides syntactic sugar for composing projectors sequentially. It is an alias for `compose_partial_projector`.

Usage

```
compose_partial_projector(...)

lhs %>>% rhs
```

Arguments

...	A sequence of projectors that implement `partial_project()`, optionally named.
lhs	The left-hand side projector (or a composed projector).
rhs	The right-hand side projector to add to the sequence.

Value

A `composed_partial_projector` object.

A `composed_partial_projector` object representing the combined sequence.

`compose_projector` *Compose Two Projectors*

Description

Combine two projector models into a single projector by sequentially applying the first projector and then the second projector.

Usage

```
compose_projector(x, y, ...)
```

Arguments

`x`	A fitted model object (e.g., `projector`) that has been fit to a dataset and will be applied first in the composition.
`y`	A second fitted model object (e.g., `projector`) that has been fit to a dataset and will be applied after the first projector.
`...`	Additional arguments to be passed to the specific model implementation of `compose_projector`.

Value

A new `projector` object representing the composed projector, which can be used to project data onto the combined subspace.

`concat_pre_processors` *bind together blockwise pre-processors*

Description

concatenate a sequence of pre-processors, each applied to a block of data.

Usage

```
concat_pre_processors(preprocs, block_indices)
```

Arguments

`preprocs`	a list of initialized `pre_processor` objects
`block_indices`	a list of integer vectors specifying the global column indices for each block

Value

a new pre_processor object that applies the correct transformations blockwise

Examples

```
p1 <- center() |> prep()
p2 <- center() |> prep()

x1 <- rbind(1:10, 2:11)
x2 <- rbind(1:10, 2:11)

p1a <- init_transform(p1,x1)
p2a <- init_transform(p2,x2)

clist <- concat_pre_processors(list(p1,p2), list(1:10, 11:20))
t1 <- apply_transform(clist, cbind(x1,x2))

t2 <- apply_transform(clist, cbind(x1,x2[,1:5]), colind=1:15)
```

---

cPCAplus

Contrastive PCA++ (cPCA++) Performs Contrastive PCA++ (cPCA++) to find directions that capture variation enriched in a "foreground" dataset relative to a "background" dataset. This implementation follows the cPCA++ approach which directly solves the generalized eigenvalue problem Rf v = lambda Rb v, where Rf and Rb are the covariance matrices of the foreground and background data, centered using the background mean.

Description

Contrastive PCA++ (cPCA++) Performs Contrastive PCA++ (cPCA++) to find directions that capture variation enriched in a "foreground" dataset relative to a "background" dataset. This implementation follows the cPCA++ approach which directly solves the generalized eigenvalue problem Rf v = lambda Rb v, where Rf and Rb are the covariance matrices of the foreground and background data, centered using the *background mean*.

Usage

```
cPCAplus(
  X_f,
  X_b,
  ncomp = NULL,
  center_background = TRUE,
  lambda = 0,
  method = c("geigen", "primme", "sdiag", "corpcor"),
  strategy = c("auto", "feature", "sample"),
  verbose = getOption("multivarious.verbose", TRUE),
  sample_rank = NULL,
```

```
  sample_oversample = 10L,
  ...
)
```

Arguments

X_f	A numeric matrix representing the foreground dataset (samples x features).
X_b	A numeric matrix representing the background dataset (samples x features). X_f and X_b must have the same number of features (columns).
ncomp	Integer. The number of contrastive components to compute. Defaults to min(ncol(X_f), nrow(X_f), nrow(X_b)), and may be further capped by the effective background rank (especially under the sample-space strategy).
center_background	Logical. If TRUE (default), both X_f and X_b are centered using the column means of X_b. If FALSE, it assumes data is already appropriately centered.
lambda	Shrinkage intensity for covariance estimation (0 <= lambda <= 1). Defaults to 0 (no shrinkage). Uses corpcor::cov.shrink. Can help stabilize results if Rb is ill-conditioned or singular.
method	A character string specifying the primary computation method. Options include: • "geigen" (Default): Use geneig from the geigen package. • "primme": Use geneig with the PRIMME library backend (requires special geigen build). • "sdiag": Use geneig with a spectral decomposition method. • "corpcor": Use a corpcor-based whitening approach followed by standard PCA.
strategy	Controls the GEVD approach when method is not "corpcor". Options include: • "auto" (Default): Chooses based on dimensions (feature vs. sample space). • "feature": Forces direct computation via p x p covariance matrices. • "sample": Forces sample-space computation via SVD and a smaller GEVD (efficient for large p).
verbose	Logical; if TRUE (default), prints brief status messages about strategy selection and defaults. Set to FALSE to silence these messages.
sample_rank	Optional integer controlling the background subspace rank used in the sample-space strategy. If NULL (default), uses the full background rank min(n_b-1, p). If provided, the solver will target approximately sample_rank + sample_oversample and will be bounded above by the full background rank.
sample_oversample	Integer oversampling margin (default 10) applied when sample_rank is given. Ignored when sample_rank is NULL.
...	Additional arguments passed to the underlying computation functions (geigen::geneig or irlba::irlba based on method and strategy).

Details

Preprocessing: Following the cPCA++ paper, if `center_background = TRUE`, both `X_f` and `X_b` are centered by subtracting the column means calculated *only* from the background data `X_b`. This is crucial for isolating variance specific to `X_f`.

Core Algorithm (methods "geigen", "primme", "sdiag", strategy="feature"):

1. Center `X_f` and `X_b` using the mean of `X_b`.
2. Compute potentially shrunk $p \times p$ covariance matrices `Rf` (from centered `X_f`) and `Rb` (from centered `X_b`) using `corpcor::cov.shrink`.
3. Solve the generalized eigenvalue problem `Rf v = lambda Rb v` for the top `ncomp` eigenvectors `v` using `geigen::geneig`. These eigenvectors are the contrastive principal components (loadings).
4. Compute scores by projecting the centered foreground data onto the eigenvectors: `S = X_f_centered %*% v`.

Core Algorithm (Large-D / Sample Space Strategy, strategy="sample"): When $p \gg n$, forming $p \times p$ matrices `Rf` and `Rb` is infeasible. The "sample" strategy follows cPCA++ §3.2:

1. Center `X_f` and `X_b` using the mean of `X_b`.
2. Compute the SVD of centered $X_b = UbSbVb^T$ (using `irlba` for efficiency).
3. Project centered `X_f` into the background's principal subspace: `Zf = X_f_centered %*% Vb`.
4. Form small $r \times r$ matrices: `Rf_small = cov(Zf)` and `Rb_small = (1/(n_b-1)) * Sb^2`.
5. Solve the small $r \times r$ GEVD: `Rf_small w = lambda Rb_small w` using `geigen::geneig`.
6. Lift eigenvectors back to feature space: `v = Vb %*% w`.
7. Compute scores: `S = X_f_centered %*% v`.

Alternative Algorithm (method "corpcor"):

1. Center `X_f` and `X_b` using the mean of `X_b`.
2. Compute `Rb` and its inverse square root `Rb_inv_sqrt`.
3. Whiten the foreground data: `X_f_whitened = X_f_centered %*% Rb_inv_sqrt`.
4. Perform standard PCA (`stats::prcomp`) on `X_f_whitened`.
5. The returned `v` and `s` are the loadings and scores *in the whitened space*. The loadings are *not* the generalized eigenvectors `v`. A specific class `corpcor_pca` is added to signal this.

Value

A `bi_projector`-like object with classes `c("cPCAplus", "<method_class>", "bi_projector")` containing:

v Loadings matrix (features x ncomp). Interpretation depends on `method` (see Details).

s Scores matrix (samples_f x ncomp).

sdev Vector (length ncomp). Standard deviations (sqrt of generalized eigenvalues for `geigen` methods, PCA std devs for `corpcor`).

values Vector (length ncomp). Generalized eigenvalues (for `geigen` methods) or PCA eigenvalues (for `corpcor`).

strategy The strategy used ("feature" or "sample") if method was not "corpcor".

preproc The initialized `preprocessor` object used.

method The computation method used.

ncomp The number of components computed.

nfeatures The number of features.

References

Abid, A., Zhang, M. J., Bagaria, V. K., & Zou, J. (2018). Exploring patterns enriched in a dataset with contrastive principal component analysis. Nature Communications, 9(1), 2134.

Salloum, R., & Kuo, C. C. J. (2022). cPCA++: An efficient method for contrastive feature learning. Pattern Recognition, 124, 108378.

Wu, M., Sun, Q., & Yang, Y. (2025). PCA++: How Uniformity Induces Robustness to Background Noise in Contrastive Learning. arXiv preprint arXiv:2511.12278.

Woller, J. P., Menrath, D., & Gharabaghi, A. (2025). Generalized contrastive PCA is equivalent to generalized eigendecomposition. PLOS Computational Biology, 21(10), e1013555.

Examples

```
# Simulate data where foreground has extra variance in first few dimensions
set.seed(123)
n_f <- 100
n_b <- 150
n_features <- 50

# Background: standard normal noise
X_b <- matrix(rnorm(n_b * n_features), nrow=n_b, ncol=n_features)
colnames(X_b) <- paste0("Feat_", 1:n_features)

# Foreground: background noise + extra variance in first 5 features
X_f_signal <- matrix(rnorm(n_f * 5, mean=0, sd=2), nrow=n_f, ncol=5)
X_f_noise <- matrix(rnorm(n_f * (n_features-5)), nrow=n_f, ncol=n_features-5)
X_f <- cbind(X_f_signal, X_f_noise) + matrix(rnorm(n_f * n_features), nrow=n_f, ncol=n_features)
colnames(X_f) <- paste0("Feat_", 1:n_features)
rownames(X_f) <- paste0("SampleF_", 1:n_f)

# Apply cPCA++ (requires geigen and corpcor packages)
# install.packages(c("geigen", "corpcor"))
if (requireNamespace("geigen", quietly = TRUE) && requireNamespace("corpcor", quietly = TRUE)) {
  # Assuming helper constructors like bi_projector are available
  # library(multivarious)

  res_cpca_plus <- cPCAplus(X_f, X_b, ncomp = 5, method = "geigen")

  # Scores for the foreground data (samples x components)
  print(head(res_cpca_plus$s))
```

```
  # Loadings (contrastive directions) (features x components)
  print(head(res_cpca_plus$v))
}


# Plot example (slow graphics)
if (requireNamespace("geigen", quietly = TRUE) && requireNamespace("corpcor", quietly = TRUE)) {
  set.seed(123)
  X_b <- matrix(rnorm(150 * 50), nrow=150, ncol=50)
  X_f <- cbind(matrix(rnorm(100*5, sd=2), 100, 5), matrix(rnorm(100*45), 100, 45))
  res <- cPCAplus(X_f, X_b, ncomp = 5, method = "geigen")
  plot(res$s[, 1], res$s[, 2],
       xlab = "Contrastive Component 1", ylab = "Contrastive Component 2",
       main = "cPCA++ Scores")
}
```

`cross_projector` *Two-way (cross) projection to latent components*

Description

A projector that reduces two blocks of data, X and Y, yielding a pair of weights for each component. This structure can be used, for example, to store weights derived from canonical correlation analysis.

Usage

```
cross_projector(
  vx,
  vy,
  preproc_x = prep(pass()),
  preproc_y = prep(pass()),
  ...,
  classes = NULL
)
```

Arguments

`vx`	the X coefficients. Must have the same number of columns as `vy`.
`vy`	the Y coefficients. Must have the same number of columns as `vx`.
`preproc_x`	the X pre-processor
`preproc_y`	the Y pre-processor
`...`	extra parameters or results to store
`classes`	additional class names

Details

This class extends `projector` and therefore basic operations such as `project`, `shape`, `reprocess`, and `coef` work, but by default, it is assumed that the `X` block is primary. To access `Y` block operations, an additional argument `source` must be supplied to the relevant functions, e.g., `coef(fit, source = "Y")`

Value

a cross_projector object

Examples

```
# Create two scaled matrices X and Y
X <- scale(matrix(rnorm(10 * 5), 10, 5))
Y <- scale(matrix(rnorm(10 * 5), 10, 5))

# Perform canonical correlation analysis on X and Y
cres <- cancor(X, Y)
sx <- X %*% cres$xcoef
sy <- Y %*% cres$ycoef

# Create a cross_projector object using the canonical correlation analysis results
canfit <- cross_projector(cres$xcoef, cres$ycoef, cor = cres$cor,
                          sx = sx, sy = sy, classes = "cancor")
```

---

cv *Cross-validation Framework*

Description

Generic function for performing cross-validation on various objects or data. Specific methods should be implemented for different data types or model types.

Usage

```
cv(x, folds, ...)
```

Arguments

`x`	The object to perform cross-validation on (e.g., data matrix, formula, model object).
`folds`	A list defining the cross-validation folds, typically containing `train` and `test` indices for each fold.
`...`	Additional arguments passed to specific methods.

Details

The specific implementation details, default functions, and relevant arguments vary by method.

Bi-Projector Method (`cv.bi_projector`): Relevant arguments: `x`, `folds`, `max_comp`, `fit_fun`, `measure`, `measure_fun`, `return_models`, `....`

This method performs cross-validation specifically for `bi_projector` models (or models intended to be used like them, typically from unsupervised methods like PCA or SVD). For each fold, it fits a single model using the training data with the maximum number of components specified (`max_comp`). It then iterates from 1 to `max_comp` components:

1. It truncates the full model to k components using `truncate()`. (Requires a `truncate` method for the fitted model class).
2. It reconstructs the held-out test data using the k-component truncated model via `reconstruct_new()`.
3. It calculates reconstruction performance metrics (e.g., MSE, R2) by comparing the original test data to the reconstruction using the `measure` argument or a custom `measure_fun`.

The `fit_fun` must accept an argument `ncomp`. Additional arguments in `...` are passed to `fit_fun` and `measure_fun`.

The return value is a `cv_fit` object (a list with class `cv_fit`), where the `$results` element is a tibble. Each row corresponds to a fold, containing the fold index (`fold`) and a nested tibble (`component_metrics`). The `component_metrics` tibble has rows for each component evaluated (1 to `max_comp`) and columns for the component index (`comp`) plus all calculated metrics (e.g., `mse`, `r2`, `mae`) or error messages (`comp_error`). If `return_models=TRUE`, the full model fitted on the training data for each fold is included in a list column `model_full`.

Value

The structure of the return value depends on the specific S3 method. Typically, it will be an object containing the results of the cross-validation, such as performance metrics per fold or aggregated metrics.

See Also

`cv_generic`

`cv_generic` *Generic cross-validation engine*

Description

For each fold (train/test indices):

1. Subset `data[train, ]`
2. Fit a model with `.fit_fun(train_data, ...)`
3. Evaluate with `.measure_fun(model, test_data, ...)`

Usage

```
cv_generic(
  data,
  folds,
  .fit_fun,
  .measure_fun,
  fit_args = list(),
  measure_args = list(),
  backend = c("serial", "future"),
  ...
)
```

Arguments

`data`	A matrix or data.frame of shape (n x p).
`folds`	A list of folds, each a list with `$train` and `$test`.
`.fit_fun`	Function: signature `function(train_data, ...){}`. Returns a fitted model.
`.measure_fun`	Function: signature `function(model, test_data, ...){}`. Returns a tibble or named list/vector of metrics.
`fit_args`	A list of additional named arguments passed to `.fit_fun`.
`measure_args`	A list of additional named arguments passed to `.measure_fun`.
`backend`	Character string: "serial" (default) or "future" for parallel execution using the `future` framework.
`...`	Currently ignored (arguments should be passed via `fit_args` or `measure_args`).

Value

A tibble with columns:

`fold`	integer fold index
`model`	list of fitted models
`metrics`	list of metric tibbles/lists

`discriminant_projector` *Construct a Discriminant Projector*

Description

A `discriminant_projector` is an instance that extends `bi_projector` with a projection that maximizes class separation. This can be useful for dimensionality reduction techniques that take class labels into account, such as Linear Discriminant Analysis (LDA).

Usage

```
discriminant_projector(
  v,
  s,
  sdev,
  preproc = prep(pass()),
  labels,
  classes = NULL,
  ...
)
```

Arguments

v	The projection matrix (often X %*% v). Rows correspond to observations, columns to components.
s	The score matrix (often X %*% v). Rows correspond to observations, columns to components.
sdev	The standard deviations associated with the scores or components (e.g., singular values from LDA).
preproc	A prepper or pre_processor object, or a pre-processing function (e.g., center, pass).
labels	A factor or character vector of class labels corresponding to the rows of X (and s).
classes	Additional S3 classes to prepend.
...	Extra arguments passed to bi_projector.

Value

A discriminant_projector object.

See Also

bi_projector

Examples

```
# Simulate data and labels
set.seed(123)
X <- matrix(rnorm(100 * 10), 100, 10)
labels <- factor(rep(1:2, each = 50))

# Perform LDA and create a discriminant projector
lda_fit <- MASS::lda(X, labels)

dp <- discriminant_projector(lda_fit$scaling, X %*% lda_fit$scaling, sdev = lda_fit$svd,
labels = labels)
```

feature_importance *Evaluate feature importance*

Description

Calculate the importance of features in a model

Usage

```
feature_importance(x, ...)
```

Arguments

x	the model fit
...	extra args

Value

the feature importance scores

feature_importance.classifier *Evaluate Feature Importance for a Classifier*

Description

Estimates the importance of features or blocks of features for the classification performance using either a "marginal" (leave-one-block-out) or "standalone" (use-only-one-block) approach.

Usage

```
## S3 method for class 'classifier'
feature_importance(
  x,
  new_data,
  true_labels,
  ncomp = NULL,
  blocks = NULL,
  metric = c("cosine", "euclidean", "ejaccard"),
  fun = rank_score,
  fun_direction = c("lower_is_better", "higher_is_better"),
  approach = c("marginal", "standalone"),
  ...
)
```

Arguments

x	A fitted classifier object.
new_data	The data matrix used for evaluating importance (typically validation or test data).
true_labels	The true class labels corresponding to the rows of new_data.
ncomp	Optional integer; the number of components to use from the projector for classification (default: all components used during classifier creation).
blocks	A list where each element is a numeric vector of feature indices (columns in the original data space) defining a block. If NULL, each feature is treated as its own block.
metric	Character string specifying the similarity or distance metric for k-NN. Choices: "euclidean", "cosine", "ejaccard".
fun	A function to compute the performance metric (e.g., rank_score, topk, or a custom function). The function should take a probability matrix and observed labels and return a data frame where the first column is the metric value per observation.
fun_direction	Character string, either "lower_is_better" or "higher_is_better", indicating whether lower or higher values of the metric calculated by fun signify better performance. This is used to interpret the importance score correctly.
approach	Character string: "marginal" (calculates importance as change from baseline when block is removed) or "standalone" (calculates importance as performance using only the block).
...	Additional arguments passed to predict.classifier during internal predictions.

Details

Importance is measured by the change in a performance metric (fun) when features are removed (marginal) or used exclusively (standalone).

Value

A data.frame with columns block (character representation of feature indices in the block) and importance (numeric importance score). Higher importance values generally indicate more influential blocks, considering fun_direction.

See Also

rank_score, topk

Examples

```
# Assume clf is a fitted classifier object, dat is new data, true_lbls are correct labels for dat
# Assume blocks_list defines feature groups e.g., list(1:5, 6:10)
# feature_importance(clf, new_data = dat, true_labels = true_lbls, blocks = blocks_list)
```

`fit` *Fit a preprocessing pipeline*

Description

Learn preprocessing parameters from training data. This function fits the preprocessing pipeline to the provided data matrix, learning parameters such as means, standard deviations, or other transformation parameters.

Usage

```
fit(object, X, ...)
```

Arguments

`object`	A preprocessing object (e.g., `prepper` or `pre_processor`)
`X`	A matrix or data frame to fit the preprocessing pipeline to
`...`	Additional arguments passed to methods

Value

A fitted preprocessing object that can be used with `transform()` and `inverse_transform()`

See Also

`fit_transform()`, `transform()`, `inverse_transform()`

Examples

```
# Fit a centering preprocessor
X <- matrix(rnorm(100), 10, 10)
preproc <- center()
fitted_preproc <- fit(preproc, X)

# Transform new data
X_new <- matrix(rnorm(50), 5, 10)
X_transformed <- transform(fitted_preproc, X_new)
```

`fit_transform` *Fit and transform data in one step*

Description

Convenience function that fits a preprocessing pipeline to data and immediately applies the transformation. This is equivalent to calling `fit()` followed by `transform()` but is more efficient and convenient.

Usage

```
fit_transform(object, X, ...)
```

Arguments

`object`	A preprocessing object (e.g., `prepper` or `pre_processor`)
`X`	A matrix or data frame to fit and transform
`...`	Additional arguments passed to methods

Value

A list with two elements: `preproc` (the fitted preprocessor) and `transformed` (the transformed data)

See Also

`fit()`, `transform()`, `inverse_transform()`

Examples

```
# Fit and transform in one step
X <- matrix(rnorm(100), 10, 10)
preproc <- center()
result <- fit_transform(preproc, X)
fitted_preproc <- result$preproc
X_transformed <- result$transformed
```

fresh *Get a fresh pre-processing node cleared of any cached data*

Description

Get a fresh pre-processing node cleared of any cached data

Usage

```
fresh(x, ...)
```

Arguments

x	the processing pipeline
...	extra args

Value

a fresh pre-processing pipeline

geneig *Generalized Eigenvalue Decomposition*

Description

Computes the generalized eigenvalues and eigenvectors for the problem: A x = lambda B x. Supports multiple dense and iterative solvers with a unified eigenpair selection interface.

Usage

```
geneig(
  A = NULL,
  B = NULL,
  ncomp = 2,
  preproc = prep(pass()),
  method = c("robust", "sdiag", "geigen", "primme", "rspectra", "subspace"),
  which = "LA",
  ...
)
```

Arguments

A	The left-hand side square matrix.
B	The right-hand side square matrix, same dimension as A.
ncomp	Number of eigenpairs to return.
preproc	A preprocessing function to apply to the matrices before solving the generalized eigenvalue problem.
method	One of: • "robust": Uses a stable decomposition via a whitening transform (B must be symmetric PD). • "sdiag": Uses a spectral decomposition of B (must be symmetric PD). Requires A to be symmetric for meaningful results. • "geigen": Uses the **geigen** package for a general solution (A and B can be non-symmetric). • "primme": Uses the **PRIMME** package for large/sparse symmetric problems (A and B must be symmetric). • "rspectra": Uses **RSpectra**; if B is SPD it calls `eigs_sym(A, B, ...)` directly, otherwise it applies a reciprocal transform to support all targets. • "subspace": Block subspace iteration for symmetric pairs with SPD B (iterative, no external package required).
which	Which eigenpairs to return. One of `"LA"` (largest algebraic), `"SA"` (smallest algebraic), `"LM"` (largest magnitude), or `"SM"` (smallest magnitude). Aliases: `"top"`/`"largest"` -> `"LA"`, `"bottom"`/`"smallest"` -> `"SA"`. Dense backends select eigenpairs post hoc; `"primme"` supports `"LA"`, `"SA"`, `"SM"` (not `"LM"`); `"rspectra"` honors all four options. Default is `"LA"`.
...	Additional arguments to pass to the underlying solver.

Value

A `projector` object with generalized eigenvectors and eigenvalues.

References

Golub, G. H. & Van Loan, C. F. (2013) *Matrix Computations*, 4th ed., Section 8.7 – textbook derivation for the "robust" (Cholesky) and "sdiag" (spectral) transforms.

Moler, C. & Stewart, G. (1973) "An Algorithm for Generalized Matrix Eigenvalue Problems". *SIAM J. Numer. Anal.*, 10 (2): 241-256 – the QZ algorithm behind the `geigen` backend.

Stathopoulos, A. & McCombs, J. R. (2010) "PRIMME: PReconditioned Iterative Multi-Method Eigensolver". *ACM TOMS* 37 (2): 21:1-21:30 – the algorithmic core of the `primme` backend.

See also the **geigen** (CRAN) and **PRIMME** documentation.

See Also

`projector` for the base class structure.

Examples

```
# Simulate two matrices
set.seed(123)
A <- matrix(rnorm(50 * 50), 50, 50)
B <- matrix(rnorm(50 * 50), 50, 50)
A <- A %*% t(A) # Make A symmetric
B <- B %*% t(B) + diag(50) * 0.1 # Make B symmetric positive definite

# Solve generalized eigenvalue problem
result <- geneig(A = A, B = B, ncomp = 3)
```

---

group_means *Compute column-wise mean in X for each factor level of Y*

Description

This function computes group means for each factor level of Y in the provided data matrix X.

Usage

```
group_means(Y, X)
```

Arguments

Y	a vector of labels to compute means over disjoint sets
X	a data matrix from which to compute means

Value

a matrix with row names corresponding to factor levels of Y and column-wise means for each factor level

Examples

```
# Example data
X <- matrix(rnorm(50), 10, 5)
Y <- factor(rep(1:2, each = 5))

# Compute group means
gm <- group_means(Y, X)
```

inverse_projection *Inverse of the Component Matrix*

Description

Return the inverse projection matrix, which can be used to map back to data space. If the component matrix is orthogonal, then the inverse projection is the transpose of the component matrix.

Usage

```
inverse_projection(x, ...)

## S3 method for class 'projector'
inverse_projection(x, ...)
```

Arguments

x	The model fit.
...	Extra arguments.

Value

The inverse projection matrix.

See Also

project for projecting data onto the subspace.

inverse_projection.composed_projector *Compute the Inverse Projection for a Composed Projector*

Description

Calculates the pseudo-inverse of the composed projector, mapping from the final output space back towards the original input space. This is computed by multiplying the pseudo-inverses of the individual projector stages in reverse order: `V_k+ %*% ... %*% V_2+ %*% V_1+`.

Usage

```
## S3 method for class 'composed_projector'
inverse_projection(x, ...)
```

Arguments

x	A composed_projector object.
...	Additional arguments passed to the underlying inverse_projection methods.

Details

Requires that each stage implements the inverse_projection method.

Value

A matrix representing the combined pseudo-inverse.

---

inverse_projection.cross_projector

Default inverse_projection method for cross_projector

Description

This function obtains the matrix that maps factor scores in the latent space back into the original domain (X or Y). By default, we assume v_domain is *not* necessarily orthonormal or invertible, so we use a pseudoinverse approach (e.g. MASS::ginv).

Usage

```
## S3 method for class 'cross_projector'
inverse_projection(x, domain = c("X", "Y"), ...)
```

Arguments

x	A cross_projector object.
domain	Either "X" or "Y", indicating which block's inverse loading matrix we want (i.e., if you want to reconstruct data in the X space or Y space).
...	Additional arguments (currently unused, but may be used by subclasses).

Value

A matrix that, when multiplied by the factor scores, yields the reconstruction in the specified domain's original space.

Examples

```
# Suppose 'cp' is a cross_projector object. If we want the
# inverse for the Y domain:
#   inv_mat <- inverse_projection(cp, domain="Y")
# Then reconstruct:  Yhat <- Fscores %*% inv_mat
```

`inverse_transform` *Inverse transform data using a fitted preprocessing pipeline*

Description

Reverse the preprocessing transformation, converting transformed data back to the original scale. The preprocessing object must have been fitted before calling this function.

Usage

```
inverse_transform(object, X, ...)
```

Arguments

`object`	A fitted preprocessing object
`X`	A matrix or data frame of transformed data to reverse
`...`	Additional arguments passed to methods

Value

The data matrix in original scale

See Also

`fit()`, `fit_transform()`, `transform()`

Examples

```
# Inverse transform data back to original scale
X <- matrix(rnorm(100), 10, 10)
preproc <- center()
fitted_preproc <- fit(preproc, X)
X_transformed <- transform(fitted_preproc, X)
X_reconstructed <- inverse_transform(fitted_preproc, X_transformed)

# X and X_reconstructed should be approximately equal
all.equal(X, X_reconstructed)
```

is_orthogonal *is it orthogonal*

Description

test whether components are orthogonal

Usage

```
is_orthogonal(x, tol = 1e-06)
```

Arguments

x	the object
tol	tolerance for checking orthogonality

Value

a logical value indicating whether the transformation is orthogonal

is_orthogonal.projector *Stricter check for true orthogonality*

Description

We test if v^T * v = I (when rows >= cols) or v * v^T = I (when cols > rows).

Usage

```
## S3 method for class 'projector'
is_orthogonal(x, tol = 1e-06)
```

Arguments

x	the projector object
tol	tolerance for checking orthogonality

measure_interblock_transfer_error

Compute inter-block transfer error metrics for a cross_projector

Description

We measure how well the model can transfer from X->Y or Y->X, e.g. "x2y.mse".

Usage

```
measure_interblock_transfer_error(Xtrue, Ytrue, model, metrics = c("x2y.mse"))
```

Arguments

`Xtrue`	The X block test data
`Ytrue`	The Y block test data
`model`	The fitted `cross_projector`
`metrics`	A character vector like `c("x2y.mse","y2x.r2")`

Details

The metric names are of the form "x2y.mse", "x2y.rmse", "y2x.r2", etc.

Value

A 1-row tibble with columns for each requested metric

measure_reconstruction_error

Compute reconstruction-based error metrics

Description

Given two numeric matrices `Xtrue` and `Xrec`, compute:

- MSE (`"mse"`)
- RMSE (`"rmse"`)
- R^2 (`"r2"`)
- MAE (`"mae"`)

Usage

```
measure_reconstruction_error(
  Xtrue,
  Xrec,
  metrics = c("mse", "rmse", "r2"),
  by_column = FALSE
)
```

Arguments

Xtrue	Original data matrix, shape (n x p).
Xrec	Reconstructed data matrix, shape (n x p).
metrics	Character vector of metric names, e.g. c("mse","rmse","r2","mae").
by_column	Logical, if TRUE calculate R2 metric per column and average (default: FALSE).

Value

A one-row tibble with columns matching metrics.

---

multiblock_biprojector

Create a Multiblock Bi-Projector

Description

Constructs a multiblock bi-projector using the given component matrix (v), score matrix (s), singular values (sdev), a preprocessing function, and a list of block indices. This allows for two-way mapping with multiblock data.

Usage

```
multiblock_biprojector(
  v,
  s,
  sdev,
  preproc = prep(pass()),
  ...,
  block_indices,
  classes = NULL
)
```

Arguments

v	A matrix of components (nrow = number of variables, ncol = number of components).
s	A matrix of scores (nrow = samples, ncol = components).
sdev	A numeric vector of singular values or standard deviations.
preproc	A pre-processing object (default: prep(pass())).
...	Extra arguments.
block_indices	A list of numeric vectors specifying data block variable indices.
classes	Additional class attributes (default NULL).

Value

A multiblock_biprojector object.

See Also

bi_projector, multiblock_projector

---

multiblock_projector *Create a Multiblock Projector*

Description

Constructs a multiblock projector using the given component matrix (v), a preprocessing function, and a list of block indices. This allows for the projection of multiblock data, where each block represents a different set of variables or features.

Usage

```
multiblock_projector(
  v,
  preproc = prep(pass()),
  ...,
  block_indices,
  classes = NULL
)
```

Arguments

v	A matrix of components with dimensions nrow(v) by ncol(v) (columns = number of components).
preproc	A pre-processing function for the data (default: prep(pass())).
...	Extra arguments.
block_indices	A list of numeric vectors specifying the indices of each data block.
classes	(optional) A character vector specifying additional class attributes of the object, default is NULL.

Value

A multiblock_projector object.

See Also

projector

Examples

```
# Generate some example data
X1 <- matrix(rnorm(10 * 5), 10, 5)
X2 <- matrix(rnorm(10 * 5), 10, 5)
X <- cbind(X1, X2)

# Compute PCA on the combined data
pc <- pca(X, ncomp = 8)

# Create a multiblock projector using PCA components and block indices
mb_proj <- multiblock_projector(pc$v, block_indices = list(1:5, 6:10))

# Project multiblock data using the multiblock projector
mb_scores <- project(mb_proj, X)
```

---

nblocks *get the number of blocks*

Description

The number of data blocks in a multiblock element

Usage

```
nblocks(x)
```

Arguments

x	the object

Value

the number of blocks

ncomp *Get the number of components*

Description

This function returns the total number of components in the fitted model.

Usage

```
ncomp(x)
```

Arguments

x	A fitted model object.

Value

The number of components in the fitted model.

Examples

```
# Example using the svd_wrapper function
data(iris)
X <- as.matrix(iris[, 1:4])
fit <- svd_wrapper(X, ncomp = 3, preproc = center(), method = "base")
ncomp(fit) # Should return 3
```

nystrom_approx *Nyström approximation for kernel-based decomposition (Unified Version)*

Description

Approximate the eigen-decomposition of a large kernel matrix K using either the standard Nyström method (Williams & Seeger, 2001) or the Double Nyström method (Lim et al., 2015, Algorithm 3).

Usage

```
nystrom_approx(
  X,
  kernel_func = NULL,
  ncomp = NULL,
  landmarks = NULL,
  nlandmarks = 10,
  preproc = pass(),
  method = c("standard", "double"),
```

```
  center = FALSE,
  l = NULL,
  use_RSpectra = TRUE,
  ...
)
```

Arguments

X	A numeric matrix or data frame of size (N x D), where N is number of samples.
kernel_func	A kernel function with signature kernel_func(X, Y, ...). If NULL, defaults to a linear kernel: X %*% t(Y).
ncomp	Number of components (eigenvectors/eigenvalues) to return. Cannot exceed the number of landmarks. Default capped at length(landmarks).
landmarks	A vector of row indices (1-based, from X) specifying the landmark points. If NULL, nlandmarks points are sampled uniformly at random.
nlandmarks	The number of landmark points to sample if landmarks is NULL. Default is 10.
preproc	A pre-processing pipeline object (e.g., from prep()) or a pre-processing function (default pass()) to apply before computing the kernel.
method	Either "standard" (the classic single-stage Nyström) or "double" (the two-stage Double Nyström method).
center	Logical. If TRUE, attempts kernel centering. Default FALSE. **Note:** True kernel centering (required for equivalence to Kernel PCA) is computationally expensive and **not fully implemented**. Setting center=TRUE currently only issues a warning. For results equivalent to standard PCA, use a linear kernel and center the input data X (e.g., via preproc). See Details.
l	Intermediate rank for the double Nyström method. Ignored if method="standard". Typically, l < length(landmarks) to reduce complexity.
use_RSpectra	Logical. If TRUE, use RSpectra::svds for partial SVD. Recommended for large problems.
...	Additional arguments passed to kernel_func.

Details

The Double Nyström method introduces an intermediate step that reduces the size of the decomposition problem, potentially improving efficiency and scalability.

Kernel Centering: Standard Kernel PCA requires the kernel matrix K to be centered in the feature space (Schölkopf et al., 1998). This implementation currently **does not perform kernel centering** by default (center=FALSE) due to computational complexity. Consequently, with non-linear kernels, the results approximate the eigen-decomposition of the *uncentered* kernel matrix, and are not strictly equivalent to Kernel PCA. If using a linear kernel, centering the input data X (e.g., using preproc=prep(center())) yields results equivalent to standard PCA, which is often sufficient.

Standard Nyström: Uses the method from Williams & Seeger (2001), including the sqrt(m/N) scaling for eigenvectors and N/m for eigenvalues (m landmarks, N samples).

Double Nyström: Implements Algorithm 3 from Lim et al. (2015).

Value

A `bi_projector` object with class "nystrom_approx" and additional fields:

`v` The eigenvectors (N x ncomp) approximating the kernel eigenbasis.

`s` The scores (N x ncomp) = v * diag(sdev), analogous to principal component scores.

`sdev` The square roots of the eigenvalues.

`preproc` The pre-processing pipeline used.

`meta` A list containing parameters and intermediate results used (method, landmarks, kernel_func, etc.).

References

Schölkopf, B., Smola, A., & Müller, K. R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5), 1299-1319.

Williams, C. K. I., & Seeger, M. (2001). Using the Nyström Method to Speed Up Kernel Machines. In *Advances in Neural Information Processing Systems 13* (pp. 682-688).

Lim, D., Jin, R., & Zhang, L. (2015). An Efficient and Accurate Nystrom Scheme for Large-Scale Data Sets. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (pp. 2765-2771).

Examples

```
set.seed(123)
# Smaller example matrix
X <- matrix(rnorm(1000*300), 1000, 300)

# Standard Nyström
res_std <- nystrom_approx(X, ncomp=5, nlandmarks=50, method="standard")
print(res_std)

# Double Nyström
res_db <- nystrom_approx(X, ncomp=5, nlandmarks=50, method="double", l=20)
print(res_db)

# Projection (using standard result as example)
scores_new <- project(res_std, X[1:10,])
head(scores_new)
```

---

partial_inverse_projection

Partial Inverse Projection of a Columnwise Subset of Component Matrix

---

Description

Compute the inverse projection of a columnwise subset of the component matrix (e.g., a sub-block). Even when the full component matrix is orthogonal, there is no guarantee that the partial component matrix is orthogonal.

Usage

```
partial_inverse_projection(x, colind, ...)
```

Arguments

x	A fitted model object, such as a projector, that has been fit to a dataset.
colind	A numeric vector specifying the column indices of the component matrix to consider for the partial inverse projection.
...	Additional arguments to be passed to the specific model implementation of partial_inverse_projection.

Value

A matrix representing the partial inverse projection.

---

partial_inverse_projection.cross_projector

Partial Inverse Projection of a Subset of the Loading Matrix in cross_projector

---

Description

This function obtains the "inverse" mapping for a columnwise subset of the loading matrix in the specified domain. In practice, if v_mat is not orthonormal or not square, we use a pseudoinverse approach (via MASS::ginv).

Usage

```
## S3 method for class 'cross_projector'
partial_inverse_projection(x, colind, domain = c("X", "Y"), ...)
```

Arguments

x	A cross_projector object.
colind	A numeric vector specifying the columns (indices) of the *latent factors* or loadings to invert. Typically these correspond to a subset of canonical components or principal components, etc.
domain	Either "X" or "Y", indicating which block's partial loadings we want to invert.
...	Additional arguments (unused by default, but may be used by subclasses).

Details

By default, this is a minimal-norm solution for partial columns of v_mat. If you need a different approach (e.g., ridge, direct solve, etc.), you can override this method in your specific class or code.

Value

A matrix of shape (length(colind) x p_block) that, when multiplied by factor scores restricted to colind columns, yields an (n x p_block) reconstruction in the original domain block.

Examples

```
# Suppose 'cp' is a cross_projector, and we want only columns 1:3 of
# the Y block factors. Then:
#   inv_mat_sub <- partial_inverse_projection(cp, colind=1:3, domain="Y")
# The shape will be (3 x pY), so factor_scores_sub (n x 3) %*% inv_mat_sub => (n x pY).
```

---

partial_inverse_projection.regress

Partial Inverse Projection for a regress *Object*

Description

This function computes a sub-block inversion of the regression coefficients, allowing you to focus on only certain columns (e.g. partial factors). If your coefficient matrix is not orthonormal or is not square, we use a pseudoinverse approach (via corpcor::pseudoinverse) to find a minimal-norm solution.

Usage

```
## S3 method for class 'regress'
partial_inverse_projection(x, colind, ...)
```

Arguments

x	A regress object (created by regress).
colind	A numeric vector specifying which columns of the *factor space* (i.e., the second dimension of x$coefficients) you want to invert. Typically these refer to a subset of canonical / PCA / PLS components.
...	Further arguments passed to or used by methods (not used here).

Value

A matrix of shape (length(colind) x nrow(x$coefficients)). When multiplied by partial factor scores (n x length(colind)), it yields an (n x nrow(x$coefficients)) reconstruction in the original domain.

partial_project *Partially project a new sample onto subspace*

Description

Project a selected subset of column indices (`colind`) of `new_data` onto the subspace defined by the model `x`. Optionally do a ridge-regularized least-squares solve if columns are non-orthonormal.

Usage

```
partial_project(x, new_data, colind, least_squares = TRUE, lambda = 1e-06, ...)
```

Arguments

`x`	The fitted model, e.g. `bi_projector`, that has a partial_project method.
`new_data`	A numeric matrix (n x length(colind)) or vector, representing the observations to be projected.
`colind`	A numeric vector of column indices in the original data space that correspond to `new_data`'s columns.
`least_squares`	Logical; if TRUE (default), do a ridge-regularized solve.
`lambda`	Numeric; ridge penalty (default 1e-6). Ignored if `least_squares=FALSE`.
`...`	Additional arguments passed to class-specific partial_project methods.

Value

A numeric matrix (n x d) of factor scores in the model's subspace, for those columns only.

partial_project.composed_partial_projector *Partial Project Through a Composed Partial Projector*

Description

Applies `partial_project()` through each projector in the composition. If `colind` is a single vector, it applies to the first projector only. Subsequent projectors apply full columns. If `colind` is a list, each element specifies the `colind` for the corresponding projector in the chain.

Usage

```
## S3 method for class 'composed_partial_projector'
partial_project(x, new_data, colind = NULL, ...)
```

Arguments

x	A composed_partial_projector object.
new_data	The input data matrix or vector.
colind	A numeric vector or a list of numeric vectors/NULLs. If a single vector, applies to the first projector only. If a list, its length should ideally match the number of projectors. colind[[i]] specifies the column indices (relative to the *input* of stage i) to use for the partial projection at stage i. A NULL entry means use full projection for that stage. If the list is shorter than the number of stages, NULL (full projection) is assumed for remaining stages. If a single numeric vector is provided, it is treated as list(colind, NULL, NULL, ...) for backward compatibility (partial only at first stage).
...	Additional arguments passed to partial_project() or project() methods.

Value

The partially projected data after all projectors are applied.

---

partial_project.cross_projector

Partially project data for a cross_projector

Description

Projects new data from either the X or Y domain onto the latent subspace, considering only a specified subset of original features (colind).

Usage

```
## S3 method for class 'cross_projector'
partial_project(
  x,
  new_data,
  colind,
  least_squares = TRUE,
  lambda = 1e-06,
  source = c("X", "Y"),
  ...
)
```

Arguments

x	A cross_projector object.
new_data	A numeric matrix (n x length(colind)) or vector, representing the observations corresponding to the columns specified by colind.
colind	A numeric vector of column indices in the original data space (either X or Y domain, specified by source) that correspond to new_data's columns.

least_squares	Logical; if TRUE (default), use ridge-regularized least squares for projection.
lambda	Numeric; ridge penalty (default 1e-6). Ignored if least_squares=FALSE.
source	Character, either "X" or "Y", indicating which domain new_data and colind belong to.
...	Additional arguments (currently ignored).

Value

A numeric matrix (n x d) of factor scores in the latent subspace.

---

partial_projector *Construct a partial projector*

Description

Create a new projector instance restricted to a subset of input columns. This function allows for the generation of a new projection object that focuses only on the specified columns, enabling the projection of data using a limited set of variables.

Usage

```
partial_projector(x, colind, ...)
```

Arguments

x	The original projector instance, typically an object of class bi_projector or any other class that implements a partial_projector method
colind	A numeric vector of column indices to select in the projection matrix. These indices correspond to the variables used for the partial projector
...	Additional arguments passed to the underlying partial_projector method

Value

A new projector instance, with the same class as the original object, that is restricted to the specified subset of input columns

See Also

bi_projector for an example of a class that implements a partial_projector method

Examples

```
# Example with the bi_projector class
X <- matrix(rnorm(10*20), 10, 20)
svdfit <- svd(X)
p <- bi_projector(svdfit$v, s = svdfit$u %*% diag(svdfit$d), sdev=svdfit$d)

# Create a partial projector using only the first 10 variables
colind <- 1:10
partial_p <- partial_projector(p, colind)
```

---

pass *a no-op pre-processing step*

Description

pass simply passes its data through the chain

Usage

```
pass(preproc = prepper())
```

Arguments

preproc	the pre-processing pipeline

Value

a prepper list

---

pca *Principal Components Analysis (PCA)*

Description

Compute the directions of maximal variance in a data matrix using the Singular Value Decomposition (SVD).

Usage

```
pca(
  X,
  ncomp = min(dim(X)),
  preproc = center(),
  method = c("fast", "base", "irlba", "propack", "rsvd", "svds"),
  ...
)
```

Arguments

X	The data matrix.
ncomp	The number of requested components to estimate (default is the minimum dimension of the data matrix).
preproc	The pre-processing function to apply to the data matrix (default is centering).
method	The SVD method to use, passed to svd_wrapper (default is "fast").
...	Extra arguments to send to svd_wrapper.

Value

A bi_projector object containing the PCA results.

See Also

svd_wrapper for details on SVD methods.

Examples

```
data(iris)
X <- as.matrix(iris[, 1:4])
res <- pca(X, ncomp = 4)
tres <- truncate(res, 3)
```

---

pca_outliers *PCA Outlier Diagnostics*

Description

Calculates Hotelling T^2 (score distance) and Q-residual (orthogonal distance) for each observation, given a chosen number of components.

Usage

```
pca_outliers(x, X, ncomp, cutoff = FALSE)
```

Arguments

x	A pca object.
X	The original data matrix used for PCA.
ncomp	Number of components to consider.
cutoff	Logical or numeric specifying threshold for labeling outliers. If TRUE, uses some typical statistical threshold (F-dist) for T^2, or sets an arbitrary Q limit. If numeric, treat it as a cutoff. Default is FALSE (no labeling).

Value

A data frame with columns T2 and Q, and optionally an outlier flag.

perm_ci *Permutation Confidence Intervals*

Description

Estimate confidence intervals for model parameters using permutation testing.

Usage

```
perm_ci(x, X, nperm, ...)

## S3 method for class 'pca'
perm_ci(x, X, nperm = 100, k = 4, distr = "gamma", parallel = FALSE, ...)
```

Arguments

x	A model fit object.
X	The original data matrix used to fit the model.
nperm	The number of permutations to perform for the confidence interval estimation.
...	Additional arguments to be passed to the specific model implementation of perm_ci.
k	Number of components to test (default 4).
distr	Distribution assumption (default "gamma"); currently ignored in forwarding.
parallel	Logical; if TRUE, use parallel processing.

Value

A list containing the estimated lower and upper bounds of the confidence intervals for model parameters.

perm_test *Generic Permutation-Based Test*

Description

This generic function implements a permutation-based test to assess the significance of components or statistics in a fitted model. The actual procedure depends on the method defined for the specific model class. Typical usage:

Arguments

x	A fitted model object (e.g. pca, cross_projector, discriminant_projector, multiblock_biprojector).
...	Additional arguments passed down to shuffle_fun or measure_fun (if applicable). Note: For multiblock methods, Xlist, comps, alpha, and use_rspectra (for biprojector) are handled as direct named arguments, not via ....
X	(Used by pca, cross_projector, discriminant_projector) The original primary data matrix used to fit x. Ignored by the multiblock_biprojector method.
Y	(Used by cross_projector) The secondary data block (n x pY). Ignored by other methods.
Xlist	(Used by multiblock_biprojector [optional, default NULL] and multiblock_projector [required]) List of data blocks.
nperm	Integer number of permutations (Default: 1000 for PCA, 500 for multiblock methods, 100 otherwise).
measure_fun	(Optional; Used by pca, cross_projector, discriminant_projector, multiblock_projector) A function for computing the statistic(s) of interest. Ignored by multiblock_biprojector. Signature/default varies by method (see Details).
shuffle_fun	(Optional; Used by all methods) A function for permuting the data appropriately. Signature/default varies by method (see Details).
fit_fun	(Optional; Used by cross_projector, discriminant_projector) A function for re-fitting a new model. Ignored by PCA and multiblock methods. Signature/default varies by method (see Details).
stepwise	(Used by pca) Logical indicating if sequential testing (P3 projection) should be performed. Default TRUE. (The multiblock methods also perform sequential testing based on alpha and comps, but this argument is ignored). Ignored by other methods.
parallel	(Used by all methods) Logical; if TRUE, attempt parallel execution via future.apply::future_lapply.
alternative	(Used by all methods) Character string for the alternative hypothesis: "greater" (default), "less", or "two.sided".
alpha	(Used by pca, multiblock_biprojector, multiblock_projector) Significance level for sequential stopping rule (default 0.05). Passed directly as a named argument to these methods.
comps	(Used by pca, multiblock_biprojector, multiblock_projector) Maximum number of components to test sequentially (default 4). Passed directly as a named argument to these methods.
use_svd_solver	(Used by pca) Optional string specifying the SVD solver (default "fast").
use_rspectra	(Used by multiblock_biprojector) Logical indicating whether to use RSpectra for eigenvalue calculation (default TRUE). Passed directly as a named argument.
predict_method	(Used by discriminant_projector) Prediction method ("lda" or "euclid") used by the default measure function (default "lda").

Details

1. Shuffle or permute the data in a way that breaks the structure of interest (e.g., shuffle labels for supervised methods, shuffle columns/rows for unsupervised).
2. Re-fit or re-project the model on the permuted data. Depending on the class, this can be done via a `fit_fun` or a class-specific approach.
3. Measure the statistic of interest (e.g., variance explained, classification accuracy, canonical correlation).
4. Compare the distribution of permuted statistics to the observed statistic to compute an empirical p-value.

S3 methods define the specific defaults and required signatures for the functions involved in shuffling, fitting, and measuring.

This function provides a framework for permutation testing in various multivariate models. The specific implementation details, default functions, and relevant arguments vary by method.

PCA Method (`perm_test.pca`**):** Relevant arguments: `X`, `nperm`, `measure_fun`, `shuffle_fun`, `stepwise`, `parallel`, `alternative`, `alpha`, `comps`, `use_svd_solver`, `....` Assesses significance of variance explained by each PC (Vitale et al., 2017). Default statistic: F_a. Default shuffle: column-wise. Default uses P3 projection and sequential stopping with `alpha`.

Cross Projector Method (`perm_test.cross_projector`**):** Relevant arguments: `X`, `Y`, `nperm`, `measure_fun`, `shuffle_fun`, `fit_fun`, `parallel`, `alternative`, `....` Tests the X-Y relationship. Default statistic: `x2y.mse`. Default shuffle: rows of Y. Default fit: `stats::cancor`.

Discriminant Projector Method (`perm_test.discriminant_projector`**):** Relevant arguments: `X`, `nperm`, `measure_fun`, `shuffle_fun`, `fit_fun`, `predict_method`, `parallel`, `alternative`, `....` Tests class separation. Default statistic: prediction accuracy. Default shuffle: labels. Default fit: `MASS::lda`.

Multiblock Bi-Projector Method (`perm_test.multiblock_biprojector`**):** Relevant arguments: `Xlist` (optional), `nperm`, `shuffle_fun`, `parallel`, `alternative`, `alpha`, `comps`, `use_rspectra`, `....` Tests consensus using fixed internal statistic (eigenvalue) on scores for each component. The statistic is the leading eigenvalue of the covariance matrix of block scores for a given component (T^T, where T columns are scores of block b on component k). By default, it shuffles rows within each block independently (either from `Xlist` if provided via `...`, or using the internally stored scores). It performs sequential testing for components specified by `comps` using the stopping rule defined by `alpha` (both passed via `...`).

Multiblock Projector Method (`perm_test.multiblock_projector`**):** Relevant arguments: `Xlist` (required), `nperm`, `measure_fun`, `shuffle_fun`, `parallel`, `alternative`, `alpha`, `comps`, `....` Tests consensus using `measure_fun` (default: mean abs corr) on scores projected from `Xlist` using the original model `x`. Does not refit.

Value

The structure of the return value depends on the method:

`cross_projector` **and** `discriminant_projector`**:** Returns an object of class `perm_test`, a list containing: `statistic`, `perm_values`, `p.value`, `alternative`, `method`, `nperm`, `call`.

`pca`**,** `multiblock_biprojector`**, and** `multiblock_projector`**:** Returns an object inheriting from `perm_test` (classes `perm_test_pca`, `perm_test_multiblock`, or `perm_test` respectively for

multiblock_projector), a list containing: `component_results` (data frame with observed stat, pval, CIs per component), `perm_values` (matrix of permuted stats), `alpha` (if applicable), `alternative`, `method`, `nperm` (vector of successful permutations per component), `call`.

References

Buja, A., & Eyuboglu, N. (1992). Remarks on parallel analysis. *Multivariate Behavioral Research*, 27(4), 509-540. (Relevant for PCA permutation concepts)

Vitale, R., Westerhuis, J. A., Næs, T., Smilde, A. K., de Noord, O. E., & Ferrer, A. (2017). Selecting the number of factors in principal component analysis by permutation testing— Numerical and practical aspects. *Journal of Chemometrics*, 31(10), e2937. doi:10.1002/cem.2937 (Specific to `perm_test.pca`)

See Also

`pca`, `cross_projector`, `discriminant_projector`, `multiblock_biprojector`, `measure_interblock_transfer_error`

Examples

```
# PCA Example
data(iris)
X_iris <- as.matrix(iris[,1:4])
mod_pca <- pca(X_iris, ncomp=4, preproc=center()) # Ensure centering

# Test first 3 components sequentially (faster with more nperm)
# Ensure a future plan is set for parallel=TRUE, e.g., future::plan("multisession")
res_pca <- perm_test(mod_pca, X_iris, nperm=50, comps=3, parallel=FALSE)
print(res_pca)

# PCA Example with row shuffling (tests different null hypothesis)
row_shuffle <- function(dat, ...) dat[sample(nrow(dat)), ]
res_pca_row <- perm_test(mod_pca, X_iris, nperm=50, comps=3,
                         shuffle_fun=row_shuffle, parallel=FALSE)
print(res_pca_row)
```

`perm_test.plsc` *Permutation test for PLSC latent variables*

Description

Uses row-wise permutation of the Y block to assess the significance of each latent variable (LV) in a fitted `plsc` model. The test statistic is the singular value of the cross-covariance matrix for each LV.

Usage

```
## S3 method for class 'plsc'
perm_test(
  x,
  X,
  Y,
  nperm = 1000,
  comps = ncomp(x),
  stepwise = TRUE,
  shuffle_fun = NULL,
  parallel = FALSE,
  alternative = c("greater", "less", "two.sided"),
  alpha = 0.05,
  ...
)
```

Arguments

x	A fitted plsc model object.
X	Original X block used to fit x.
Y	Original Y block used to fit x.
nperm	Number of permutations to perform (default 1000).
comps	Number of components (LVs) to test. Defaults to ncomp(x).
stepwise	Logical; if TRUE (default), perform sequential testing with deflation.
shuffle_fun	Optional function to permute Y; defaults to shuffling rows.
parallel	Logical; if TRUE, use parallel processing via future.apply.
alternative	Character string for the alternative hypothesis: "greater" (default), "less", or "two.sided".
alpha	Significance level used to report n_significant; not used directly in p-value calculation.
...	Additional arguments (currently unused).

---

plsc *Partial Least Squares Correlation (PLSC)*

Description

Reference implementation of symmetric brain-behavior PLS (a.k.a. Behavior PLSC). It finds paired weight vectors for X and Y that maximize their cross-block covariance, obtained from the SVD of the cross-covariance (or correlation) matrix $C_{XY} = X^\top Y/(n-1)$.

Usage

```
plsc(
  X,
  Y,
  ncomp = NULL,
  preproc_x = standardize(),
  preproc_y = standardize(),
  ...
)
```

Arguments

X	Numeric matrix of predictors (n x p_x).
Y	Numeric matrix of outcomes/behaviors (n x p_y). Must have the same number of rows as X.
ncomp	Number of latent variables to return. Defaults to min(nrow(X), ncol(X), ncol(Y)).
preproc_x	Preprocessor for the X block (default: standardize()). Use center() if you want covariance-based PLSC instead of correlation.
preproc_y	Preprocessor for the Y block (default: standardize()).
...	Extra arguments stored on the returned object.

Value

A cross_projector with class "plsc" containing

- vx, vy: X and Y loading/weight matrices.
- sx, sy: subject scores for X and Y blocks.
- singvals: singular values of C_{XY} (strength of each LV).
- explained_cov: proportion of cross-block covariance per LV.
- preproc_x, preproc_y: fitted preprocessors for reuse.

Examples

```
set.seed(1)
X <- matrix(rnorm(80), 20, 4)
Y <- matrix(rnorm(60), 20, 3)
fit <- plsc(X, Y, ncomp = 3)
fit$singvals
```

predict.classifier *Predict Class Labels using a Classifier Object*

Description

Predicts class labels and probabilities for new data using a fitted `classifier` object. It performs k-Nearest Neighbors (k-NN) classification in the projected component space.

Usage

```
## S3 method for class 'classifier'
predict(
  object,
  new_data,
  ncomp = NULL,
  colind = NULL,
  metric = c("euclidean", "cosine", "ejaccard"),
  normalize_probs = FALSE,
  prob_type = c("knn_proportion", "avg_similarity"),
  ...
)
```

Arguments

`object`	A fitted object of class `classifier`.
`new_data`	A numeric matrix or vector of new observations to classify. Rows are observations, columns are variables matching the original data space used by the projector OR matching `colind` if provided.
`ncomp`	Optional integer; the number of components to use from the projector for classification (default: all components used during classifier creation).
`colind`	Optional numeric vector specifying column indices from the original data space. If provided, `new_data` is projected using only these features (`partial_project`). This overrides any `colind` stored default in the `object`. The resulting projection is compared against the reference scores (`object$scores`) stored in the classifier.
`metric`	Character string specifying the similarity or distance metric for k-NN. Choices: "euclidean", "cosine", "ejaccard".
`normalize_probs`	Logical; **DEPRECATED** Normalization behavior is now implicit in `prob_type="avg_similarity"`.
`prob_type`	Character string; method for calculating probabilities: • "knn_proportion" (default): Calculates the proportion of each class among the k nearest neighbors. • "avg_similarity": Calculates average similarity to all training points per class (uses `avg_probs` helper).

`...` Extra arguments passed down to projection methods (`project`, `partial_project`) or potentially to distance/similarity calculations (e.g., for `proxy::simil` if used with `ejaccard`).

Details

The function first projects the `new_data` into the component space defined by the classifier's internal projector. If `colind` is specified, a partial projection using only those features is performed. This projection is then compared to the reference scores stored within the `classifier` object (`object$scores`) using the specified `metric`. The k-NN algorithm identifies the `k` nearest reference samples (based on similarity or distance) and predicts the class via majority vote. Probabilities are estimated based on the average similarity/distance to each class among the neighbors or all reference points.

Value

A list containing:

`class` A factor vector of predicted class labels for `new_data`.

`prob` A numeric matrix (rows corresponding to `new_data`, columns to classes) of estimated class probabilities.

See Also

`classifier.projector`, `classifier.multiblock_biprojector`, `partial_project`

Other classifier predict: `predict.rf_classifier()`

Examples

```
# Assume clf is a fitted classifier object (e.g., from classifier.projector)
# Assume new_dat is a matrix of new observations
# preds <- predict(clf, new_data = new_dat, metric = "cosine")
# print(preds$class)
# print(preds$prob)
```

---

predict.discriminant_projector

Predict method for a discriminant_projector, supporting LDA or Euclid

---

Description

This produces class predictions or posterior-like scores for new data. We first project the data into the subspace defined by `x$v`, then either:

1. **LDA approach** (`method="lda"`), which uses a (simplified) linear discriminant formula or distance to class means in the subspace combined with prior probabilities.

2. **Euclid approach** (`method="euclid"`), which uses plain Euclidean distance to each class mean in the subspace.

We return either a `type="class"` label or `type="prob"` posterior-like matrix.

Usage

```
## S3 method for class 'discriminant_projector'
predict(
  object,
  new_data,
  method = c("lda", "euclid"),
  type = c("class", "prob"),
  colind = NULL,
  ...
)
```

Arguments

`object`	A `discriminant_projector` object.
`new_data`	A numeric matrix (or vector) with the same # of columns as the original data (unless partial usage). Rows=observations, columns=features.
`method`	Either `"lda"` (the default) or `"euclid"` (nearest-mean).
`type`	`"class"` (default) for predicted class labels, or `"prob"` for posterior-like probabilities.
`colind`	(optional) if partial columns are used, specify which columns map to the subspace. If `NULL`, assume full columns.
`...`	further arguments (not used or for future expansions).

Value

If `type="class"`, a factor vector of length n (predicted classes). If `type="prob"`, an (n x #classes) numeric matrix of posterior-like values, with row names matching `new_data` if available.

Predict method for a discriminant_projector

This produces class predictions or posterior-like scores for new data, based on:

- **LDA approach** (`method="lda"`), which uses a linear discriminant formula with a pooled covariance matrix if `x\$Sigma` is given, or the identity matrix if `Sigma=NULL`. If that covariance matrix is not invertible, a pseudo-inverse is used and a warning is emitted.
- **Euclid approach** (`method="euclid"`), which uses plain Euclidean distance to each class mean in the subspace.

We return either a `type="class"` label or `type="prob"` posterior-like matrix.

If `type="class"`, a factor vector of length n (predicted classes). If `type="prob"`, an (n x #classes) numeric matrix of posterior-like values.

`predict.rf_classifier` *Predict Class Labels using a Random Forest Classifier Object*

Description

Predicts class labels and probabilities for new data using a fitted `rf_classifier` object. This method projects the `new_data` into the component space and then uses the stored `randomForest` model to predict outcomes.

Usage

```
## S3 method for class 'rf_classifier'
predict(object, new_data, ncomp = NULL, colind = NULL, ...)
```

Arguments

`object`	A fitted object of class `rf_classifier`.
`new_data`	A numeric matrix or vector of new observations to classify. Rows are observations, columns are variables matching the original data space used by the projector OR matching `colind` if provided.
`ncomp`	Optional integer; the number of components to use from the projector for classification (default: all components used during classifier creation).
`colind`	Optional numeric vector specifying column indices from the original data space. If provided, `new_data` is projected using only these features (`partial_project`). This overrides any `colind` stored default in the `object`. The resulting projection is compared against the reference scores (`object$scores`) stored in the classifier.
`...`	Extra arguments passed to `predict.randomForest`.

Value

A list containing:

`class`	Predicted class labels (typically factor) from the random forest model.
`prob`	A numeric matrix of predicted class probabilities from the random forest model.

See Also

`rf_classifier.projector`, `predict.randomForest`

Other classifier predict: `predict.classifier()`

prep *prepare a dataset by applying a pre-processing pipeline*

Description

prepare a dataset by applying a pre-processing pipeline

Usage

```
prep(x, ...)
```

Arguments

x	the pipeline
...	extra args

Value

the pre-processed data

preprocess *Convenience function for preprocessing workflow*

Description

This helper function provides a simple interface for the common preprocessing workflow: fit a preprocessor to data and return both the fitted preprocessor and the transformed data.

Usage

```
preprocess(preproc, X, ...)
```

Arguments

preproc	A preprocessing object (e.g., created with `center()`, `standardize()`, etc.)
X	A matrix or data frame to preprocess
...	Additional arguments passed to methods

Value

A list with two elements:

preproc	The fitted preprocessing object
transformed	The transformed data matrix

See Also

fit(), fit_transform(), transform(), inverse_transform()

Examples

```
# Simple preprocessing workflow
X <- matrix(rnorm(100), 10, 10)
result <- preprocess(center(), X)
fitted_preproc <- result$preproc
X_centered <- result$transformed

# Equivalent to:
# fitted_preproc <- fit(center(), X)
# X_centered <- transform(fitted_preproc, X)
```

---

prinang *Calculate Principal Angles Between Subspaces*

Description

Computes the principal angles between two subspaces defined by the columns of two orthonormal matrices Q1 and Q2.

Usage

```
prinang(Q1, Q2)
```

Arguments

Q1	An n x p matrix whose columns form an orthonormal basis for the first subspace.
Q2	An n x q matrix whose columns form an orthonormal basis for the second subspace.

Value

A numeric vector containing the principal angles in radians, sorted in ascending order. The number of angles is min(p, q).

Examples

```
# Example: Angle between xy-plane and a plane rotated 45 degrees around x-axis
Q1 <- cbind(c(1,0,0), c(0,1,0)) # xy-plane basis
theta <- pi/4
R <- matrix(c(1, 0, 0, 0, cos(theta), sin(theta), 0, -sin(theta), cos(theta)), 3, 3)
Q2 <- R %*% Q1 # Rotated basis
angles_rad <- prinang(Q1, Q2)
angles_deg <- angles_rad * 180 / pi
print(angles_deg) # Should be approximately 0 and 45 degrees
```

```
# Example with PCA loadings (after ensuring orthonormality if needed)
# Assuming pca1$v and pca2$v are loading matrices (variables x components)
# Orthonormalize them first if they are not already (e.g., from standard SVD)
# Q1 <- qr.Q(qr(pca1$v[, 1:3]))
# Q2 <- qr.Q(qr(pca2$v[, 1:3]))
# prinang(Q1, Q2)
```

`principal_angles` *Principal angles (two sub-spaces)*

Description

Principal angles (two sub-spaces)

Usage

```
principal_angles(fit1, fit2, k = NULL)
```

Arguments

`fit1`, `fit2`	bi_projector objects (or any object with $v loadings)
`k`	number of dimensions to compare (default: min(ncomp))

Value

numeric vector of principal angles (radians, length = k)

`print.bi_projector` *Pretty Print S3 Method for bi_projector Class*

Description

Pretty Print S3 Method for bi_projector Class

Usage

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

Arguments

`x`	A `bi_projector` object
`...`	Additional arguments passed to the print function

Value

Invisible `bi_projector` object

print.classifier *Pretty Print Method for* classifier *Objects*

Description

Display a human-readable summary of a classifier object.

Usage

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

Arguments

x	A classifier object.
...	Additional arguments.

Value

classifier object.

Examples

```
# Assume clf is a fitted classifier object
# print(clf)
```

print.concat_pre_processor *Print a concat_pre_processor object*

Description

Print a concat_pre_processor object

Usage

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

Arguments

x	A concat_pre_processor object.
...	Additional arguments (ignored).

print.multiblock_biprojector *Pretty Print Method for* multiblock_biprojector *Objects*

Description

Display a summary of a multiblock_biprojector object.

Usage

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

Arguments

x	A multiblock_biprojector object.
...	Additional arguments passed to print().

Value

Invisible multiblock_biprojector object.

print.pca *Print Method for PCA Objects*

Description

Provide a color-enhanced summary of the PCA object, including dimensions, variance explained, and a quick component breakdown.

Usage

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

Arguments

x	A pca object.
...	Ignored (for compatibility).

print.perm_test *Print Method for perm_test Objects*

Description

Provides a concise summary of the permutation test results.

Usage

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

Arguments

x	An object of class perm_test.
...	Additional arguments passed to printing methods.

Value

Invisibly returns the input object x.

print.perm_test_pca *Print Method for perm_test_pca Objects*

Description

Provides a concise summary of the PCA permutation test results.

Usage

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

Arguments

x	An object of class perm_test_pca.
...	Additional arguments passed to printing methods.

Value

Invisibly returns the input object x.

print.prepper *Print a prepper pipeline*

Description

Uses crayon to produce a colorful and readable representation of the pipeline steps.

Usage

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

Arguments

x	A prepper object.
...	Additional arguments (ignored).

print.pre_processor *Print a pre_processor object*

Description

Display information about a pre_processor using crayon-based formatting.

Usage

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

Arguments

x	A pre_processor object.
...	Additional arguments (ignored).

print.regress *Pretty Print Method for* regress *Objects*

Description

Display a human-readable summary of a regress object using crayon formatting, including information about the method and dimensions.

Usage

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

Arguments

x	A regress object (a bi_projector with regression info).
...	Additional arguments passed to print().

print.rf_classifier *Pretty Print Method for* rf_classifier *Objects*

Description

Display a human-readable summary of an rf_classifier object.

Usage

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

Arguments

x	An rf_classifier object.
...	Additional arguments passed to print.randomForest.

Value

rf_classifier object.

Examples

```
# Assume rf_clf is a fitted rf_classifier object
# print(rf_clf)
```

`project` *New sample projection*

Description

Project one or more samples onto a subspace. This function takes a model fit and new observations, and projects them onto the subspace defined by the model. This allows for the transformation of new data into the same lower-dimensional space as the original data.

Usage

```
project(x, new_data, ...)
```

Arguments

`x`	The model fit, typically an object of class bi_projector or any other class that implements a project method
`new_data`	A matrix or vector of new observations with the same number of columns as the original data. Rows represent observations and columns represent variables
`...`	Extra arguments to be passed to the specific project method for the object's class

Value

A matrix or vector of the projected observations, where rows represent observations and columns represent the lower-dimensional space

See Also

`bi_projector` for an example of a class that implements a project method

Other project: `project.cross_projector()`, `project_block()`, `project_vars()`

Examples

```
# Example with the bi_projector class
X <- matrix(rnorm(10*20), 10, 20)
svdfit <- svd(X)
p <- bi_projector(svdfit$v, s = svdfit$u %*% diag(svdfit$d), sdev=svdfit$d)

# Project new_data onto the same subspace as the original data
new_data <- matrix(rnorm(5*20), 5, 20)
projected_data <- project(p, new_data)
```

project.cross_projector

project a cross_projector instance

Description

project a cross_projector instance

Usage

```
## S3 method for class 'cross_projector'
project(x, new_data, source = c("X", "Y"), ...)
```

Arguments

x	The model fit, typically an object of class bi_projector or any other class that implements a project method
new_data	A matrix or vector of new observations with the same number of columns as the original data. Rows represent observations and columns represent variables
source	the source of the data (X or Y block)
...	Extra arguments to be passed to the specific project method for the object's class

Value

the projected data

See Also

Other project: `project()`, `project_block()`, `project_vars()`

project.nystrom_approx

Project new data using a Nyström approximation model

Description

Project new data using a Nyström approximation model

Usage

```
## S3 method for class 'nystrom_approx'
project(x, new_data, ...)
```

Arguments

x	A nystrom_approx object (inheriting from bi_projector).
new_data	New data matrix to project.
...	Additional arguments (currently ignored).

Value

A matrix of projected scores.

---

projector *Construct a* projector *instance*

Description

A projector maps a matrix from an N-dimensional space to d-dimensional space, where d may be less than N. The projection matrix, v, is not necessarily orthogonal. This function constructs a projector instance which can be used for various dimensionality reduction techniques like PCA, LDA, etc.

Usage

```
projector(v, preproc = prep(pass()), ..., classes = NULL)
```

Arguments

v	A matrix of coefficients with dimensions nrow(v) by ncol(v) (columns = components)
preproc	A prepped pre-processing object (S3 class pre_processor). Default is the no-op pass() preprocessor.
...	Extra arguments to be stored in the projector object.
classes	Additional class information used for creating subtypes of projector. Default is NULL.

Value

An instance of type projector.

`project_block` *Project a single "block" of data onto the subspace*

Description

When observations are concatenated into "blocks", it may be useful to project one block from the set. This function facilitates the projection of a specific block of data onto a subspace. It is a convenience method for multi-block fits and is equivalent to a "partial projection" where the column indices are associated with a given block.

Usage

```
project_block(x, new_data, block, least_squares, ...)
```

Arguments

`x`	The model fit, typically an object of a class that implements a `project_block` method
`new_data`	A matrix or vector of new observation(s) with the same number of columns as the original data
`block`	An integer representing the block ID to select in the block projection matrix. This ID corresponds to the specific block of data to be projected
`least_squares`	Logical. If `TRUE` use least squares projection.
`...`	Additional arguments passed to the underlying `project_block` method

Value

A matrix or vector of the projected data for the specified block

See Also

`project` for the generic projection function

Other project: `project()`, `project.cross_projector()`, `project_vars()`

`project_block.multiblock_projector`
Project Data onto a Specific Block

Description

Projects the new data onto the subspace defined by a specific block of variables.

Usage

```
## S3 method for class 'multiblock_projector'
project_block(x, new_data, block, least_squares = TRUE, ...)
```

Arguments

x	A multiblock_projector object.
new_data	The new data to be projected.
block	The block index (1-based) to project onto.
least_squares	Logical. If TRUE (default), use least squares projection.
...	Additional arguments passed to partial_project.

Value

The projected scores for the specified block.

---

project_vars *Project one or more variables onto a subspace*

Description

This function projects one or more variables onto a subspace. It is often called supplementary variable projection and can be computed for a biorthogonal decomposition, such as Singular Value Decomposition (SVD).

Usage

```
project_vars(x, new_data, ...)
```

Arguments

x	The model fit, typically an object of a class that implements a project_vars method
new_data	A matrix or vector of new observation(s) with the same number of rows as the original data
...	Additional arguments passed to the underlying project_vars method

Value

A matrix or vector of the projected variables in the subspace

See Also

project for the generic projection function for samples

Other project: project(), project.cross_projector(), project_block()

rank_score *Calculate Rank Score for Predictions*

Description

Computes the rank score (normalized rank of the true class probability) for each observation. Lower rank scores indicate better predictions (true class has higher probability).

Usage

```
rank_score(prob, observed)
```

Arguments

prob	Numeric matrix of predicted probabilities (observations x classes). Column names must correspond to class labels.
observed	Factor or vector of observed class labels. Must be present in `colnames(prob)`.

Value

A `data.frame` with columns `prank` (the normalized rank score) and `observed` (the input labels).

See Also

Other classifier evaluation: `topk()`

Examples

```
probs <- matrix(c(0.1, 0.9, 0.8, 0.2), 2, 2, byrow=TRUE,
                dimnames = list(NULL, c("A", "B")))
obs <- factor(c("B", "A"))
rank_score(probs, obs)
```

reconstruct *Reconstruct the data*

Description

Reconstruct a data set from its (possibly) low-rank representation. This can be useful when analyzing the impact of dimensionality reduction or when visualizing approximations of the original data.

Usage

```
reconstruct(x, ...)
```

Arguments

x	The model fit, typically an object of a class that implements a `reconstruct` method
...	Additional arguments passed to specific methods. Common parameters include: `comp` A vector of component indices to use in the reconstruction `rowind` The row indices to reconstruct (optional) `colind` The column indices to reconstruct (optional) `scores` (For `composed_projector` only) A numeric matrix of scores to reconstruct from

Value

A reconstructed data set based on the selected components, rows, and columns

See Also

`bi_projector` for an example of a two-way mapping model that can be reconstructed

Other reconstruct: `reconstruct_new()`

`reconstruct.composed_projector` *Reconstruct Data from Scores using a Composed Projector*

Description

Maps scores from the final latent space back towards the original input space using the composed projector's combined inverse projection. Requires scores to be provided explicitly.

Usage

```
## S3 method for class 'composed_projector'
reconstruct(x, scores, comp = NULL, rowind = NULL, colind = NULL, ...)
```

Arguments

x	A `composed_projector` object.
scores	A numeric matrix of scores (observations x components) in the final latent space of the composed projector.
comp	Numeric vector of component indices (columns of `scores`, rows of `inverse_projection`) to use for reconstruction. Defaults to all components.
rowind	Numeric vector of row indices (observations in `scores`) to reconstruct. Defaults to all rows.
colind	Numeric vector of original variable indices (columns of the final reconstructed matrix) to return. Defaults to all original variables.
...	Additional arguments (currently unused).

Details

Attempts to apply the `reverse_transform` of the *first* stage's preprocessor to return data in the original units. If the first stage preprocessor is unavailable or invalid, a warning is issued, and data is returned in the (potentially) preprocessed space of the first stage.

Value

A matrix representing the reconstructed data, ideally in the original data space.

`reconstruct.pca` *Reconstruct Data from PCA Results*

Description

Reconstructs the original (centered) data matrix from the PCA scores and loadings.

Usage

```
## S3 method for class 'pca'
reconstruct(x, comp = 1:ncomp(x), ...)
```

Arguments

`x`	A `pca` object.
`comp`	Integer vector specifying which components to use for reconstruction (default: all components in `x`).
`...`	Extra arguments (ignored).

Value

A matrix representing the reconstructed data in the *original* scale (preprocessing reversed).

`reconstruct.regress` *Reconstruct fitted or subsetted outputs for a* `regress` *object*

Description

For regression-based bi_projectors, reconstruction should map from the design matrix side (scores) to the output space using the regression coefficients, without applying any reverse preprocessing (which belongs to the input/basis side).

Usage

```
## S3 method for class 'regress'
reconstruct(
  x,
  comp = 1:ncol(x$coefficients),
  rowind = 1:nrow(scores(x)),
  colind = 1:nrow(x$coefficients),
  ...
)
```

Arguments

x	A regress object produced by regress().
comp	Integer vector of component indices (columns of the design matrix / predictors) to use.
rowind	Integer vector of row indices in the design matrix (observations) to reconstruct.
colind	Integer vector of output indices (columns of Y) to reconstruct.
...	Ignored.

---

reconstruct_new *Reconstruct new data in a model's subspace*

Description

This function takes a model (e.g., projector or bi_projector) and a new dataset, and computes the rank-d approximation of the new data in the same subspace that was defined by the model. In other words, we **project** the new data into the fitted subspace and then **map it back** to the original dimensionality.

Usage

```
reconstruct_new(x, new_data, ...)
```

Arguments

x	The fitted model object (e.g., bi_projector) that defines a subspace or factorization.
new_data	A numeric matrix (or data frame) of shape (n x p_full) or possibly fewer columns if you allow partial reconstruction.
...	Additional arguments passed to the specific reconstruct_new method for the class of x.

Details

Similar to reconstruct but operates on an external new_data rather than the original fitted data. Often used to see how well the model's subspace explains unseen data.

Value

A numeric matrix (same number of rows as `new_data`, and typically the same number of columns if you're reconstructing fully) representing the rank-d approximation in the model's subspace.

See Also

`reconstruct` for reconstructing the original data in the model.

Other reconstruct: `reconstruct()`

`refit` *refit a model*

Description

refit a model given new data or new parameter(s)

Usage

```
refit(x, new_data, ...)
```

Arguments

`x`	the original model fit object
`new_data`	the new data to process
`...`	extra args

Value

a refit model object

`regress` *Multi-output linear regression*

Description

Fit a multivariate regression model for a matrix of basis functions, `X`, and a response matrix `Y`. The goal is to find a projection matrix that can be used for mapping and reconstruction.

Usage

```
regress(
  X,
  Y,
  preproc = pass(),
  method = c("lm", "enet", "mridge", "pls"),
  intercept = FALSE,
  lambda = 0.001,
  alpha = 0,
  ncomp = ceiling(ncol(X)/2),
  ...
)
```

Arguments

X	the set of independent (basis) variables
Y	the response matrix
preproc	A preprocessing pipeline applied to X before fitting the model
method	the regression method: lm, enet, mridge, or pls
intercept	whether to include an intercept term
lambda	ridge shrinkage parameter (for methods mridge and enet)
alpha	the elastic net mixing parameter if method is enet
ncomp	number of PLS components if method is pls
...	extra arguments sent to the underlying fitting function

Value

a bi-projector of type regress. The sdev component of this object stores the standard deviations of the columns of the design matrix (X potentially including an intercept) used in the fit, not the standard deviations of latent components as might be typical in other bi_projector contexts (e.g., SVD).

Examples

```
# Generate synthetic data
set.seed(123) # for reproducibility
Y <- matrix(rnorm(10 * 100), 10, 100)
X <- matrix(rnorm(10 * 9), 10, 9)

# Fit regression models and reconstruct the fitted response matrix
r_lm <- regress(X, Y, intercept = FALSE, method = "lm")
recon_lm <- reconstruct(r_lm) # Reconstructs fitted Y

r_mridge <- regress(X, Y, intercept = TRUE, method = "mridge", lambda = 0.001)
recon_mridge <- reconstruct(r_mridge)

r_enet <- regress(X, Y, intercept = TRUE, method = "enet", lambda = 0.001, alpha = 0.5)
```

```
recon_enet <- reconstruct(r_enet)

r_pls <- regress(X, Y, intercept = TRUE, method = "pls", ncomp = 5)
recon_pls <- reconstruct(r_pls)
```

`reprocess` *apply pre-processing parameters to a new data matrix*

Description

Given a new dataset, process it in the same way the original data was processed (e.g. centering, scaling, etc.)

Usage

```
reprocess(x, new_data, colind, ...)
```

Arguments

`x`	the model fit object
`new_data`	the new data to process
`colind`	the column indices of the new data
`...`	extra args

Value

the reprocessed data

`reprocess.cross_projector`
reprocess a cross_projector instance

Description

reprocess a cross_projector instance

Usage

```
## S3 method for class 'cross_projector'
reprocess(x, new_data, colind = NULL, source = c("X", "Y"), ...)
```

Arguments

x	the model fit object
new_data	the new data to process
colind	the column indices of the new data
source	the source of the data (X or Y block)
...	extra args

Details

When colind is provided, each index is validated to be within the available coefficient rows using chk::chk_subset.

Value

the re(pre-)processed data

---

reprocess.nystrom_approx

Reprocess data for Nyström approximation

Description

Apply preprocessing to new data for projection using a Nyström approximation. This method overrides the default reprocess.projector to handle the fact that Nyström components are in kernel space (not feature space).

Usage

```
## S3 method for class 'nystrom_approx'
reprocess(x, new_data, colind = NULL, ...)
```

Arguments

x	A nystrom_approx object
new_data	A matrix with the same number of columns as the original training data
colind	Optional column indices (not typically used for Nyström)
...	Additional arguments (ignored)

Value

Preprocessed data matrix

residualize *Compute a regression model for each column in a matrix and return residual matrix*

Description

Compute a regression model for each column in a matrix and return residual matrix

Usage

```
residualize(form, X, design, intercept = FALSE)
```

Arguments

form	the formula defining the model to fit for residuals
X	the response matrix
design	the data.frame containing the design variables specified in form argument.
intercept	add an intercept term (default is FALSE)

Value

a matrix of residuals

Examples

```
X <- matrix(rnorm(20*10), 20, 10)
des <- data.frame(a=rep(letters[1:4], 5), b=factor(rep(1:5, each=4)))
xresid <- residualize(~ a+b, X, design=des)

## design is saturated, residuals should be zero
xresid2 <- residualize(~ a*b, X, design=des)
sum(xresid2) == 0
```

residuals *Obtain residuals of a component model fit*

Description

Calculate the residuals of a model after removing the effect of the first ncomp components. This function is useful to assess the quality of the fit or to identify patterns that are not captured by the model.

Usage

```
residuals(x, ncomp, xorig, ...)
```

Arguments

x	The model fit object.
ncomp	The number of components to factor out before calculating residuals.
xorig	The original data matrix (X) used to fit the model.
...	Additional arguments passed to the method.

Value

A matrix of residuals, with the same dimensions as the original data matrix.

---

reverse_transform *reverse a pre-processing transform*

Description

reverse a pre-processing transform

Usage

```
reverse_transform(x, X, colind, ...)
```

Arguments

x	the pre_processor
X	the data matrix
colind	column indices
...	extra args

Value

the reverse-transformed data

rf_classifier *construct a random forest wrapper classifier*

Description

Given a model object (e.g. `projector` construct a random forest classifier that can generate predictions for new data points.

Usage

```
rf_classifier(x, colind, ...)
```

Arguments

`x`	the model object
`colind`	the (optional) column indices used for prediction
`...`	extra arguments to `randomForest` function

Value

a random forest classifier

rf_classifier.projector *Create a random forest classifier*

Description

Uses `randomForest` to train a random forest on the provided scores and labels.

Usage

```
## S3 method for class 'projector'
rf_classifier(x, colind = NULL, labels, scores, ...)
```

Arguments

`x`	a projector object
`colind`	optional col indices
`labels`	class labels
`scores`	reference scores
`...`	passed to `randomForest`

Value

a `rf_classifier` object with rfres (rf model), labels, scores

See Also

`randomForest`

Other classifier: `classifier()`, `classifier.multiblock_biprojector()`

Examples

```
# Assume proj is a fitted projector object
# Assume lbls are labels and sc are scores
# if (requireNamespace("randomForest", quietly = TRUE)) {
#   rf_classifier(proj, labels = lbls, scores = sc)
# }
```

`rotate` *Rotate a Component Solution*

Description

Perform a rotation of the component loadings to improve interpretability.

Usage

```
rotate(x, ncomp, type, ...)
```

Arguments

`x`	The model fit, typically a result from a dimensionality reduction method like PCA.
`ncomp`	The number of components to rotate.
`type`	The type of rotation to apply (e.g., "varimax", "quartimax", "promax").
`...`	extra args

Value

A modified model fit with the rotated components.

`rotate.pca` *Rotate PCA Loadings*

Description

Apply a specified rotation to the component loadings of a PCA model. This function leverages the GPArotation package to apply orthogonal or oblique rotations.

Usage

```
## S3 method for class 'pca'
rotate(
  x,
  ncomp,
  type = c("varimax", "quartimax", "promax"),
  loadings_type = c("pattern", "structure"),
  score_method = c("auto", "recompute", "original"),
  ...
)
```

Arguments

`x`	A PCA model object, typically created using the `pca()` function.
`ncomp`	The number of components to rotate. Must be <= ncomp(x).
`type`	The type of rotation to apply. Supported rotation types: **"varimax"** Orthogonal Varimax rotation **"quartimax"** Orthogonal Quartimax rotation **"promax"** Oblique Promax rotation
`loadings_type`	For oblique rotations, which loadings to use: **"pattern"** Use pattern loadings as `v` **"structure"** Use structure loadings (`pattern_loadings %*% Phi`) as `v` Ignored for orthogonal rotations.
`score_method`	How to recompute scores after rotation: **"auto"** For orthogonal rotations, use `scores_new = scores_original %*% t(R)` For oblique rotations, recompute from the pseudoinverse. **"recompute"** Always recompute scores from `X_proc` and the pseudoinverse of rotated loadings. **"original"** For orth rotations, same as `auto`, but may not work for oblique rotations.
`...`	Additional arguments passed to GPArotation functions.

Value

A modified PCA object with class `rotated_pca` and additional fields:

v Rotated loadings

s Rotated scores

sdev Updated standard deviations of rotated components

explained_variance Proportion of explained variance for each rotated component

rotation A list with rotation details: `type`, `R` (orth) or `Phi` (oblique), and `loadings_type`

Examples

```
# Perform PCA on the iris dataset
data(iris)
X <- as.matrix(iris[,1:4])
res <- pca(X, ncomp=4)

# Apply varimax rotation to the first 3 components
rotated_res <- rotate(res, ncomp=3, type="varimax")
```

---

scores *Retrieve the component scores*

Description

Extract the factor score matrix from a fitted model. The factor scores represent the projections of the data onto the components, which can be used for further analysis or visualization.

Usage

```
scores(x, ...)
```

Arguments

`x`	The model fit object.
`...`	Additional arguments passed to the method.

Value

A matrix of factor scores, with rows corresponding to samples and columns to components.

See Also

`project` for projecting new data onto the components.

scores.plsc *Extract scores from a PLSC fit*

Description

Extract scores from a PLSC fit

Usage

```
## S3 method for class 'plsc'
scores(x, block = c("X", "Y"), ...)
```

Arguments

x	A plsc object.
block	Which block to return scores for: "X" (default) or "Y".
...	Ignored.

Value

Numeric matrix of scores for the chosen block.

screeplot *Screeplot for PCA*

Description

Displays the variance explained by each principal component as a bar or line plot.

Usage

```
screeplot(x, ...)
```

Arguments

x	A pca object.
...	extra args

screeplot.pca *Screeplot for PCA*

Description

Displays the variance explained by each principal component as a bar or line plot.

Usage

```
## S3 method for class 'pca'
screeplot(x, type = "barplot", main = "Screeplot", ...)
```

Arguments

x	A pca object.
type	"barplot" or "lines".
main	Plot title.
...	Additional args to pass to base R plotting.

sdev *standard deviations*

Description

The standard deviations of the projected data matrix

Usage

```
sdev(x)
```

Arguments

x	the model fit

Value

the standard deviations

shape *Shape of the Projector*

Description

Get the input/output shape of the projector.

Usage

```
shape(x, ...)
```

Arguments

x	The model fit.
...	Extra arguments.

Details

This function retrieves the dimensions of the sample loadings matrix v in the form of a vector with two elements. The first element is the number of rows in the v matrix, and the second element is the number of columns.

Value

A vector containing the dimensions of the sample loadings matrix v (number of rows and columns).

shape.cross_projector *shape of a cross_projector instance*

Description

shape of a cross_projector instance

Usage

```
## S3 method for class 'cross_projector'
shape(x, source = c("X", "Y"), ...)
```

Arguments

x	The model fit.
source	the source of the data (X or Y block)
...	Extra arguments.

Value

the shape of the data

standardize *center and scale each vector of a matrix*

Description

center and scale each vector of a matrix

Usage

```
standardize(preproc = prepper(), cmeans = NULL, sds = NULL)
```

Arguments

preproc	the pre-processing pipeline
cmeans	an optional vector of column means
sds	an optional vector of sds

Value

a prepper list

std_scores *Compute standardized component scores*

Description

Calculate standardized factor scores from a fitted model. Standardized scores are useful for comparing the contributions of different components on the same scale, which can help in interpreting the results.

Usage

```
std_scores(x, ...)
```

Arguments

x	The model fit object.
...	Additional arguments passed to the method.

Value

A matrix of standardized factor scores, with rows corresponding to samples and columns to components.

See Also

scores for retrieving the original component scores.

std_scores.svd *Calculate Standardized Scores for SVD results*

Description

Computes standardized scores from an SVD result performed by `svd_wrapper`. These scores are scaled to have approximately unit variance, assuming the original data used for SVD was centered. They differ from the `s` component of the `svd` object, which contains scores scaled by singular values.

Usage

```
## S3 method for class 'svd'
std_scores(x, ...)
```

Arguments

`x`	An object of class `svd`, typically from `svd_wrapper`.
`...`	Extra arguments (ignored).

Value

A matrix of standardized scores (N x k) with columns having variance close to 1.

subspace_similarity *Compute subspace similarity*

Description

Compute subspace similarity

Usage

```
subspace_similarity(
  fits,
  method = c("avg_pair", "grassmann", "worst_case"),
  ...
)
```

Arguments

`fits`	a list of bi_projector objects
`method`	the method to use for computing subspace similarity
`...`	additional arguments to pass to the method

Value

a numeric value representing the subspace similarity

summary.composed_projector *Summarize a Composed Projector*

Description

Provides a summary of the stages within a composed projector, including stage names, input/output dimensions, and the primary class of each stage.

Usage

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

Arguments

object	A composed_projector object.
...	Currently unused.

Value

A tibble summarizing the pipeline stages.

svd_wrapper *Singular Value Decomposition (SVD) Wrapper*

Description

Computes the singular value decomposition of a matrix using one of the specified methods. It is designed to be an easy-to-use wrapper for various SVD methods available in R.

Usage

```
svd_wrapper(
  X,
  ncomp = min(dim(X)),
  preproc = pass(),
  method = c("fast", "base", "irlba", "propack", "rsvd", "svds"),
  q = 2,
  p = 10,
  tol = .Machine$double.eps,
  ...
)
```

Arguments

X	the input matrix
ncomp	the number of components to estimate (default: min(dim(X)))
preproc	the pre-processor to apply on the input matrix (e.g., center(), standardize(), pass()) Can be a prepper object or a pre-processing function.
method	the SVD method to use: 'base', 'fast', 'irlba', 'propack', 'rsvd', or 'svds'
q	parameter passed to method rsvd (default: 2)
p	parameter passed to method rsvd (default: 10)
tol	minimum relative tolerance for dropping singular values (compared to the largest). Default: .Machine$double.eps.
...	extra arguments passed to the selected SVD function

Value

an SVD object that extends bi_projector

Examples

```
# Load iris dataset and select the first four columns
data(iris)
X <- as.matrix(iris[, 1:4])

# Compute SVD using the base method and 3 components
fit <- svd_wrapper(X, ncomp = 3, preproc = center(), method = "base")
```

---

topk *top-k accuracy indicator*

Description

Determines if the true class label is among the top k predicted probabilities for each observation.

Usage

```
topk(prob, observed, k)
```

Arguments

prob	Numeric matrix of predicted probabilities (observations x classes). Column names must correspond to class labels.
observed	Factor or vector of observed class labels. Must be present in colnames(prob).
k	Integer; the number of top probabilities to consider.

Value

A data.frame with columns topk (logical indicator: TRUE if observed class is in top-k) and observed.

See Also

Other classifier evaluation: rank_score()

Examples

```
probs <- matrix(c(0.1, 0.9, 0.8, 0.2, 0.3, 0.7), 3, 2, byrow=TRUE,
                dimnames = list(NULL, c("A", "B")))
obs <- factor(c("B", "A", "B"))
topk(probs, obs, k=1)
topk(probs, obs, k=2)
```

---

transfer *Transfer data from one domain/block to another via a latent space*

Description

Convert between data representations in a multiblock or cross-decomposition model by projecting the input new_data from the from domain/block onto a latent space and then reconstructing it in the to domain/block.

Usage

```
transfer(x, new_data, from, to, opts = list(), ...)
```

Arguments

x	The model fit, typically an object that implements a transfer method and ideally a block_names method.
new_data	The data to transfer, typically matching the dimension of the from domain.
from	Character string or index identifying the source domain/block. Must be present in block_names(x) if that method exists.
to	Character string or index identifying the target domain/block. Must be present in block_names(x) if that method exists.
opts	A list of optional arguments controlling the transfer process: cols Optional numeric vector specifying column indices of the *target* domain to reconstruct. If NULL (default), reconstructs all columns. comps Optional numeric vector specifying which latent components to use for the projection/reconstruction. If NULL (default), uses all components. ls_rr Logical; if TRUE, use a ridge-regularized LS approach for the initial projection from the from domain. Default FALSE. lambda Numeric ridge penalty (if ls_rr=TRUE). Default 1e-6.
...	Additional arguments passed to specific methods (discouraged, prefer opts).

Value

A matrix or data frame representing the transferred data in the `to` domain/block (or a subset of columns/components if specified in `opts`).

`transfer.cross_projector` *Transfer from X domain to Y domain (or vice versa) in a cross_projector*

Description

Convert between data representations in a multiblock or cross-decomposition model by projecting the input `new_data` from the `from` domain/block onto a latent space and then reconstructing it in the `to` domain/block.

Usage

```
## S3 method for class 'cross_projector'
transfer(x, new_data, from, to, opts = list(), ...)
```

Arguments

`x`	A `cross_projector` object.
`new_data`	The data to transfer.
`from`	Source domain ("X" or "Y").
`to`	Target domain ("X" or "Y").
`opts`	A list of options (see `transfer` generic).
`...`	Ignored.

Details

When `opts$ls_rr` is `TRUE`, the forward projection from the `from` domain is computed using a ridge-regularized least squares approach. The penalty parameter is taken from `opts$lambda`. Component subsetting via `opts$comps` is applied after computing these ridge-based scores.

Value

Transferred data matrix.

transform *Transform data using a fitted preprocessing pipeline*

Description

Apply a fitted preprocessing pipeline to new data. The preprocessing object must have been fitted using `fit()` or `fit_transform()` before calling this function.

Usage

```
transform(object, X, ...)
```

Arguments

`object`	A fitted preprocessing object
`X`	A matrix or data frame to transform
`...`	Additional arguments passed to methods

Value

The transformed data matrix

See Also

`fit()`, `fit_transform()`, `inverse_transform()`

Examples

```
# Transform new data with fitted preprocessor
X_train <- matrix(rnorm(100), 10, 10)
X_test <- matrix(rnorm(50), 5, 10)

preproc <- center()
fitted_preproc <- fit(preproc, X_train)
X_test_transformed <- transform(fitted_preproc, X_test)
```

transpose *Transpose a model*

Description

This function transposes a model by switching coefficients and scores. It is useful when you want to reverse the roles of samples and variables in a model, especially in the context of dimensionality reduction methods.

Usage

```
transpose(x, ...)
```

Arguments

x	The model fit, typically an object of a class that implements a transpose method
...	Additional arguments passed to the underlying transpose method

Value

A transposed model with coefficients and scores switched

See Also

bi_projector for an example of a two-way mapping model that can be transposed

---

truncate *truncate a component fit*

Description

take the first n components of a decomposition

Usage

```
truncate(x, ncomp)
```

Arguments

x	the object to truncate
ncomp	number of components to retain

Value

a truncated object (e.g. PCA with 'ncomp' components)

truncate.composed_projector *Truncate a Composed Projector*

Description

Reduces the number of output components of the composed projector by truncating the *last* stage in the sequence.

Usage

```
## S3 method for class 'composed_projector'
truncate(x, ncomp, ...)
```

Arguments

x	A composed_projector object.
ncomp	The desired number of final output components.
...	Currently unused.

Details

Note: This implementation currently only supports truncating the final stage. Truncating intermediate stages would require re-computing subsequent stages or combined attributes and is not yet implemented.

Value

A new composed_projector object with the last stage truncated.

variables_used *Identify Original Variables Used by a Projector*

Description

Determines which columns from the *original* input space contribute (have non-zero influence) to *any* of the output components of the projector.

Usage

```
variables_used(x, ...)

## S3 method for class 'composed_projector'
variables_used(x, tol = 1e-08, ...)
```

Arguments

x	A projector object (e.g., projector, composed_projector).
...	Additional arguments passed to specific methods.
tol	Numeric tolerance for determining non-zero coefficients. Default is 1e-8 for some methods. Passed via ....

Value

A sorted numeric vector of unique indices corresponding to the original input variables.

---

vars_for_component *Identify Original Variables for a Specific Component*

Description

Determines which columns from the *original* input space contribute (have non-zero influence) to a *specific* output component of the projector.

Usage

```
vars_for_component(x, k, ...)

## S3 method for class 'composed_projector'
vars_for_component(x, k, tol = 1e-08, ...)
```

Arguments

x	A projector object (e.g., projector, composed_projector).
k	The index of the output component to query.
...	Additional arguments passed to specific methods.
tol	Numeric tolerance for determining non-zero coefficients. Default is 1e-8 for some methods. Passed via ....

Value

A sorted numeric vector of unique indices corresponding to the original input variables.

Index