Package: xnet (via r-universe)

October 31, 2024

Type Package

Title Two-Step Kernel Ridge Regression for Network Predictions

Version 0.1.11

Description Fit a two-step kernel ridge regression model for predicting edges in networks, and carry out cross-validation using shortcuts for swift and accurate performance assessment (Stock et al, 2018 <doi:10.1093/bib/bby095>).

Date 2020-02-03

BugReports https://github.com/CenterForStatistics-UGent/xnet/issues

URL https://github.com/CenterForStatistics-UGent/xnet

Depends R(>= 3.4.0)

Imports methods, utils, graphics, stats, grDevices

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Suggests testthat, knitr, rmarkdown, ChemmineR, covr, fmcsR

VignetteBuilder knitr

Collate 'Class_linearFilter.R' 'all_generics.R' 'Class_permtest.R' 'Class_tskrr.R' 'Class_tskrrHeterogeneous.R' 'Class_tskrrHomogeneous.R' 'Class_tskrrImpute.R' 'Class_tskrrImputeHeterogeneous.R' 'Class_tskrrImputeHomogeneous.R' 'Class_tskrrTune.R' 'Class_tskrrTuneHeterogeneous.R' 'Class_tskrrTuneHomogeneous.R' 'as_tuned.R' 'create_grid.R' 'data_drugtarget.R' 'data_proteinInteraction.R' 'dim.R' 'eigen2hat.R' 'fitted.R' 'get_loo_fun.R' 'getlooInternal.R' 'getters_linearFilter.R' 'getters_permtest.R' 'getters_tskrr.R' 'getters_tskrrImpute.R' 'getters_tskrrTune.R' 'hat.R' 'impute_tskrr.R' 'impute_tskrr.fit.R' 'internal_helpers.R' 'is_symmetric.R' 'labels.R' 'linear_filter.R' 'loo.R' 'looInternal.R' 'loss.R' 'loss_functions.R' 'match_labels.R' 'permtest.R' 'plot.tskrr.R' 'plot_grid.R' 'predict.R' 'prepare_lambdas.R' 'residuals.R' 'test_input.R' 'test_symmetry.R' 'tskrr.R' 'tskrr.fit.R' 'tune.R' 'update.R' 'valid_dimensions.R' 'valid_labels.R' 'weights.R' 'xnet-package.R'

Repository https://centerforstatistics-ugent.r-universe.dev

RemoteUrl https://github.com/centerforstatistics-ugent/xnet

RemoteRef HEAD

RemoteSha 4093905ae81281b6cf81c6a3425bdaf884e78fb4

Contents

`xnet-package` *Two-step kernel ridge regression for network analysis*

Description

This package implements the two-step kernel ridge regression model, a supervised network prediction method that can be used for all kinds of network analyses. Examples are protein-protein interaction, foodwebs, ...

Author(s)

Joris Meys and Michiel Stock

See Also

Send your bug reports to:

`https://github.com/CenterForStatistics-UGent/xnet/issues`

More background in the paper by Stock et al, 2018:

`http://doi.org/10.1093/bib/bby095`

alpha *Getters for linearFilter objects*

Description

These functions allow you to extract slots from objects of the class `linearFilter`.

Usage

```
alpha(x)

na_removed(x)

## S3 method for class 'linearFilter'
mean(x, ...)

## S4 method for signature 'linearFilter'
mean(x, ...)

## S4 method for signature 'linearFilter'
colMeans(x)

## S4 method for signature 'linearFilter'
rowMeans(x)

## S4 method for signature 'linearFilter'
alpha(x)

## S4 method for signature 'linearFilter'
na_removed(x)
```

Arguments

`x`	a `linearFilter` object
`...`	arguments passed to or from other methods.

Value

for `mean`: the mean of the original matrix

for `colMeans`: a numeric vector with the column means

for `rowMeans`: a numeric vector with the row means

for `alpha`: a numeric vector of length 4 with the alpha values.

for `na_removed`: a logical value indicating whether missing values were removed prior to the fitting of the filter.

Examples

```
data(drugtarget)
lf <- linear_filter(drugTargetInteraction, alpha = 0.25)
alpha(lf)
mean(lf)
colMeans(lf)
na_removed(lf)
```

---

as_tuned — *convert tskrr models*

Description

These functions allow converting models that inherit from the `tskrr` and `tskrrTune` class into each other, keeping track of whether the model is homogeneous or heterogeneous. The dots argument allows specifying values for possible extra slots when converting from `tskrr` to `tskrrTune`. More information on these slots can be found on the help page of `tskrrTune`. **These functions are not exported.**

Usage

```
as_tuned(x, ...)

as_tskrr(x, ...)

## S4 method for signature 'tskrrHomogeneous'
as_tuned(x, ...)

## S4 method for signature 'tskrrHeterogeneous'
as_tuned(x, ...)

## S4 method for signature 'tskrrTune'
as_tskrr(x)

## S4 method for signature 'tskrrImpute'
as_tskrr(x)

## S4 method for signature 'tskrr'
as_tskrr(x)
```

Arguments

`x`	a model of class `tskrr`
`...`	values for the extra slots defined by the class `tskrrTune`

Value

For `as_tuned`: a `tskrrTune` object of the proper class (homogeneous or heterogeneous)

For `as_tskrr`: an object of class `tskrrHomogeneous` or `tskrrHeterogeneous` depending on whether the original object was homogeneous or heterogeneous.

Warning

This functions do NOT tune a model. they are used internally to make the connection between both types in the methods.

See Also

- `tune` for actually tuning a model.
- `tskrrTune` for names and possible values of the slots passed through ...

`create_grid` *Create a grid of values for tuning tskrr*

Description

This function creates a grid of values for tuning a `tskrr` model. The grid is equally spaced on a logarithmic scale. Normally it's not needed to call this method directly, it's usually called from `tune`.

Usage

```
create_grid(lim = c(1e-04, 10000), ngrid = 10)
```

Arguments

`lim`	a numeric vector with 2 values giving the lower and upper limit for the grid.
`ngrid`	the number of values that have to be produced. If this number is not integer, it is truncated. The value should be 2 or larger.

Details

The `lim` argument sets the boundaries of the domain in which the lambdas are sought. The lambda values at which the function is evaluated, are calculated as:

`exp(seq(log(1e-4), log(1e4), length.out = ngrid))`

Value

a numeric vector with values evenly spaced on a logarithmic scale.

See Also

`tune` for tuning a tskrr model.

Examples

```
create_grid(lim = c(1e-4, 1), ngrid = 5)
```

`dim,tskrr-method` *Get the dimensions of a tskrr object*

Description

These functions allow you to extract the dimensions of a tskrr object. These dimensions are essentially the dimensions of the label matrix y.

Usage

```
## S4 method for signature 'tskrr'
dim(x)
```

Arguments

`x`	a `tskrr` object.

Value

a vector with two values indicating the number of rows and the number of columns.

Examples

```
data(drugtarget)
mod <- tskrr(drugTargetInteraction, targetSim, drugSim)
dim(mod)
nrow(mod)
ncol(mod)
```

`drugTargetInteraction` *drug target interactions for neural receptors*

Description

A dataset for examining the interaction between 54 drugs and 26 neural receptors. It consists of three different matrices.

Usage

```
drugTargetInteraction
```

Format

- for drugTargetInteraction: a numeric matrix of 26 rows by 54 columns.
- For drugSim: a numeric square matrix with 54 rows/columns.
- For targetSim: a numeric square matrix with 26 rows/columns.

Details

The dataset consists of the following objects :

- drugTargetInteraction: a matrix indicating whether or not a certain drug compound interacts with a certain neural receptor.
- targetSim: a similarity matrix for the neural receptors.
- drugSim: a similarity matrix for the drugs

The data originates from Yamanishi et al (2008) but was partly reworked to be suitable for two-step kernel ridge regression. This is explained in detail in the Preparation of the example data vignette.

Source

`https://doi.org/10.1093/bioinformatics/btn162`

References

Yamanishi et al, 2008 : Prediction of drug-target interaction networks from the integration of chemical and genomic spaces.

`eigen2hat` *Calculate the hat matrix from an eigen decomposition*

Description

These functions calculate either the hat matrix, the mapping matrix or the original (kernel) matrix for a two-step kernel ridge regression, based on the eigendecomposition of the kernel matrix.

Usage

```
eigen2hat(eigen, val, lambda)

eigen2map(eigen, val, lambda)

eigen2matrix(eigen, val)
```

Arguments

`eigen`	a matrix with the eigenvectors.
`val`	an numeric vector with the eigenvalues.
`lambda`	a single numeric value for the hyperparameter lambda

Details

For the hat matrix, this boils down to:

$$U\Sigma(\Sigma + \lambda I)^{-1}U^T$$

For the map matrix, this is :

$$U(\Sigma + \lambda I)^{-1}U^T$$

with U the matrix with eigenvectors, Σ a diagonal matrix with the eigenvalues on the diagonal, I the identity matrix and λ the hyperparameter linked to this kernel. The internal calculation is optimized to avoid having to invert a matrix. This is done using the fact that Σ is a diagonal matrix.

Value

a numeric matrix representing either the hat matrix (`eigen2hat`), the map matrix (`eigen2map`) or the original matrix (`eigen2matrix`)

`fitted.tskrr` *extract the predictions*

Description

This functions extracts the fitted predictions from a `tskrr` object or an object inheriting from that class. The `xnet` package provides an S4 generic for the function `fitted` from the package `stats`, and a method for `tskrr` objects.

Usage

```
## S3 method for class 'tskrr'
fitted(object, labels = TRUE, ...)

## S3 method for class 'linearFilter'
fitted(object, ...)

## S4 method for signature 'tskrr'
fitted(object, labels = TRUE, ...)

## S4 method for signature 'linearFilter'
fitted(object, ...)
```

Arguments

`object`	an object for which the extraction of model fitted values is meaningful.
`labels`	a logical value indicating whether the labels should be shown. Defaults to TRUE
`...`	arguments passed to or from other methods.

Value

a numeric matrix with the predictions

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)
pred <- fitted(mod)
```

---

get_loo_fun   *Retrieve a loo function*

Description

This function returns the correct function needed to perform one of the leave-one-out cross-validations. It's primarily meant for internal use but can be useful when doing simulations.

Usage

```
get_loo_fun(x, ...)

## S4 method for signature 'tskrrHeterogeneous'
get_loo_fun(
  x,
  exclusion = c("interaction", "row", "column", "both"),
  replaceby0 = FALSE
)

## S4 method for signature 'tskrrHomogeneous'
get_loo_fun(
  x,
  exclusion = c("edges", "vertices", "interaction", "both"),
  replaceby0 = FALSE
)

## S4 method for signature 'linearFilter'
get_loo_fun(x, replaceby0 = FALSE)

## S4 method for signature 'character'
get_loo_fun(
  x = c("tskrrHeterogeneous", "tskrrHomogeneous", "linearFilter"),
  ...
)

## S4 method for signature 'tskrrTune'
get_loo_fun(x, ...)
```

Arguments

`x`	a character value with the class or a `tskrr` or `linearFilter` object.
`...`	arguments passed to or from other methods.
`exclusion`	a character value with possible values "interaction", "row", "column", "both" for heterogeneous models, and "edges", "vertices", "interaction" or "both" for homogeneous models. Defaults to "interaction". See details.
`replaceby0`	a logical value indicating whether the interaction should be simply removed (`FALSE`) or replaced by 0 (`TRUE`).

Details

This function can be used to select the correct loo function in a simulation or tuning algorithm, based on the model object you created. Depending on its class, the returned functions will have different arguments, so you should only use this if you know what you're doing and after you checked the actual returned functions in `loo_internal`.

Using `replaceby0` only makes sense if you only remove the interaction. In all other cases, this argument is ignored.

For the class `tskrrHomogeneous`, it doesn't make sense to remove rows or columns. If you chose this option, the function will throw an error. Removing edges corresponds to the setting "edges" or "interaction". Removing vertices corresponds to the setting "vertices" or "both". These terms can be used interchangeably.

For the class `linearFilter` it only makes sense to exclude the interaction (i.e., a single cell). Therefore you do not have an argument `exclusion` for that method.

For the classes `tskrrTune` and `tskrrImpute`, not specifying `exclusion` or `replaceby0` returns the used loo function. If you specify either of them, it will use the method for the appropriate model and return a new loo function.

Value

a function taking the arguments y, and possibly pred for calculating the leave-one-out cross-validation. For class `tskrrHeterogeneous`, the returned function also has an argument Hk and Hg, representing the hat matrix for the rows and the columns respectively. For class `tskrrHomogeneous`, only the extra argument Hk is available. For class `linearFilter`, the extra argument is called `alpha` and takes the alpha vector of that model.

See Also

`loo` for carrying out a leave on out crossvalidation, and `loo_internal` for more information on the internal functions one retrieves with this one.

has_imputed_values *Getters for tskrrImpute objects*

Description

The functions described here are convenience functions to get information out of a tskrrImpute object.

Usage

```
has_imputed_values(x)

which_imputed(x)

is_imputed(x)
```

Arguments

x	a tskrrImpute object or an object inheriting from tskrrImpute.

Value

For has_imputed_values: a logical value indicating whether the model has imputed values. If x is not some form of a tskrr model, the function will return an error.

For which_imputed: a integer vector with the positions for which the values are imputed.

for is_imputed: a matrix of the same dimensions as the label matrix. It contains the value FALSE at positions that were not imputed, and TRUE at positions that were.

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)

naid <- sample(length(drugTargetInteraction), 30)
drugTargetInteraction[naid] <- NA

impmod <- impute_tskrr(drugTargetInteraction, targetSim, drugSim)

has_imputed_values(mod)
has_imputed_values(impmod)

# For illustration: extract imputed values
id <- is_imputed(impmod)
fitted(impmod)[id]
```

hat — *Return the hat matrix of a tskrr model*

Description

This function returns the hat matrix or hat matrices of a tskrr model. `xnet` creates an S4 generic for `hat` and links the default method to the `hat` function of `stats`

Usage

```
hat(x, ...)

## S4 method for signature 'tskrrHeterogeneous'
hat(x, which = c("row", "column"))

## S4 method for signature 'tskrrHomogeneous'
hat(x, ...)
```

Arguments

`x`	a tskrr model
`...`	arguments passed to other methods.
`which`	a character value with possible values "row" or "column" to indicate which should be returned. For homogeneous models, this parameter is ignored.

Value

the requested hat matrix of the model.

impute_tskrr — *Impute missing values in a label matrix*

Description

This function implements an optimization algorithm that allows imputing missing values in the label matrix while fitting a `tskrr` model.

Usage

```
impute_tskrr(
  y,
  k,
  g = NULL,
  lambda = 0.01,
  testdim = TRUE,
  testlabels = TRUE,
  symmetry = c("auto", "symmetric", "skewed"),
  keep = FALSE,
  niter = 10000,
  tol = sqrt(.Machine$double.eps),
  start = mean(y, na.rm = TRUE),
  verbose = FALSE
)
```

Arguments

`y`	a label matrix
`k`	a kernel matrix for the rows
`g`	an optional kernel matrix for the columns
`lambda`	a numeric vector with one or two values for the hyperparameter lambda. If two values are given, the first one is used for the k matrix and the second for the g matrix.
`testdim`	a logical value indicating whether symmetry and the dimensions of the kernel(s) should be tested. Defaults to `TRUE`, but for large matrices putting this to `FALSE` will speed up the function.
`testlabels`	a logical value indicating wether the row- and column names of the matrices have to be checked for consistency. Defaults to `TRUE`, but for large matrices putting this to `FALSE` will speed up the function.
`symmetry`	a character value with the possibilities "auto", "symmetric" or "skewed". In case of a homogeneous fit, you can either specify whether the label matrix is symmetric or skewed, or you can let the function decide (option "auto").
`keep`	a logical value indicating whether the kernel hat matrices should be stored in the model object. Doing so makes the model object quite larger, but can speed up predictions in some cases. Defaults to `FALSE`.
`niter`	an integer giving the maximum number of iterations
`tol`	a numeric value indicating the tolerance for convergence of the algorithm. It is the maximum sum of squared differences between to iteration steps.
`start`	a numeric value indicating the value with which NA's are replaced in the first step of the algorithm. Defaults to 0.
`verbose`	either a logical value, 1 or 2. `1` means "show the number of iterations and the final deviation", 2 means "show the deviation every 10 iterations". A value `TRUE` is read as `1`.

Value

A `tskrr` model of the class `tskrrImputeHeterogeneous` or `tskrrImputeHomogeneous` depending on whether or not `g` has a value.

Examples

```
data(drugtarget)

naid <- sample(length(drugTargetInteraction), 30)
drugTargetInteraction[naid] <- NA

impute_tskrr(drugTargetInteraction, targetSim, drugSim)
```

`impute_tskrr.fit` *Impute values based on a two-step kernel ridge regression*

Description

This function provides an interface for the imputation of values based on a `tskrr` model and is the internal function used by `impute_tskrr`.

Usage

```
impute_tskrr.fit(y, Hk, Hg, naid = NULL, niter, tol, start, verbose)
```

Arguments

`y`	a label matrix
`Hk`	a hat matrix for the rows (see also `eigen2hat` on how to calculate them from an eigen decomposition)
`Hg`	a hat matrix for the columns. For homogeneous networks, this should be Hk again.
`naid`	an optional index with the values that have to be imputed, i.e. at which positions you find a `NA` value. It can be a vector with integers or a matrix with `TRUE`/`FALSE` values.
`niter`	an integer giving the maximum number of iterations
`tol`	a numeric value indicating the tolerance for convergence of the algorithm. It is the maximum sum of squared differences between to iteration steps.
`start`	a numeric value indicating the value with which NA's are replaced in the first step of the algorithm. Defaults to 0.
`verbose`	either a logical value, 1 or 2. 1 means "show the number of iterations and the final deviation", 2 means "show the deviation every 10 iterations". A value `TRUE` is read as `1`.

Details

This function is mostly available for internal use. In most cases, it makes much more sense to use `impute_tskrr`, as that function returns an object one can work with. The function `impute_tskrr.fit` could be useful when doing simulations or creating fitting algorithms.

Value

a list with two elements:

- a matrix `y` with the imputed values filled in.
- a numeric value `niter` with the amount of iterations

See Also

- `impute_tskrr` for the user-level function, and
- `eigen2hat` for conversion of a eigen decomposition to a hat matrix.

Examples

```
data(drugtarget)

K <- eigen(targetSim)
G <- eigen(drugSim)

Hk <- eigen2hat(K$vectors, K$values, lambda = 0.01)
Hg <- eigen2hat(G$vectors, G$values, lambda = 0.05)

drugTargetInteraction[c(3,17,123)] <- NA

res <- impute_tskrr.fit(drugTargetInteraction, Hk, Hg,
                        niter = 1000, tol = 10e-10,
                        start = 0, verbose = FALSE)
```

`is_symmetric` *Test symmetry of a matrix*

Description

The function `isSymmetric` tests for symmetry of a matrix but also takes row and column names into account. This function is a toned-down (and slightly faster) version that ignores row and column names. Currently, the function only works for real matrices, not complex ones.

Usage

```
is_symmetric(x, tol = 100 * .Machine$double.eps)
```

Arguments

`x`	a matrix to be tested.
`tol`	the tolerance for comparing the numbers.

Value

a logical value indicating whether or not the matrix is symmetric

Examples

```
x <- matrix(1:16,ncol = 4)
is_symmetric(x)

x <- x %*% t(x)
is_symmetric(x)
```

---

is_tuned *Getters for tskrrTune objects*

Description

The functions described here are convenience functions to get information out of a `tskrrTune` object.

Usage

```
is_tuned(x)

get_grid(x)

get_loss_values(x)

has_onedim(x)
```

Arguments

x	a `tskrrTune` object or an object inheriting from `tskrrTune`.

Value

For `is_tuned`: a logical value indicating whether the model is tuned.

For `get_grid` a list with the elements `k` and possibly `g`, each containing the different lambdas tried in the tuning for the row and column kernel matrices respectively.

For `get_loss_values` a matrix with the calculated loss values. Note that each row represents the result for one lambda value related to the row kernel matrix K. For heterogeneous models, every column represents the result for one lambda related to the column kernel matrix G.

for `is_onedim` a single logical value telling whether the grid search in the object was onedimensional.

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)
tuned <- tune(mod, ngrid = 10)

is_tuned(mod)
is_tuned(tuned)

# Basic visualization of the grid.

gridvals <- get_grid(tuned)
z <- get_loss_values(tuned)

## Not run:
image(gridvals$k,gridvals$g,log(z), log = 'xy',
xlab = "lambda k", ylab = "lambda g")

## End(Not run)
```

---

labels.tskrr *Extract labels from a tskrr object*

Description

These functions allow you to extract the labels from a tskrr object. The function labels and the function dimnames are aliases and do the exact same thing. The functions rownames and colnames work like you would expect. Note that contrary to the latter two, labels will never return NULL. If no labels are found, it will construct labels using the prefixes defined in the argument prefix.

Usage

```
## S3 method for class 'tskrr'
labels(
  object,
  prefix = if (is_homogeneous(object)) "row" else c("row", "col"),
  ...
)

## S4 method for signature 'tskrr'
labels(
  object,
  prefix = if (is_homogeneous(object)) "row" else c("row", "col"),
  ...
)

## S4 method for signature 'tskrr'
```

```
dimnames(x)

## S4 method for signature 'tskrr'
rownames(x, do.NULL = TRUE, prefix = "row")

## S4 method for signature 'tskrr'
colnames(x, do.NULL = TRUE, prefix = "col")
```

Arguments

`object`	a `tskrr` object
`prefix`	a prefix used for construction of the labels in case none are available. For `label`, a character vector of length 1 for homogeneous networks or of length 2 for heterogeneous networks. In case two values are given, the first is used for the rows and the second for the columns. Otherwise the only value is used for both. In the case of `rownames` and `colnames`, a single value. See also `row+colnames`
`...`	arguments passed to/from other methods.
`x`	a `tskrr` object
`do.NULL`	logical. If `FALSE` and labels are `NULL`, labels are created. If `TRUE`, the function returns `NULL` in the absence of labels.

Value

for `labels` and `dimnames`: a list with two elements `k` and `g`

Warning

If the original data didn't contain row- or column names for the label matrix, `rownames` and `colnames` will return `NULL`. Other functions will extract the automatically generated labels, so don't count on `rownames` and `colnames` if you want to predict output from other functions!

`linearFilter-class` *Class linearFilter*

Description

The class represents the outcome of a linear filter, and is normally generated by the function `linear_filter`

Slots

`y` the original label matrix with responses.

`alpha` a numeric vector with the 4 alpha values of the model.

`pred` a matrix with the predictions

`mean` a numeric vector containing the global mean of `y` `colmeans` a numeric vector containing the column means of `y`

`rowmeans` a numeric vector containing the row means of `y`.

`na.rm` a logical value indicating whether missing values were removed prior to the calculation of the means.

See Also

`linear_filter` for creating a linear filter model, and `getter fuctions for linearFilter`.

`linear_filter` *Fit a linear filter over a label matrix*

Description

This function fits a linear filter over a label matrix. It calculates the row, column and total means, and uses those to construct the linear filter.

Usage

```
linear_filter(y, alpha = 0.25, na.rm = FALSE)
```

Arguments

`y`	a label matrix
`alpha`	a vector with 4 alpha values, or a single alpha value which then is used for all 4 alphas.
`na.rm`	a logical value indicating whether missing values should be removed before calculating the row-, column- and total means.

Details

If there are missing values and they are removed before calculating the means, a warning is issued. If `na.rm = FALSE` and there are missing values present, the outcome is, by definition, a matrix filled with NA values.

Value

an object of class `linearFilter`

Examples

```
data(drugtarget)
linear_filter(drugTargetInteraction, alpha = 0.25)
linear_filter(drugTargetInteraction, alpha = c(0.1,0.1,0.4,0.4))
```

loo — *Leave-one-out cross-validation for tskrr*

Description

Perform a leave-one-out cross-validation for two-step kernel ridge regression based on the shortcuts described in Stock et al, 2018. (http://doi.org/10.1093/bib/bby095).

Usage

```
loo(x, ...)

## S4 method for signature 'tskrrHeterogeneous'
loo(
  x,
  exclusion = c("interaction", "row", "column", "both"),
  replaceby0 = FALSE
)

## S4 method for signature 'tskrrHomogeneous'
loo(
  x,
  exclusion = c("edges", "vertices", "interaction", "both"),
  replaceby0 = FALSE
)

## S4 method for signature 'linearFilter'
loo(x, replaceby0 = FALSE)
```

Arguments

`x`	an object of class `tskrr` or `linearFilter`.
`...`	arguments passed to methods. See Details.
`exclusion`	a character value with possible values "interaction", "row", "column", "both" for heterogeneous models, and "edges", "vertices", "interaction" or "both" for homogeneous models. Defaults to "interaction". See details.
`replaceby0`	a logical value indicating whether the interaction should be simply removed (`FALSE`) or replaced by 0 (`TRUE`).

Details

The parameter `exclusion` defines what is left out. The value "interaction" means that a single interaction is removed. In the case of a homogeneous model, this can be interpreted as the removal of the interaction between two edges. The values "row" and "column" mean that all interactions for a row edge resp. a column edge are removed. The value "both" removes all interactions for a row and a column edge.

In the case of a homogeneous model, "row" and "column" don't make sense and will be replaced by "both" with a warning. This can be interpreted as removing vertices, i.e. all interactions between one edge and all other edges. Alternatively one can use "edges" to remove edges and "vertices" to remove vertices. In the case of a homogeneous model, the setting "edges" translates to "interaction", and "vertices" translates to "both". For more information, see Stock et al. (2018).

Replacing by 0 only makes sense when `exclusion = "interaction"` and the label matrix contains only 0 and 1 values. The function checks whether the conditions are fulfilled and if not, returns an error.

Value

a numeric matrix with the leave-one-out predictions for the model.

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim,
             lambda = c(0.01,0.01))

delta <- loo(mod, exclusion = 'both') - response(mod)
delta0 <- loo(mod, replaceby0 = TRUE) - response(mod)
```

`loo_internal` *Leave-one-out cross-validation for two-step kernel ridge regression*

Description

These functions implement different cross-validation scenarios for two-step kernel ridge regression. It uses the shortcuts for leave-one-out cross-validation.

Usage

```
loo.i(Y, Hk, Hg, pred)

loo.i0(Y, Hk, Hg, pred)

loo.r(Y, Hk, Hg, ...)

loo.c(Y, Hk, Hg, ...)

loo.b(Y, Hk, Hg, ...)

loo.e.sym(Y, Hk, pred)

loo.e.skew(Y, Hk, pred)
loo.e0.sym(Y, Hk, pred)

loo.e0.skew(Y, Hk, pred)

loo.v(Y, Hk, ...)

loo.i.lf(Y, alpha, pred)

loo.i0.lf(Y, alpha, pred)
```

Arguments

`Y`	the matrix with responses
`Hk`	the hat matrix for the first kernel (rows of Y)
`Hg`	the hat matrix for the second kernel (columns of Y)
`pred`	the predictions
`...`	added to allow for specifying pred even when not needed.
`alpha`	a vector of length 4 with the alpha values from a `linearFilter` model

Details

These functions are primarily for internal use and hence not exported. Be careful when using them, as they do not perform any sanity check on the input. It is up to the user to make sure the input makes sense.

Value

a matrix with the leave-one-out predictions

See Also

`loo` for the user-level function.

`loss` *Calculate or extract the loss of a tskrr model*

Description

This function allows calculating the loss of a tskrr model using either one of the functions defined in `loss_functions` or a custom user function. If the model inherits from class `tskrrTune` and no additional arguments are given, the loss is returned for the settings used when tuning. The function can also be used to extract the original loss from a `permtest` object.

Usage

```
loss(x, ...)

## S4 method for signature 'tskrr'
loss(
  x,
  fun = loss_mse,
  exclusion = c("interaction", "row", "column", "both"),
  replaceby0 = FALSE,
  predictions = FALSE,
  ...
)

## S4 method for signature 'tskrrTune'
loss(
  x,
  fun = loss_mse,
  exclusion = c("interaction", "row", "column", "both"),
  replaceby0 = FALSE,
  predictions = FALSE,
  ...
)

## S4 method for signature 'permtest'
loss(x, ...)
```

Arguments

`x`	a model that inherits from class `tskrr`
`...`	extra arguments passed to the loss function in `fun`.
`fun`	a function to be used for calculating the loss. This can also be a character value giving the name of one of the loss functions provided in the package
`exclusion`	a character value with possible values "interaction", "row", "column" or "both". See also `loo` for more information.
`replaceby0`	a logical value indicating whether the interaction should be simply removed (`FALSE`) or replaced by 0 (`TRUE`).
`predictions`	a logical value to indicate whether the predictions should be used instead of leave one out crossvalidation. If set to `TRUE`, the other arguments are ignored.

Value

a numeric value with the calculated loss

See Also

- `loss_functions` for possible loss functions
- `tune` for tuning a model based on loss functions

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)

loss(mod, fun = loss_auc)

tuned <- tune(mod, fun = loss_auc)

loss(tuned)
loss(tuned, fun = loss_mse)
```

`loss_functions` *loss functions*

Description

These functions can be used as loss functions in `tune`. Currently, two functions are provided: a function calculating the classic mean squared error (`loss_mse`) and a function calculating 1 - AUC (`loss_auc`).

Usage

```
loss_mse(Y, LOO, na.rm = FALSE)

loss_auc(Y, LOO)
```

Arguments

`Y`	the label matrix with observed responses
`LOO`	the leave-one-out crossvalidation (or predictions if you must). This one can be calculated by the function `loo`.
`na.rm`	a logical value

Details

The AUC is calculated by sorting the `Y` matrix based on the order of the values in the `LOO` matrix. The false and true positive rates are calculated solely based on that ordering, which allows for values in `LOO` outside the range [0,1]. It's a naive implementation which is good enough for tuning, but shouldn't be used as a correct value for 1 - auc in case the values in `LOO` are outside the range [0,1].

Note

The function `loss_auc` should only be used for a `Y` matrix that contains solely the values 0 and 1.

See Also

tune for application of the loss function

Examples

```
x <- c(1,0,0,1,0,0,1,0,1)
y <- c(0.8,-0.1,0.2,0.2,0.4,0.01,1.12,0.9,0.9)
loss_mse(x,y)
loss_auc(x,y)
```

---

match_labels *Reorder the label matrix*

Description

Reorders the label matrix based on the labels of the kernel matrices. In case there are no labels, the original label matrix is returned, but with the labels in `rows` and `cols` as rownames and column names respectively.

Usage

```
match_labels(y, rows, cols = NULL)
```

Arguments

`y`	a matrix representing the label matrix.
`rows`	a character vector with the labels for the rows or a matrix with rownames that will be used as labels.
`cols`	a character vector with the labels for the cols or a matrix with colnames that will be used as labels. If `NULL`, `rows` will be used for both row and column labels.

Value

a matrix with the rows and columns reordered.

Examples

```
mat <- matrix(1:6, ncol = 2,
              dimnames = list(c("b", "a", "d"),
                              c("ca", "cb"))
             )

match_labels(mat, c("a","b", "d"), c("ca","cb"))

#Using matrices
data(drugtarget)
out <- match_labels(drugTargetInteraction, targetSim, drugSim)
```

permtest — *Calculate the relative importance of the edges*

Description

This function does a permutation-based evaluation of the impact of different edges on the final result. It does so by permuting the kernel matrices, refitting the model and calculating a loss function.

Usage

```
permtest(x, ...)

## S3 method for class 'permtest'
print(x, digits = max(3L, getOption("digits") - 3), ...)

## S4 method for signature 'tskrrHeterogeneous'
permtest(
  x,
  n = 100,
  permutation = c("both", "row", "column"),
  exclusion = c("interaction", "row", "column", "both"),
  replaceby0 = FALSE,
  fun = loss_mse,
  exact = FALSE
)

## S4 method for signature 'tskrrHomogeneous'
permtest(
  x,
  n = 100,
  permutation = c("both"),
  exclusion = c("interaction", "both"),
  replaceby0 = FALSE,
  fun = loss_mse,
  exact = FALSE
)

## S4 method for signature 'tskrrTune'
permtest(x, permutation = c("both", "row", "column"), n = 100)
```

Arguments

`x`	either a `tskrr-class` or a `tskrrTune-class` object
`...`	arguments passed to other methods
`digits`	the number of digits shown in the output
`n`	the number of permutations for every kernel matrix

`permutation`	a character string that defines whether the row, column or both kernel matrices should be permuted. Ignored in case of a homogeneous network
`exclusion`	the exclusion to be used in the `loo` function. See also `get_loo_fun`
`replaceby0`	a logical value indicating whether `loo` removes a value in the leave-one-out procedure or replaces it by zero. See also `get_loo_fun`.
`fun`	a function (or a character string with the name of a function) that calculates the loss. See also `tune` and `loss_functions`
`exact`	a logical value that indicates whether or not an exact p-value should be calculated, or be approximated based on a normal distribution.

Details

The test involved uses a normal approximation. It assumes that under the null hypothesis, the loss values are approximately normally distributed. The cumulative probability of a loss as small or smaller than the one found in the original model, is calculated based on a normal distribution from which the mean and sd are calculated from the permutations.

Value

An object of the class permtest.

Warning

It should be noted that this normal approximation is an ad-hoc approach. There's no guarantee that the actual distribution of the loss under the null hypothesis is normal. Depending on the loss function, a significant deviation from the theoretic distribution can exist. Hence this functions should only be used as a rough guidance in model evaluation.

Examples

```
# Heterogeneous network

data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)
permtest(mod, fun = loss_auc)
```

`permtest-class` *Class permtest*

Description

This class represents the permutation test outcomes. See also the function `permtest`.

Slots

`orig_loss` a numeric value with the original loss of the model.

`perm_losses` a numeric vector with the losses of the different permutations.

`n` the number of permutations

`loss_function` the function used to calculate the losses.

`exclusion` a character value indicating the exclusion setting used for the test

`replaceby0` a locigal value that indicates whether the exclusion was done by replacing with zero. See also `loo`.

`permutation` a character value that indicats in which kernel matrices were permuted.

`pval` a p value indicating how likely it is to find a smaller loss than the one of the model based on a normal approximation.

`exact` a logical value indicating whether the P value was calculated exactly or approximated by the normal distribution.

See Also

- the function `permtest` for the actual test.
- the function `loo` for the leave one out procedures
- the function `t.test` for the actual test

`permutations` *Getters for permtest objects*

Description

The functions described here are convenience functions to get information out of a `permtest` object.

Usage

```
permutations(x)

## S4 method for signature 'permtest'
x[i]
```

Arguments

`x`	a `permtest` object
`i`	either a numeric vector, a logical vector or a character vector with the elements that need extraction.

Value

the requested values

See Also

loss to extract the original loss value.

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)
ptest <- permtest(mod, fun = loss_auc)

loss(ptest)
ptest[c(2,3)]
permutations(ptest)
```

---

plot.tskrr *plot a heatmap of the predictions from a tskrr model*

Description

This function plots a heatmap of the fitted values in a tskrr model. The function is loosely based on heatmap, but uses a different mechanism and adds a legend by default.

Usage

```
## S3 method for class 'tskrr'
plot(
  x,
  dendro = c("both", "row", "col", "none"),
  which = c("fitted", "loo", "response", "residuals"),
  exclusion = c("interaction", "row", "column", "both"),
  replaceby0 = FALSE,
  nbest = 0,
  rows,
  cols,
  col = rev(heat.colors(20)),
  breaks = NULL,
  legend = TRUE,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  labRow = NULL,
  labCol = NULL,
  margins = c(5, 5),
  ...
)
```

Arguments

`x`	a tskrr model
`dendro`	a character value indicating whether a dendrogram should be constructed.
`which`	a character value indicating whether the fitted values, the leave-one-out values, the original response values or the residuals should be plotted.
`exclusion`	if `which = "loo"`, this argument is passed to `loo` for the exclusion settings
`replaceby0`	if `which = "loo"`, this argument is passed to `loo`.
`nbest`	a single integer value indicating the amount of best values that should be selected. If `0`, all data is shown.
`rows`	a numeric or character vector indicating which rows should be selected from the model.
`cols`	a numeric or character vector indicating which columns should be selected from the model.
`col`	a vector with colors to be used for plotting
`breaks`	a single value specifying the number of breaks (must be 1 more than number of colors), or a numeric vector with the breaks used for the color code. If `NULL`, the function tries to find evenly spaced breaks.
`legend`	a logical value indicating whether or not the legend should be added to the plot.
`main`	a character value with a title for the plot
`xlab`	a character label for the X axis
`ylab`	a character label for the Y axis
`labRow`	a character vector with labels to be used on the rows. Note that these labels are used as is (possibly reordered to match the dendrogram). They can replace the labels from the model. Set to `NA` to remove the row labels.
`labCol`	the same as `labRow` but then for the columns.
`margins`	a numeric vector with 2 values indicating the margins to be used for the row and column labels (cfr `par("mar")`)
`...`	currently ignored

Details

The function can select a part of the model for plotting. Either you specify `rows` and `cols`, or you specify `nbest`. If `nbest` is specified, `rows` and `cols` are ignored. The `n` highest values are looked up in the plotted values, and only the rows and columns related to these values are shown then. This allows for a quick selection of the highest predictions.

Dendrograms are created by converting the kernel matrices to a distance, using

d(x,y) = K(x,x)^2 + K(y,y)^2 - 2*K(x,y)

with K being the kernel function. The resulting distances are clustered using `hclust` and converted to a dendrogram using `as.dendrogram`.

Value

an invisible list with the following elements:

- `val`: the values plotted
- `ddK`: if a row dendrogram was requested, the row dendrogram
- `ddG`: if a column dendrogram was requested, the column dendrogram
- `breaks`: the breaks used for the color codes
- `col`: the colors used

See Also

`tskrr`, `tune` and `link{impute_tskrr}` to construct tskrr models.

Examples

```
data(drugtarget)
mod <- tskrr(drugTargetInteraction, targetSim, drugSim)

plot(mod)
plot(mod, dendro = "row", legend = FALSE)
plot(mod, col = rainbow(20), dendro = "none", which = "residuals")
plot(mod, labCol = NA, labRow = NA, margins = c(0.2,0.2))
```

plot_grid *Plot the grid of a tuned tskrr model*

Description

With this function, you can visualize the grid search for optimal lambdas from a `tskrrTune` object. In the case of two-dimensional grid search, this function plots a contour plot on a grid, based on the functions `image` and `contour`. For one-dimensional grid search, the function creates a single line plot.

Usage

```
plot_grid(
  x,
  addlambda = TRUE,
  lambdapars = list(col = "red"),
  log = TRUE,
  opts.contour = list(nlevels = 10),
  ...
)
```

Arguments

`x`	an object that inherits from `tskrrTune`
`addlambda`	a logical value indicating whether the lambda with the minimum loss should be added to the plot. In case of a one dimensional plot, this adds a colored vertical line. In the case of a two dimensional plot, this adds a colored point at the minimum.
`lambdapars`	a list with named `par` values passed to the function `abline` or `points` for plotting the best lambda value when `addmin = TRUE`.
`log`	a logical value indicating whether the lambdas should be plotted at a log scale (the default) or not.
`opts.contour`	options passed to the function `contour` for 2D grid plots. Ignored for 1D grid plots.
`...`	arguments passed to other functions. For a one dimensional plot, this will be the function `plot`

Value

`NULL` invisibly

Examples

```
data(drugtarget)

## One dimensional tuning
tuned1d <- tune(drugTargetInteraction, targetSim, drugSim,
                lim = c(1e-4,2), ngrid = 40,
                fun = loss_auc, onedim = TRUE)

plot_grid(tuned1d)
plot_grid(tuned1d, lambdapars = list(col = "green",
                                     lty = 1, lwd = 2),
          log = FALSE, las = 2, main = "1D tuning")

## Two dimensional tuning
tuned2d <- tune(drugTargetInteraction, targetSim, drugSim,
                lim = c(1e-4,10), ngrid = 20,
                fun = loss_auc)

plot_grid(tuned2d)
```

`predict.tskrr` *predict method for tskrr fits*

Description

Obtains the predictions from a `tskrr` model for new data. To get the predictions on the training data, use the function `fitted` or set both `k` and `g` to `NULL`.

Usage

```
## S3 method for class 'tskrr'
predict(object, k = NULL, g = NULL, testdim = TRUE, ...)

## S4 method for signature 'tskrr'
predict(object, k = NULL, g = NULL, testdim = TRUE, ...)
```

Arguments

`object`	an object of class `tskrr`.
`k`	a new K matrix or `NULL`. if `NULL`, the fitted values on the training data are returned.
`g`	a new G matrix or `NULL`. If `NULL`, K is used for both.
`testdim`	a logical value indicating whether the dimensions should be checked prior to the calculation. You can set this to `FALSE` but you might get more obscure errors if dimensions don't match.
`...`	arguments passed to or from other methods

Details

Predictions can be calculated between new vertices and the vertices used to train the model, between new sets of vertices, or both. Which predictions are given, depends on the kernel matrices passed to the function.

In any case, both the K and G matrix need the kernel values for every combination of the new vertices and the vertices used to train the model. This is illustrated for both homogeneous and heterogeneous networks in the examples.

To predict the links between a new set of vertices and the training vertices, you need to provide the kernel matrix for either the K or the G set of vertices. If you want to predict the mutual links between two new sets of vertices, you have to provide both the K and the G matrix. This is particularly important for homogeneous networks: if you only supply the `k` argument, you will get predictions for the links between the new vertices and the vertices on which the model is trained. So in order to get the mutual links between the new vertices, you need to provide the kernel matrix as the value for both the `k` and the `g` argument.

Value

a matrix with predicted values.

Warning

This function is changed in version 0.1.9 so it's more consistent in how it expects the K and G matrices to be ordered. Up to version 0.1.8 the new vertices should be on the rows for the K matrix and on the columns for the G matrix. This lead to confusion.

If you're using old code, you'll get an error pointing this out. You need to transpose the G matrix in the old code to make it work with the new version.

See Also

tskrr and tskrrTune for fitting the models.

Examples

```
## Predictions for homogeneous networks

data(proteinInteraction)

idnew <- sample(nrow(Kmat_y2h_sc), 20)

trainY <- proteinInteraction[-idnew,-idnew]
trainK <- Kmat_y2h_sc[-idnew,-idnew]

testK <- Kmat_y2h_sc[idnew, - idnew]

mod <- tskrr(trainY, trainK, lambda = 0.1)
# Predict interaction between test vertices
predict(mod, testK, testK)

# Predict interaction between test and train vertices
predict(mod, testK)
predict(mod, g = testK)

## Predictions for heterogeneous networks
data("drugtarget")

idnewK <- sample(nrow(targetSim), 10)
idnewG <- sample(ncol(drugSim), 10)

trainY <- drugTargetInteraction[-idnewK, -idnewG]
trainK <- targetSim[-idnewK, -idnewK]
trainG <- drugSim[-idnewG, -idnewG]

testK <- targetSim[idnewK, -idnewK]
testG <- drugSim[idnewG, -idnewG]

mod <- tskrr(trainY, trainK, trainG, lambda = 0.01)

# Predictions for new targets on drugs in model
predict(mod, testK)
# Predictions for new drugs on targets in model
predict(mod, g = testG)
# Predictions for new drugs and targets
predict(mod, testK, testG)
```

`proteinInteraction` *Protein interaction for yeast*

Description

A dataset for examining the interaction between proteins of yeast. The dataset consists of the following objects:

Usage

```
proteinInteraction
```

Format

- proteinInteraction: a numeric square matrix with 150 rows/columns
- Kmat_y2h_sc: a numeric square matrix with 150 rows/columns

Details

- proteinInteraction: the label matrix based on the protein network taken from the KEGG/PATHWAY database
- Kmat_y2h_sc: a kernel matrix indicating similarity of proteins.

The proteins in the dataset are a subset of the 769 proteins used in Yamanishi et al (2004). The kernel matrix used is the combination of 4 kernels: one based on expression data, one on protein interaction data, one on localization data and one on phylogenetic profile. These kernels and their combination are also explained in Yamanishi et al (2004).

Source

`https://doi.org/10.1093/bioinformatics/bth910`

References

Yamanishi et al, 2004: Protein network inference from multiple genomic data: a supervised approach.

`residuals` *calculate residuals from a tskrr model*

Description

This function returns the residuals for an object inheriting from class `tskrr`

Usage

```
residuals(object, ...)

## S3 method for class 'tskrr'
residuals(
  object,
  method = c("predictions", "loo"),
  exclusion = c("interaction", "row", "column", "both"),
  replaceby0 = FALSE,
  ...
)

## S4 method for signature 'tskrr'
residuals(
  object,
  method = c("predictions", "loo"),
  exclusion = c("interaction", "row", "column", "both"),
  replaceby0 = FALSE,
  ...
)
```

Arguments

`object`	a tskrr model
`...`	arguments passed from/to other methods.
`method`	a character value indicating whether the residuals should be based on the predictions or on a leave-one-out crossvalidation.
`exclusion`	a character value with possible values "interaction", "row", "column", "both" for heterogeneous models, and "edges", "vertices", "interaction" or "both" for homogeneous models. Defaults to "interaction". See details.
`replaceby0`	a logical value indicating whether the interaction should be simply removed (`FALSE`) or replaced by 0 (`TRUE`).

Details

The parameter `exclusion` defines what is left out. The value "interaction" means that a single interaction is removed. In the case of a homogeneous model, this can be interpreted as the removal of the interaction between two edges. The values "row" and "column" mean that all interactions for a row edge resp. a column edge are removed. The value "both" removes all interactions for a row and a column edge.

In the case of a homogeneous model, "row" and "column" don't make sense and will be replaced by "both" with a warning. This can be interpreted as removing vertices, i.e. all interactions between one edge and all other edges. Alternatively one can use "edges" to remove edges and "vertices" to remove vertices. In the case of a homogeneous model, the setting "edges" translates to "interaction", and "vertices" translates to "both". For more information, see Stock et al. (2018).

Replacing by 0 only makes sense when `exclusion = "interaction"` and the label matrix contains only 0 and 1 values. The function checks whether the conditions are fulfilled and if not, returns an error.

Value

a matrix(!) with the requested residuals

Examples

```
data(drugtarget)
mod <- tskrr(drugTargetInteraction, targetSim, drugSim,
             lambda = c(0.01,0.01))
delta <- response(mod) - loo(mod, exclusion = "both")
resid <- residuals(mod, method = "loo", exclusion = "both")
all.equal(delta, resid)
```

`response,tskrr-method` *Getters for tskrr objects*

Description

The functions described here are convenience functions to get information out of a `tskrr` object.

Usage

```
## S4 method for signature 'tskrr'
response(x, ...)

## S4 method for signature 'tskrrHomogeneous'
lambda(x)

## S4 method for signature 'tskrrHeterogeneous'
lambda(x)

is_tskrr(x)

is_homogeneous(x)

is_heterogeneous(x)

symmetry(x)

get_eigen(x, which = c("row", "column"))

get_kernelmatrix(x, which = c("row", "column"))
```

```
has_hat(x)

get_kernel(x, which = c("row", "column"))
```

Arguments

x	a tskrr object or an object inheriting from tskrr.
...	arguments passed to other methods.
which	a character value indicating whether the eigen decomposition for the row kernel matrix or the column kernel matrix should be returned.

Value

For `response`: the original label matrix

For `lambda`: a named numeric vector with one resp both lambda values used in the model. The names are "k" and "g" respectively.

For `is_tskrr` a logical value indicating whether the object is a `tskrr` object

For `is_homogeneous` a logical value indicating whether the tskrr model is a homogeneous one.

For `is_heterogeneous` a logical value indicating whether the tskrr model is a heterogeneous one.

For `symmetry` a character value indicating the symmetry for a `homogeneous model`. If the model is not homogeneous, `NA` is returned.

For `get_eigen` the eigen decomposition of the requested kernel matrix.

For `get_kernelmatrix` the original kernel matrix for the rows or columns.

For `has_hat` a logical value indicating whether the tskrr model contains the kernel hat matrices.

Warning

The function `get_kernel` is deprecated. Use `get_kernelmatrix` instead.

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)
is_homogeneous(mod)

EigR <- get_eigen(mod)
EigC <- get_eigen(mod, which = 'column')
lambda(mod)
```

test_symmetry *test the symmetry of a matrix*

Description

This function tells you whether a matrix is symmetric, skewed symmetric, or not symmetric. It's used by tskrr to determine which kind of homologous network is represented by the label matrix.

Usage

```
test_symmetry(x, tol = .Machine$double.eps)
```

Arguments

x	a matrix
tol	a single numeric value with the tolerance for comparison

Value

a character value with the possible values "symmetric", "skewed" or "none".

See Also

tskrrHomogeneous for more information on the values for the slot symmetry

Examples

```
mat1 <- matrix(c(1,0,0,1),ncol = 2)
test_symmetry(mat1)
mat2 <- matrix(c(1,0,0,-1), ncol = 2)
test_symmetry(mat2)
mat3 <- matrix(1:4, ncol = 2)
test_symmetry(mat3)
```

tskrr *Fitting a two step kernel ridge regression*

Description

tskrr is the primary function for fitting a two-step kernel ridge regression model. It can be used for both homogeneous and heterogeneous networks.

Usage

```
tskrr(
  y,
  k,
  g = NULL,
  lambda = 1e-04,
  testdim = TRUE,
  testlabels = TRUE,
  symmetry = c("auto", "symmetric", "skewed"),
  keep = FALSE
)
```

Arguments

`y`	a label matrix
`k`	a kernel matrix for the rows
`g`	an optional kernel matrix for the columns
`lambda`	a numeric vector with one or two values for the hyperparameter lambda. If two values are given, the first one is used for the k matrix and the second for the g matrix.
`testdim`	a logical value indicating whether symmetry and the dimensions of the kernel(s) should be tested. Defaults to `TRUE`, but for large matrices putting this to `FALSE` will speed up the function.
`testlabels`	a logical value indicating wether the row- and column names of the matrices have to be checked for consistency. Defaults to `TRUE`, but for large matrices putting this to `FALSE` will speed up the function.
`symmetry`	a character value with the possibilities "auto", "symmetric" or "skewed". In case of a homogeneous fit, you can either specify whether the label matrix is symmetric or skewed, or you can let the function decide (option "auto").
`keep`	a logical value indicating whether the kernel hat matrices should be stored in the model object. Doing so makes the model object quite larger, but can speed up predictions in some cases. Defaults to `FALSE`.

Value

a `tskrr` object

See Also

`response`, `fitted`, `get_eigen`, `eigen2hat`

Examples

```
# Heterogeneous network

data(drugtarget)
```

```
mod <- tskrr(drugTargetInteraction, targetSim, drugSim)

Y <- response(mod)
pred <- fitted(mod)

# Homogeneous network

data(proteinInteraction)

modh <- tskrr(proteinInteraction, Kmat_y2h_sc)

Yh <- response(modh)
pred <- fitted(modh)
```

`tskrr-class` *Class tskrr*

Description

The class tskrr represents a two step kernel ridge regression fitting object, and is normally generated by the function `tskrr`. This is a superclass so it should not be instantiated directly.

Slots

`y` the matrix with responses

`k` the eigen decomposition of the kernel matrix for the rows

`lambda.k` the lambda value used for k

`pred` the matrix with the predictions

`has.hat` a logical value indicating whether the kernel hat matrices are stored in the object.

`Hk` the kernel hat matrix for the rows.

`labels` a list with two character vectors, `k` and `g`, containing the labels for the rows resp. columns. See `tskrrHomogeneous` and `tskrrHeterogeneous` for more details.

See Also

the classes `tskrrHomogeneous` and `tskrrHeterogeneous` for the actual classes.

`tskrr.fit` *Carry out a two-step kernel ridge regression*

Description

This function provides an interface for two-step kernel ridge regression. To use this function, you need at least one kernel matrix and one label matrix. It's the internal engine used by the function `tskrr`.

Usage

```
tskrr.fit(y, k, g = NULL, lambda.k = NULL, lambda.g = NULL, ...)
```

Arguments

`y`	a matrix representing the links between the nodes of both networks.
`k`	an object of class `eigen` containing the eigen decomposition of the first kernel matrix.
`g`	an optional object of class `eigen` containing the eigen decomposition of the second kernel matrix. If `NULL`, the network is considered to be homogeneous.
`lambda.k`	a numeric value for the lambda parameter tied to the first kernel.
`lambda.g`	a numeric value for the lambda parameter tied to the second kernel. If `NULL`, the model is fit using the same value for `lambda.k` and `lambda.g`
`...`	arguments passed to other functions. Currently ignored.

Details

This function is mostly available for internal use. In most cases, it makes much more sense to use `tskrr`, as that function returns an object one can work with. The function `tskrr.fit` could be useful when doing simulations or fitting algorithms, as the information returned from this function is enough to use the functions returned by `get_loo_fun`.

Value

a list with three elements:

- k : the hat matrix for the rows
- g : the hat matrix for the columns (or `NULL`) for homogeneous networks.
- pred : the predictions

Examples

```
data(drugtarget)

K <- eigen(targetSim)
G <- eigen(drugSim)

res <- tskrr.fit(drugTargetInteraction,K,G,
                 lambda.k = 0.01, lambda.g = 0.05)
```

`tskrrHeterogeneous-class` *Class tskrrHeterogeneous*

Description

The class tskrrHeterogeneous is a subclass of the superclass `tskrr` specifically for heterogeneous networks.

Slots

`y` the matrix with responses

`k` the eigen decomposition of the kernel matrix for the rows

`lambda.k` the lambda value used for k

`pred` the matrix with the predictions

`g` the eigen decomposition of the kernel matrix for the columns

`lambda.g` the lambda value used for g

`has.hat` a logical value indicating whether the kernel hat matrices are stored in the object.

`Hk` the kernel hat matrix for the rows.

`Hg` the kernel hat matrix for the columns.

`labels` a list with elements `k` and `g` (see `tskrr-class`). If any element is `NA`, the labels used are integers indicating the row resp column number.

`tskrrHomogeneous-class` *Class tskrrHomogeneous*

Description

The class tskrrHomogeneous is a subclass of the superclass `tskrr` specifically for homogeneous networks.

Slots

`y` the matrix with responses

`k` the eigen decomposition of the kernel matrix for the rows

`lambda.k` the lambda value used for k

`pred` the matrix with the predictions

`symmetry` a character value that can have the possible values `"symmetric"`, `"skewed"` or `"not"`. It indicates whether the `y` matrix is symmetric, skewed-symmetric or not symmetric.

`has.hat` a logical value indicating whether the kernel hat matrices are stored in the object.

`Hk` the kernel hat matrix for the rows.

`labels` a list with elements `k` and `g` (see `tskrr-class`). For homogeneous networks, `g` is always `NA`. If `k` is `NA`, the labels used are integers indicating the row resp column number.

`tskrrImpute-class` *Class tskrrImpute*

Description

The class `tskrrImpute` is a virtual class that represents a `tskrr` model with imputed values in the label matrix Y. Apart from the model, it contains the following extra information on the imputed values.

Slots

`imputeid` a vector with integer values indicating which of the values in `y` are imputed

`niter` an integer value gving the number of iterations used

`tol` a numeric value with the tolerance used

`tskrrImputeHeterogeneous-class` *Class tskrrImputeHeterogeneous*

Description

The class `tskrrImputeHeterogeneous` is a subclass of the class `tskrrHeterogeneous` and `tskrrImpute` specifically for heterogeneous networks with imputed values. It is the result of the function `impute_tskrr`.

Slots

`y` the matrix with responses

`k` the eigen decomposition of the kernel matrix for the rows

`lambda.k` the lambda value used for k

`pred` the matrix with the predictions

`g` the eigen decomposition of the kernel matrix for the columns

`lambda.g` the lambda value used for g

`has.hat` a logical value indicating whether the kernel hat matrices are stored in the object.

`Hk` the kernel hat matrix for the rows.

`Hg` the kernel hat matrix for the columns.

`labels` a list with elements `k` and `g` (see `tskrr-class`). If any element is `NA`, the labels used are integers indicating the row resp column number.

`imputeid` a vector with integer values indicating which of the values in `y` are imputed

`niter` an integer value gving the number of iterations used

`tol` a numeric value with the tolerance used

`tskrrImputeHomogeneous-class` *Class tskrrImputeHomogeneous*

Description

The class `tskrrImputeHomogeneous` is a subclass of the class `tskrrHomogeneous` and `tskrrImpute` specifically for homogeneous networks with imputed values. It is the result of the function `impute_tskrr` on a homogeneous network model.

Slots

`y` the matrix with responses

`k` the eigen decomposition of the kernel matrix for the rows

`lambda.k` the lambda value used for k

`pred` the matrix with the predictions

`symmetry` a character value that can have the possible values `"symmetric"`, `"skewed"` or `"not"`. It indicates whether the `y` matrix is symmetric, skewed-symmetric or not symmetric.

`has.hat` a logical value indicating whether the kernel hat matrices are stored in the object.

`Hk` the kernel hat matrix for the rows.

`labels` a list with elements `k` and `g` (see `tskrr-class`). For homogeneous networks, `g` is always `NA`. If `k` is `NA`, the labels used are integers indicating the row resp column number.

`imputeid` a vector with integer values indicating which of the values in `y` are imputed

`niter` an integer value gving the number of iterations used

`tol` a numeric value with the tolerance used

`tskrrTune-class` *Class tskrrTune*

Description

The class tskrrTune represents a tuned `tskrr` model, and is the output of the function `tune`. Apart from the model, it contains extra information on the tuning procedure. This is a virtual class only.

Slots

`lambda_grid` a list object with the elements `k` and possibly `g` indicating the tested lambda values for the row kernel `K` and - if applicable - the column kernel `G`. Both elements have to be numeric.

`best_loss` a numeric value with the loss associated with the best lambdas

`loss_values` a matrix with the loss results from the searched grid. The rows form the X dimension (related to the first lambda), the columns form the Y dimension (related to the second lambda if applicable)

`loss_function` the used loss function

`exclusion` a character value describing the exclusion used

`replaceby0` a logical value indicating whether or not the cross validation replaced the excluded values by zero

`onedim` a logical value indicating whether the grid search was done in one dimension. For homogeneous networks, this is true by default.

See Also

- the function `tune` for the tuning itself
- the class `tskrrTuneHomogeneous` and `tskrrTuneHeterogeneous` for the actual classes.

---

tskrrTuneHeterogeneous-class *Class tskrrTuneHeterogeneous*

Description

The class tskrrTuneHeterogeneous represents a tuned Heterogeneous `tskrr` model. It inherits from the classes `tskrrHeterogeneous` and `tskrrTune`.

---

tskrrTuneHomogeneous-class *Class tskrrTuneHomogeneous*

Description

The class tskrrTuneHomogeneous represents a tuned homogeneous `tskrr` model. It inherits from the classes `tskrrHomogeneous` and `tskrrTune`.

---

tune *tune the lambda parameters for a tskrr*

Description

This function lets you tune the lambda parameter(s) of a two-step kernel ridge regression model for optimal performance. You can either tune a previously fitted `tskrr` model, or pass the label matrix and kernel matrices to fit and tune a model in one go.

Usage

```
## S4 method for signature 'tskrrHomogeneous'
tune(
  x,
  lim = c(1e-04, 1),
  ngrid = 10,
  lambda = NULL,
  fun = loss_mse,
  exclusion = "edges",
  replaceby0 = FALSE,
  onedim = TRUE,
  ...
)

## S4 method for signature 'tskrrHeterogeneous'
```

```
tune(
  x,
  lim = c(1e-04, 1),
  ngrid = 10,
  lambda = NULL,
  fun = loss_mse,
  exclusion = "interaction",
  replaceby0 = FALSE,
  onedim = FALSE,
  ...
)

## S4 method for signature 'matrix'
tune(
  x,
  k,
  g = NULL,
  lim = c(1e-04, 1),
  ngrid = 10,
  lambda = NULL,
  fun = loss_mse,
  exclusion = "interaction",
  replaceby0 = FALSE,
  testdim = TRUE,
  testlabels = TRUE,
  symmetry = c("auto", "symmetric", "skewed"),
  keep = FALSE,
  onedim = is.null(g),
  ...
)
```

Arguments

`x`	a `tskrr` object representing a two step kernel ridge regression model.
`lim`	a vector with 2 values that give the boundaries for the domain in which lambda is searched, or possibly a list with 2 elements. See details
`ngrid`	a single numeric value giving the number of points in a single dimension of the grid, or possibly a list with 2 elements. See details.
`lambda`	a vector with the lambdas that need checking for homogeneous networks, or possibly a list with two elements for heterogeneous networks. See Details. Defaults to `NULL`, which means that the function constructs the search grid from the other arguments.
`fun`	a loss function that takes the label matrix Y and the result of the crossvalidation LOO as input. The function name can be passed as a character string as well.
`exclusion`	a character value with possible values "interaction", "row", "column", "both" for heterogeneous models, and "edges", "vertices", "interaction" or "both" for homogeneous models. Defaults to "interaction". See details.

`replaceby0`	a logical value indicating whether the interaction should be simply removed (`FALSE`) or replaced by 0 (`TRUE`).
`onedim`	a logical value indicating whether the search should be done in a single dimension. See details.
`...`	arguments to be passed to the loss function
`k`	a kernel matrix for the rows
`g`	an optional kernel matrix for the columns
`testdim`	a logical value indicating whether symmetry and the dimensions of the kernel(s) should be tested. Defaults to `TRUE`, but for large matrices putting this to `FALSE` will speed up the function.
`testlabels`	a logical value indicating wether the row- and column names of the matrices have to be checked for consistency. Defaults to `TRUE`, but for large matrices putting this to `FALSE` will speed up the function.
`symmetry`	a character value with the possibilities "auto", "symmetric" or "skewed". In case of a homogeneous fit, you can either specify whether the label matrix is symmetric or skewed, or you can let the function decide (option "auto").
`keep`	a logical value indicating whether the kernel hat matrices should be stored in the model object. Doing so makes the model object quite larger, but can speed up predictions in some cases. Defaults to `FALSE`.

Details

This function currently only performs a simple grid search for all (combinations of) lambda values. If no specific lambda values are provided, then the function uses `create_grid` to create an evenly spaced (on a logarithmic scale) grid.

In the case of a heterogeneous network, you can specify different values for the two parameters that need tuning. To do so, you need to provide a list with the settings for every parameter to the arguments `lim`, `ngrid` and/or `lambda`. If you try this for a homogeneous network, the function will return an error.

Alternatively, you can speed up the grid search by searching in a single dimension. When `onedim = TRUE`, the search for a heterogeneous network will only consider cases where both lambda values are equal.

The arguments `exclusion` and `replaceby0` are used by the function `get_loo_fun` to find the correct leave-one-out function.

By default, the function uses standard mean squared error based on the cross-validation results as a measure for optimization. However, you can provide a custom function if needed, as long as it takes two matrices as input: `Y` being the observed interactions and `LOO` being the result of the chosen cross-validation.

Value

a model of class `tskrrTune`

See Also

- loo, loo_internal and get_loo_fun for more information on how leave one out validation works.
- tskrr for fitting a twostep kernel ridge regression.
- loss_functions for different loss functions.

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)
tuned <- tune(mod, lim = c(0.1,1), ngrid = list(5,10),
              fun = loss_auc)

## Not run:

# This is just some visualization of the matrix
# It can be run safely.
gridvals <- get_grid(tuned)
z <- get_loss_values(tuned)        # loss values

image(gridvals$k,gridvals$g,z, log = 'xy',
      xlab = "lambda k", ylab = "lambda g",
      col = rev(heat.colors(20)))


## End(Not run)
```

---

update *Update a tskrr object with a new lambda*

Description

This function allows you to refit a tskrr with a new lambda. It can be used to do manual tuning/cross-validation. If the object has the hat matrices stored, these are updated as well.

Usage

```
update(object, ...)

## S4 method for signature 'tskrrHomogeneous'
update(object, lambda)

## S4 method for signature 'tskrrHeterogeneous'
update(object, lambda)
```

Arguments

`object`	a `tskrr` object
`...`	arguments passed to methods
`lambda`	a numeric vector with one or two values for the hyperparameter lambda. If two values are given, the first one is used for the k matrix and the second for the g matrix.

Value

an updated `tskrr` object fitted with the new lambdas.

Examples

```
data(drugtarget)

mod <- tskrr(drugTargetInteraction, targetSim, drugSim)

# Update with the same lambda
mod2 <- update(mod, lambda = 1e-3)

# Use different lambda for rows and columns
mod3 <- update(mod, lambda = c(0.01,0.001))

# A model with the hat matrices stored
lambda <- c(0.001,0.01)
modkeep <- tskrr(drugTargetInteraction, targetSim, drugSim, keep = TRUE)
Hk_1 <- hat(modkeep, which = "row")
modkeep2 <- update(modkeep, lambda = lambda)
Hk_2 <- hat(modkeep2, which = "row")

# Calculate new hat matrix by hand:
decomp <- get_eigen(modkeep, which = "row")
Hk_byhand <- eigen2hat(decomp$vectors,
                       decomp$values,
                       lambda = lambda[1])
identical(Hk_2, Hk_byhand)
```

`valid_dimensions` *Functions to check matrices*

Description

These functions allow you to check whether the dimensions of the label matrix and the kernel matrix (matrices) are compatible. `valid_dimensions` checks whether both k and g are square matrices, whether y has as many rows as k and whether y has as many columns as g. `is_square` checks whether both dimensions are the same.

Usage

```
valid_dimensions(y, k, g = NULL)

is_square(x)
```

Arguments

`y`	a label matrix
`k`	a kernel matrix
`g`	an optional second kernel matrix or `NULL` otherwise.
`x`	any matrix

Value

a logical value indicating whether the dimensions of the matrices are compatible for a two step kernel ridge regression.

Note

The function `is_square` is not exported

`valid_labels` *Test the correctness of the labels.*

Description

This function checks whether the labels between the Y, K, and G matrices make sense. This means that all the labels found as rownames for `y` can be found as rownames *and* column names of `k`, and all the colnames for `y` can be found as rownames *and* colnames of `g` (if provided).

Usage

```
valid_labels(y, k, g = NULL)
```

Arguments

`y`	the label matrix
`k`	the kernel matrix for the rows
`g`	the kernel matrix for the columns (optional). If not available, it takes the value `NULL`

Details

Compatible labels mean that it is unequivocally clear which rows and columns can be linked throughout the model. In case none of the matrices have row- or colnames, the labels are considered compatible. In all other cases, all matrices should have both row and column names. They should fulfill the following conditions:

- the row- and column names of a kernel matrix must contain the same values in the same order. Otherwise, the matrix can't be symmetric.
- the rownames of `y` should correspond to the rownames of `k`
- the colnames of `y` should correspond to the colnames of `g` if it is supplied, or the colnames of `k` in case `g` is `NULL`

Value

`TRUE` if all labels are compatible, an error otherwise.

Note

This is a non-exported convenience function.

`weights,tskrrHeterogeneous-method`

Extract weights from a tskrr model

Description

This function calculates the weight matrix for calculating the predictions of a tskrr model.

Usage

```
## S4 method for signature 'tskrrHeterogeneous'
weights(object)

## S4 method for signature 'tskrrHomogeneous'
weights(object)
```

Arguments

`object`	a `tskrr` object for which the weights have to be calculated.

Details

The weight matrix is calculated from the map matrices through the function `eigen2map`.

Value

a matrix with the weights for the tskrr model.

Note

The package `xnet` adds a S4 generic function for `weights`.

Index