

# Package ‘multivarious’

March 28, 2024

**Title** Extensible Data Structures for Multivariate Analysis

**Version** 0.2.0

**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** 7.3.1

**Imports** rlang, chk, glmnet, corpcor, Matrix, purrr, rsvd, svd, pls, irlba, RSpecra, proxy, matrixStats, fitdistrplus

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

**URL** <https://bbuchsbaum.github.io/multivarious/>

**VignetteBuilder** knitr

**NeedsCompilation** no

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

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

**Repository** CRAN

**Date/Publication** 2024-03-28 16:00:02 UTC

## R topics documented:

add_node . . . . .	3
apply_rotation . . . . .	4
apply_transform . . . . .	4
bi_projector . . . . .	5
bi_projector_union . . . . .	6
block_indices . . . . .	6
block_lengths . . . . .	7
bootstrap . . . . .	7

bootstrap.pca . . . . .	8
center . . . . .	9
classifier . . . . .	9
classifier.discriminant_projector . . . . .	10
classifier.multiblock_biprojector . . . . .	10
classifier.projector . . . . .	11
coef.cross_projector . . . . .	12
colscale . . . . .	12
components . . . . .	13
compose_projector . . . . .	13
compose_projectors . . . . .	14
concat_pre_processors . . . . .	15
convert_domain . . . . .	16
cross_projector . . . . .	16
discriminant_projector . . . . .	18
fresh . . . . .	19
group_means . . . . .	19
inverse_projection . . . . .	20
is_orthogonal . . . . .	21
multiblock_biprojector . . . . .	21
multiblock_projector . . . . .	22
nblocks . . . . .	23
ncomp . . . . .	24
nystrom_embedding . . . . .	24
partial_inverse_projection . . . . .	25
partial_project . . . . .	26
partial_projector . . . . .	27
partial_projector.projector . . . . .	28
pass . . . . .	28
pca . . . . .	29
perm_ci . . . . .	30
predict.classifier . . . . .	30
prep . . . . .	31
prinang . . . . .	32
print.bi_projector . . . . .	32
print.bi_projector_union . . . . .	33
print.classifier . . . . .	33
print.composed_projector . . . . .	34
print.multiblock_biprojector . . . . .	34
print.projector . . . . .	35
project . . . . .	36
project.cross_projector . . . . .	37
projector . . . . .	37
project_block . . . . .	38
project_vars . . . . .	39
reconstruct . . . . .	40
refit . . . . .	40
regress . . . . .	41

<code>add_node</code>	3
<code>reprocess</code>	42
<code>reprocess.cross_projector</code>	43
<code>residualize</code>	43
<code>residuals</code>	44
<code>reverse_transform</code>	45
<code>rf_classifier</code>	45
<code>rf_classifier.projector</code>	46
<code>rotate</code>	46
<code>scores</code>	47
<code>sdev</code>	48
<code>shape</code>	48
<code>shape.cross_projector</code>	49
<code>standardize</code>	49
<code>std_scores</code>	50
<code>svd_wrapper</code>	50
<code>transpose</code>	51
<code>truncate</code>	52
<b>Index</b>	<b>53</b>

---

<code>add_node</code>	<i>add a pre-processing stage</i>
-----------------------	-----------------------------------

---

### Description

add a pre-processing stage

### Usage

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

### Arguments

<code>x</code>	the processing pipeline
<code>step</code>	the pre-processing step to add
<code>...</code>	extra args

### Value

a new pre-processing pipeline with the added step

---

apply_rotation	<i>Apply rotation</i>
----------------	-----------------------

---

**Description**

Apply a specified rotation to the fitted model

**Usage**

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

**Arguments**

x	A model object, possibly created using the <code>pca()</code> function.
rotation_matrix	matrix representing the rotation.
...	extra args

**Value**

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

---

apply_transform	<i>apply a pre-processing transform</i>
-----------------	---

---

**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

---

bi_projector	<i>Construct a bi_projector instance</i>
--------------	--

---

## 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) (number of columns = number of 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_lengths	<i>get block_lengths</i>
---------------	--------------------------

---

**Description**

extract the lengths of each block in a multiblock object

**Usage**

block\_lengths(x)

**Arguments**

x                    the object

**Value**

the block lengths

---

bootstrap	<i>Bootstrap Resampling for Multivariate Models</i>
-----------	---

---

**Description**

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

**Usage**

bootstrap(x, nboot, ...)

**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	<i>PCA Bootstrap Resampling</i>
---------------	---------------------------------

---

**Description**

Perform bootstrap resampling for Principal Component Analysis (PCA) to estimate component and score variability.

**Usage**

```
## S3 method for class 'pca'
bootstrap(x, nboot = 100, k = ncomp(x), ...)
```

**Arguments**

x	A fitted PCA model object.
nboot	The number of bootstrap resamples (default: 100).
k	The number of components to bootstrap (default: all components in the fitted PCA model).
...	Additional arguments to be passed to the specific model implementation of bootstrap.

**Value**

A list containing bootstrap z-scores for the loadings (zboot\_loadings) and scores (zboot\_scores).

**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.

**Examples**

```
X <- matrix(rnorm(10*100), 10, 100)
x <- pca(X, ncomp=9)
bootstrap_results <- bootstrap(x)
```



---

center	<i>center a data matrix</i>
--------	-----------------------------

---

**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	<i>Construct a Classifier</i>
------------	-------------------------------

---

**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, ...)
```

**Arguments**

x	A model object, such as a projector, that has been fit to a training dataset.
colind	Optional vector of column indices used for prediction. If not provided, all columns will be used.
...	Additional arguments to be passed to the specific model implementation of classifier.

**Value**

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

---

```
classifier.discriminant_projector
```

*Create a k-NN classifier for a discriminant projector*

---

### Description

Constructs a k-NN classifier for a discriminant projector, with an option to use a subset of the components.

### 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 to use for prediction (NULL by default)
knn	the number of nearest neighbors to consider in the k-NN classifier (default is 1)
...	extra arguments

### Value

a classifier object

---

```
classifier.multiblock_biprojector
```

*Multiblock Bi-Projector Classifier*

---

### Description

Constructs a classifier for a multiblock bi-projector model that can generate predictions for new data points.

### Usage

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

**Arguments**

x	A fitted multiblock bi-projector model object.
colind	An optional vector of column indices used for prediction (default: NULL).
labels	A factor or vector of class labels for the training data.
new_data	An optional data matrix for which to generate predictions (default: NULL).
block	An optional block index for prediction (default: NULL).
knn	The number of nearest neighbors to consider in the classifier (default: 1).
...	Additional arguments to be passed to the specific model implementation of classifier.

**Value**

A multiblock classifier object.

**See Also**

Other classifier: [classifier.projector\(\)](#)

`classifier.projector` *create classifier from a projector*

**Description**

create classifier from a projector

**Usage**

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

**Arguments**

x	A model object, such as a projector, that has been fit to a training dataset.
colind	Optional vector of column indices used for prediction. If not provided, all columns will be used.
labels	the labels associated with the rows of the projected data (see new_data)
new_data	reference data associated with labels and to be projected into subspace (required).
knn	the number of nearest neighbors to use when classifying a new point.
...	Additional arguments to be passed to the specific model implementation of classifier.

**Value**

a classifier object

**See Also**

Other classifier: `classifier.multiblock_biprojector()`

**Examples**

```
data(iris)
X <- iris[,1:4]
pcres <- pca(as.matrix(X),2)
cfier <- classifier(pcres, labels=iris[,5], new_data=as.matrix(iris[,1:4]))
p <- predict(cfier, as.matrix(iris[,1:4]))
```

---

`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**

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

**Value**

the coefficients

---

`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	<i>get the components</i>
------------	---------------------------

---

**Description**

Extract the component matrix of a fit.

**Usage**

components(x, ...)

**Arguments**

x	the model fit
...	extra args

**Value**

the component matrix

---

compose_projector	<i>Compose Two Projectors</i>
-------------------	-------------------------------

---

**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.

---

compose\_projectors     *Projector Composition*

---

**Description**

Compose a sequence of projector objects in forward order. This function allows the composition of multiple projectors, applying them sequentially to the input data.

**Usage**

```
compose_projectors(...)
```

**Arguments**

...            The sequence of projector objects to be composed.

**Value**

A composed\_projector object that extends the function class, allowing the composed projectors to be applied to input data.

**See Also**

[projector](#), [project](#)

**Examples**

```
# Create two PCA projectors and compose them
X <- matrix(rnorm(20*20), 20, 20)
pca1 <- pca(X, ncomp=10)
X2 <- scores(pca1)
pca2 <- pca(X2, ncomp=4)

# Compose the PCA projectors
cproj <- compose_projectors(pca1, pca2)
```

```

# Ensure the output of the composed projectors has the expected dimensions
stopifnot(ncol(cproj(X)) == 4)
# Check that the composed projectors work as expected
all.equal(project(cproj, X), cproj(X))

```

---

concat\_pre\_processors *bind together blockwise pre-processors*

---

### Description

concatenate a sequence of pre-processors, each previously 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 block indices where each vector in the list contains the global indices of the variables.

### Value

a new prepper object

### 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)

```

---

convert_domain	<i>Transfer data from one input domain to another via common latent space</i>
----------------	---

---

### Description

Convert between data representations in a multiblock decomposition/alignment by projecting the input data onto a common latent space and then reconstructing it in the target domain.

### Usage

```
convert_domain(x, new_data, i, j, comp, rowind, colind, ...)
```

### Arguments

x	The model fit, typically an object of a class that implements a transfer method
new_data	The data to transfer, with the same number of rows as the source data block
i	The index of the source data block
j	The index of the destination data block
comp	A vector of component indices to use in the reconstruction
rowind	Optional set of row indices to transfer (default: all rows)
colind	Optional set of column indices to transfer (default: all columns)
...	Additional arguments passed to the underlying convert_domain method

### Value

A matrix or data frame representing the transferred data in the target domain

### See Also

[project\\_block](#) for projecting a single block of data onto the subspace

---

cross_projector	<i>Two-way (cross) projection to latent components</i>
-----------------	--

---

### 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
vy	the Y coefficients
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 <- cancort(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 = "cancort")
```

---

 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

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

### 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)
```

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

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, ...)
```

**Arguments**

x	The model fit.
...	Extra arguments.

**Value**

The inverse projection matrix.

**See Also**

[project](#) for projecting data onto the subspace.

---

is_orthogonal	<i>is it orthogonal</i>
---------------	-------------------------

---

**Description**

test whether components are orthogonal

**Usage**

```
is_orthogonal(x)
```

**Arguments**

x                    the object

**Value**

a logical value indicating whether the transformation is orthogonal

---

multiblock_biprojector	<i>Create a Multiblock Bi-Projector</i>
------------------------	---

---

**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 the projection of multiblock data, where each block represents a different set of variables or features, with two-way mapping from samples to scores and from variables to components.

**Usage**

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

**Arguments**

<code>v</code>	A matrix of components with dimensions <code>nrow(v)</code> by <code>ncol(v)</code> (number of columns = number of components).
<code>s</code>	A matrix of scores.
<code>sdev</code>	A numeric vector of singular values.
<code>preproc</code>	A pre-processing function for the data (default is a pass-through with <code>prep(pass())</code> ).
<code>...</code>	Extra arguments.
<code>block_indices</code>	A list of numeric vectors specifying the indices of each data block.
<code>classes</code>	(optional) A character vector specifying the class attributes of the object, default is <code>NULL</code> .

**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**

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

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

---

nblocks	<i>get the number of blocks</i>
---------	---------------------------------

---

**Description**

The number of data blocks in a multiblock element

**Usage**

```
nblocks(x)
```

**Arguments**

x                    the object

**Value**

the number of blocks

---

ncomp	<i>Get the number of components</i>
-------	-------------------------------------

---

**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 <- iris[, 1:4]
fit <- svd_wrapper(X, ncomp = 3, preproc = center(), method = "base")
ncomp(fit) # Should return 3
```

---

nystrom_embedding	<i>Nystrom method for out-of-sample embedding</i>
-------------------	---

---

**Description**

Approximate the embedding of a new data point using the Nystrom method, which is particularly useful for large datasets and data-dependent embedding spaces, such as multidimensional scaling (MDS).

**Usage**

```
nystrom_embedding(  
  new_data,  
  landmark_data,  
  kernel_function,  
  eigenvectors,  
  eigenvalues,  
  ...  
)
```



**Arguments**

new_data	A matrix or data frame containing the new data points to be projected.
landmark_data	A matrix or data frame containing the landmark data points used for approximation.
kernel_function	A function used to compute the kernel matrix (e.g., a distance function for MDS).
eigenvectors	A matrix containing the eigenvectors obtained from the eigendecomposition of the kernel matrix between the landmark points.
eigenvalues	A vector containing the eigenvalues obtained from the eigendecomposition of the kernel matrix between the landmark points.
...	Additional arguments passed to the kernel_function.

**Value**

A matrix containing the approximate embedding of the new\_data in the data-dependent space.

---

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_project	<i>Partially project a new sample onto subspace</i>
-----------------	---

---

### Description

Project a selected subset of column indices onto the subspace. This function allows for the projection of new data onto a lower-dimensional space using only a subset of the variables, as specified by the column indices.

### Usage

```
partial_project(x, new_data, colind)
```

### Arguments

x	The model fit, typically an object of class <code>bi_projector</code> or any other class that implements a <code>partial_project</code> method
new_data	A matrix or vector of new observations with a subset of columns equal to length of <code>colind</code> . Rows represent observations and columns represent variables
colind	A numeric vector of column indices to select in the projection matrix. These indices correspond to the variables used for the partial projection

### Value

A matrix or vector of the partially 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 `partial_project` 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)

# Partially project new_data onto the same subspace as the original data
# using only the first 10 variables
new_data <- matrix(rnorm(5*20), 5, 20)
colind <- 1:10
partially_projected_data <- partial_project(p, new_data[,colind], colind)
```

---

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 <code>bi_projector</code> or any other class that implements a <code>partial_projector</code> 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 <code>partial_projector</code> 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)
```

---

```
partial_projector.projector
      construct a partial_projector from a projector instance
```

---

**Description**

construct a partial\_projector from a projector instance

**Usage**

```
## S3 method for class 'projector'
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 partial\_projector instance

**Examples**

```
X <- matrix(rnorm(10*10), 10, 10)
pfit <- pca(X, ncomp=9)
proj <- project(pfit, X)

pp <- partial_projector(pfit, 1:5)
```

---

```
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

---

<code>pca</code>	<i>Principal Components Analysis (PCA)</i>
------------------	--

---

**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)
```

---

perm\_ci                      *Permutation Confidence Intervals*

---

**Description**

Estimate confidence intervals for model parameters using permutation testing.

**Usage**

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

**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.

**Value**

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

---

predict.classifier      *predict with a classifier object*

---

**Description**

predict with a classifier object

**Usage**

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

**Arguments**

object	the model fit
new_data	new data to predict on
ncomp	the number of components to use
colind	the column indices to select in the projection matrix
metric	the similarity metric ("euclidean" or "cosine")
...	additional arguments to projection function

**Value**

a list with the predicted class and probabilities

---

prep	<i>prepare a dataset by applying a pre-processing pipeline</i>
------	--

---

**Description**

prepare a dataset by applying a pre-processing pipeline

**Usage**

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

**Arguments**

x	the pipeline
...	extra args

**Value**

the pre-processed data

---

prinang	<i>Compute principal angles for a set of subspaces</i>
---------	--

---

**Description**

This function calculates the principal angles between subspaces derived from a list of `bi_projector` instances.

**Usage**

```
prinang(fits)
```

**Arguments**

`fits` a list of `bi_projector` instances

**Value**

a numeric vector of principal angles with length equal to the minimum dimension of input subspaces

**Examples**

```
data(iris)
X <- as.matrix(iris[, 1:4])
res <- pca(X, ncomp = 4)
fits_list <- list(res,res,res)
principal_angles <- prinang(fits_list)
```

---

print.bi_projector	<i>Pretty Print S3 Method for bi_projector Class</i>
--------------------	--

---

**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.bi_projector_union
```

*Pretty Print S3 Method for bi\_projector\_union Class*

---

**Description**

Pretty Print S3 Method for `bi_projector_union` Class

**Usage**

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

**Arguments**

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

**Value**

Invisible `bi_projector_union` object

---

```
print.classifier
```

*Pretty Print Method for classifier Objects*

---

**Description**

Display a human-readable summary of a `classifier` object, including information about the k-NN classifier, the model fit, and the dimensions of the scores matrix.

**Usage**

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

**Arguments**

`x`                    A `classifier` object.  
`...`                Additional arguments passed to `print()`.

**Value**

`classifier` object.

---

```
print.composed_projector
```

*Pretty Print Method for composed\_projector Objects*

---

### Description

Display a human-readable summary of a `composed_projector` object, including information about the number and order of projectors.

### Usage

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

### Arguments

`x`                    A `composed_projector` object.  
`...`                 Additional arguments passed to `print()`.

### Value

The `composed_projector` object.

### Examples

```
# Create two PCA projectors and compose them  
X <- matrix(rnorm(20*20), 20, 20)  
pca1 <- pca(X, ncomp=10)  
X2 <- scores(pca1)  
pca2 <- pca(X2, ncomp=4)  
cproj <- compose_projectors(pca1, pca2)
```

---

```
print.multiblock_biprojector
```

*Pretty Print Method for multiblock\_biprojector Objects*

---

### Description

Display a human-readable summary of a `multiblock_biprojector` object, including information about the dimensions of the projection matrix, the pre-processing pipeline, and block indices.

### 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.

**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 bi-projector using PCA components and block indices
mb_biproj <- multiblock_biprojector(pc$v, s = pc$u %*% diag(sdev(pc)), sdev = sdev(pc),
  block_indices = list(1:5, 6:10))
# Pretty print the multiblock bi-projector object
print(mb_biproj)
```

---

print.projector

*Pretty Print Method for projector Objects*


---

**Description**

Display a human-readable summary of a projector object, including information about the dimensions of the projection matrix and the pre-processing pipeline.

**Usage**

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

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

**Arguments**

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

**Value**

the projector object

**Examples**

```
X <- matrix(rnorm(10*10), 10, 10)
svdfit <- svd(X)
p <- projector(svdfit$v)
print(p)
```

---

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 <code>bi_projector</code> 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\(\)](#)

---

```
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 <code>nrow(v)</code> by <code>ncol(v)</code> (number of columns = number of components)
preproc	A prepped pre-processing object. Default is the no-processing <code>pass()</code> 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.

**Examples**

```
X <- matrix(rnorm(10*10), 10, 10)
svdfit <- svd(X)
p <- projector(svdfit$v)
proj <- project(p, X)
```

---

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, ...)
```

**Arguments**

x	The model fit, typically an object of a class that implements a <code>project_block</code> 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
...	Additional arguments passed to the underlying <code>project_block</code> 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\_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\(\)](#)

---

reconstruct	<i>Reconstruct the data</i>
-------------	-----------------------------

---

**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, comp, rowind, colind, ...)
```

**Arguments**

x	The model fit, typically an object of a class that implements a reconstruct method
comp	A vector of component indices to use in the reconstruction
rowind	The row indices to reconstruct (optional). If not provided, all rows are used.
colind	The column indices to reconstruct (optional). If not provided, all columns are used.
...	Additional arguments passed to the underlying reconstruct method

**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

---

refit	<i>refit a model</i>
-------	----------------------

---

**Description**

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

**Usage**

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



**Arguments**

<code>x</code>	the original model fit object
<code>new_data</code>	the new data to process
<code>...</code>	extra args

**Value**

a refit model object

---

<code>regress</code>	<i>Multi-output linear regression</i>
----------------------	---------------------------------------

---

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

**Arguments**

<code>X</code>	the set of independent (basis) variables
<code>Y</code>	the response matrix
<code>preproc</code>	the pre-processor (currently unused)
<code>method</code>	the regression method: <code>lm</code> , <code>enet</code> , <code>mridge</code> , or <code>pls</code>
<code>intercept</code>	whether to include an intercept term
<code>lambda</code>	ridge shrinkage parameter (for methods <code>mridge</code> and <code>enet</code> )
<code>alpha</code>	the elastic net mixing parameter if method is <code>enet</code>
<code>ncomp</code>	number of PLS components if method is <code>pls</code>
<code>...</code>	extra arguments sent to the underlying fitting function

**Value**

a bi-projector of type regress

**Examples**

```
# Generate synthetic data
Y <- matrix(rnorm(100 * 10), 10, 100)
X <- matrix(rnorm(10 * 9), 10, 9)
# Fit regression models and reconstruct the response matrix
r_lm <- regress(X, Y, intercept = FALSE, method = "lm")
recon_lm <- reconstruct(r_lm)
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

**Value**

the re(pre-)processed data

---

```
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**

<code>x</code>	The model fit object.
<code>ncomp</code>	The number of components to factor out before calculating residuals.
<code>xorig</code>	The original data matrix (X) used to fit the model.
<code>...</code>	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**

create a random forest classifier

**Usage**

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

**Arguments**

x	the model object
colind	the (optional) column indices used for prediction
labels	A factor or vector of class labels for the training data.
scores	a matrix of references scores used for classification
...	extra arguments to randomForest function

**Value**

a rf\_classifier object

**Examples**

```
data(iris)  
X <- iris[,1:4]  
pcres <- pca(as.matrix(X),2)  
cfier <- rf_classifier(pcres, labels=iris[,5], scores=scores(pcres))  
p <- predict(cfier, new_data=as.matrix(iris[,1:4]))
```

---

```
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").

**Value**

A modified model fit with the rotated components.

---

scores	<i>Retrieve the component scores</i>
--------	--------------------------------------

---

**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.

---

sdev	<i>standard deviations</i>
------	----------------------------

---

**Description**

The standard deviations of the projected data matrix

**Usage**

sdev(x)

**Arguments**

x                    the model fit

**Value**

the standard deviations

---

shape	<i>Shape of the Projector</i>
-------	-------------------------------

---

**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	<i>Compute standardized component scores</i>
------------	--

---

**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.

---

svd_wrapper	<i>Singular Value Decomposition (SVD) Wrapper</i>
-------------	---

---

**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**

<code>X</code>	the input matrix
<code>ncomp</code>	the number of components to estimate (default: <code>min(dim(X))</code> )
<code>preproc</code>	the pre-processor to apply on the input matrix (e.g., <code>center()</code> , <code>standardize()</code> , <code>pass()</code> )
<code>method</code>	the SVD method to use: 'base', 'fast', 'irlba', 'propack', 'rsvd', or 'svds'
<code>q</code>	parameter passed to method <code>rsvd</code> (default: 2)
<code>p</code>	parameter passed to method <code>rsvd</code> (default: 10)
<code>tol</code>	minimum eigenvalue magnitude, otherwise component is dropped (default: <code>.Machine\$double.eps</code> )
<code>...</code>	extra arguments passed to the selected SVD function

**Value**

an SVD object that extends `projector`

**Examples**

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

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

---

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**

<code>x</code>	The model fit, typically an object of a class that implements a <code>transpose</code> method
<code>...</code>	Additional arguments passed to the underlying <code>transpose</code> 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	<i>truncate a component fit</i>
----------	---------------------------------

---

**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)

# Index

- \* **bootstrap**
  - bootstrap.pca, 8
- \* **classifier**
  - classifier.multiblock\_biprojector, 10
  - classifier.projector, 11
- \* **partial\_project**
  - partial\_project, 26
- \* **project**
  - project, 36
  - project.cross\_projector, 37
  - project\_block, 38
  - project\_vars, 39
- \* **reconstruct**
  - reconstruct, 40
- \* **reprocess**
  - reprocess.cross\_projector, 43
- \* **residuals**
  - residuals, 44
- \* **scores**
  - scores, 47
- \* **shape**
  - shape.cross\_projector, 49
- \* **transpose**
  - transpose, 51
- add\_node, 3
- apply\_rotation, 4
- apply\_transform, 4
- bi\_projector, 5, 26, 27, 36, 40, 52
- bi\_projector\_union, 6
- block\_indices, 6
- block\_lengths, 7
- bootstrap, 7
- bootstrap.pca, 8
- center, 9
- classifier, 9
- classifier.discriminant\_projector, 10
- classifier.multiblock\_biprojector, 10, 12
- classifier.projector, 11, 11
- coef.cross\_projector, 12
- colscale, 12
- components, 13
- compose\_projector, 13
- compose\_projectors, 14
- concat\_pre\_processors, 15
- convert\_domain, 16
- cross\_projector, 16
- discriminant\_projector, 18
- fresh, 19
- group\_means, 19
- inverse\_projection, 20
- is\_orthogonal, 21
- multiblock\_biprojector, 21
- multiblock\_projector, 22
- nblocks, 23
- ncomp, 24
- nystrom\_embedding, 24
- partial\_inverse\_projection, 25
- partial\_project, 26
- partial\_projector, 27
- partial\_projector.projector, 28
- pass, 28
- pca, 29
- perm\_ci, 30
- predict.classifier, 30
- prep, 31
- prinang, 32
- print.bi\_projector, 32
- print.bi\_projector\_union, 33
- print.classifier, 33

print.composed\_projector, 34  
print.multiblock\_biprojector, 34  
print.projector, 35  
project, 14, 20, 36, 37, 39, 47  
project.cross\_projector, 36, 37, 39  
project\_block, 16, 36, 37, 38, 39  
project\_vars, 36, 37, 39, 39  
projector, 14, 37  
  
reconstruct, 40  
refit, 40  
regress, 41  
reprocess, 42  
reprocess.cross\_projector, 43  
residualize, 43  
residuals, 44  
reverse\_transform, 45  
rf\_classifier, 45  
rf\_classifier.projector, 46  
rotate, 46  
  
scores, 47, 50  
sdev, 48  
shape, 48  
shape.cross\_projector, 49  
standardize, 49  
std\_scores, 50  
svd\_wrapper, 29, 50  
  
transpose, 51  
truncate, 52