Package ‘sperrorest’

October 16, 2022

Type Package

Title Perform Spatial Error Estimation and Variable Importance Assessment

Version 3.0.5

Description Implements spatial error estimation and permutation-based variable importance measures for predictive models using spatial cross-validation and spatial block bootstrap.

License GPL-3


BugReports https://github.com/giscience-fsu/sperrorest/issues

Depends R (>= 2.10)
Imports dplyr, future, future.apply, graphics, ROCR, stats, stringr
Suggests knitr, MASS, nnet, parallel, ranger, rmarkdown, rpart, sp, testthat

VignetteBuilder knitr

ByteCompile true

Encoding UTF-8

LazyData true

LazyLoad yes

RoxygenNote 7.2.0

NeedsCompilation no

Author Alexander Brenning [aut, cre] (https://orcid.org/0000-0001-6640-679X), Patrick Schratz [aut] (https://orcid.org/0000-0003-0748-6624), Tobias Herrmann [ctb] (https://orcid.org/0000-0001-9768-0708)

Maintainer Alexander Brenning <alexander.brenning@uni-jena.de>

Repository CRAN

Date/Publication 2022-10-16 12:50:02 UTC
Description

This package implements spatial error estimation and permutation-based spatial variable importance using different spatial cross-validation and spatial block bootstrap methods. To cite ‘sperrorest’ in publications, reference the paper by Brenning (2012).
References


add.distance

Add distance information to resampling objects

Description

Add distance information to resampling objects

Usage

add.distance(object, ...)

## S3 method for class 'resampling'
add.distance(object, data, coords = c("x", "y"), ...)

## S3 method for class 'represampling'
add.distance(object, data, coords = c("x", "y"), mode = "future", ...)

Arguments

- object: resampling or represampling object.
- ...: Additional arguments to dataset_distance and add.distance.resampling, respectively.
- data: data.frame containing at least the columns specified by coords (ignored by partition_cv)
- mode: Use future.apply::future_lapply() for parallelized execution if mode = "future", and lapply for sequential execution otherwise (mode = "sequential")
Details

Nearest-neighbour distances are calculated for each sample in the test set. These \texttt{nrow(???$test)} nearest-neighbour distances are then averaged. Aggregation methods other than \texttt{mean} can be chosen using the \texttt{fun} argument, which will be passed on to \texttt{dataset_distance}.

Value

A \texttt{resampling} or \texttt{represampling} object containing an additional \texttt{$distance} component in each \texttt{resampling} object. The \texttt{distance} component is a single numeric value indicating, for each train / test pair, the (by default, mean) nearest-neighbour distance between the two sets.

See Also

dataset_distance represampling resampling

Examples

# Muenchow et al. (2012), see ?ecuador
nsp.parti <- partition_cv(ecuador)
sp.parti <- partition_kmeans(ecuador)
nsp.parti <- add.distance(nsp.parti, data = ecuador)
sp.parti <- add.distance(sp.parti, data = ecuador)
# non-spatial partitioning: very small test-training distance:
nsp.parti[1][1]$$distance
# spatial partitioning: more substantial distance, depending on number of
# folds etc.
sp.parti[1][1]$$distance
Arguments

object    object of class represampling, or a list to be coerced to this class.
...        currently not used.
x         object of class represampling.

Details

represampling objects are (names) lists of resampling objects. Such objects are typically created by partition_cv, partition_kmeans, represampling_disc_bootstrap and related functions.

In r-repeated k-fold cross-validation, for example, the corresponding represampling object has length r, and each of its r resampling objects has length k.

as.resampling_list coerces object to class represampling while coercing its elements to resampling objects. Some validity checks are performed.

Value

as.resampling methods return an object of class represampling with the contents of object.

See Also

resampling, partition_cv, partition_kmeans, represampling_disc_bootstrap, etc.

Examples

# Muenchow et al. (2012), see ?ecuador
# Partitioning by elevation classes in 200 m steps:
fac <- factor(as.character(floor(ecuador$dem / 300)))
summary(fac)
parti <- as.resampling(fac)
# a list of lists specifying sets of training and test sets,
# using each factor at a time as the test set:
str(parti)
summary(parti)
Usage

as.resampling(object, ...)

## Default S3 method:
as.resampling(object, ...)

## S3 method for class 'factor'
as.resampling(object, ...)

## S3 method for class 'list'
as.resampling(object, ...)

validate.resampling(object)

is.resampling(x, ...)

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

Arguments

object depending on the function/method, a list or a vector of type factor defining a partitioning of the dataset.

... currently not used.

x object of class resampling.

Details

A resampling object is a list of lists defining a set of training and test samples.

In the case of k-fold cross-validation partitioning, for example, the corresponding resampling object would be of length k, i.e. contain k lists. Each of these k lists defines a training set of size n(k-1)/k (where n is the overall sample size), and a test set of size n/k. The resampling object does, however, not contain the data itself, but only indices between 1 and n identifying the selection (see Examples).

Another example is bootstrap resampling. represent_sampling_bootstrapping with argument oob = TRUE generates represent_sampling objects with indices of a bootstrap sample in the train component and indices of the out-of-bag sample in the test component (see Examples below).

as.resampling.factor: For each factor level of the input variable, as.resampling.factor determines the indices of samples in this level (= test samples) and outside this level (= training samples). Empty levels of object are dropped without warning.

as.resampling_list checks if the list in object has a valid resampling object structure (with components train and test etc.) and assigns the class attribute 'resampling' if successful.

Value

as.resampling methods: An object of class resampling.
See Also

represampling, partition_cv, partition_kmeans, represampling_bootstrap, etc.

Examples

# Muenchow et al. (2012), see ?ecuador

# Partitioning by elevation classes in 200 m steps:
parti <- factor(as.character(floor(ecuador$dem / 200)))
smp <- as.resampling(parti)
summary(smp)
# Compare:
summary(parti)

# k-fold (non-spatial) cross-validation partitioning:
parti <- partition_cv(ecuador)
parti <- parti[[1]] # the first (and only) resampling object in parti
# data corresponding to the test sample of the first fold:
str(ecuador[parti[[1]]$test, ])
# the corresponding training sample - larger:
str(ecuador[parti[[1]]$train, ])

# Bootstrap training sets, out-of-bag test sets:
parti <- represampling_bootstrap(ecuador, oob = TRUE)
parti <- parti[[1]] # the first (and only) resampling object in parti
# out-of-bag test sample: approx. one-third of nrow(ecuador):
str(ecuador[parti[[1]]$test, ])
# bootstrap training sample: same size as nrow(ecuador):
str(ecuador[parti[[1]]$train, ])

---

**as.tilename**

Alphanumeric tile names

Description

Functions for generating and handling alphanumeric tile names of the form `'X2:Y7'` as used by `partition_tiles` and `represampling_tile_bootstrap`.

Usage

as.tilename(x, ...)

## S3 method for class 'numeric'
as.tilename(x, ...)

## S3 method for class 'tilename'
as.character(x, ...)

## S3 method for class 'tilename'
dataset_distance

Calculate mean nearest-neighbour distance between point datasets

dataset_distance(d1, d2, x_name = "x", y_name = "y", fun = mean, method = "euclidean", ...)
Arguments

d1 a data.frame with (at least) columns with names given by x_name and y_name; these contain the x and y coordinates, respectively.
d2 see d1 - second set of points
x_name name of column in d1 and d2 containing the x coordinates of points.
y_name same for y coordinates
fun function to be applied to the vector of nearest-neighbor distances of d1 from d2.
method type of distance metric to be used; only ‘euclidean’ is currently supported.
... additional arguments to fun.

Details

Nearest-neighbour distances are calculated for each point in d1, resulting in a vector of length nrow(d1), and fun is applied to this vector.

Value

depends on fun; typically (e.g., mean) a numeric vector of length 1

See Also

add.distance

Examples

df <- data.frame(x = rnorm(100), y = rnorm(100))
dataset_distance(df, df) # == 0

Description

Calculate a variety of accuracy measures from observations and predictions of numerical and categorical response variables.

Usage

err_default(obs, pred)

Arguments

obs factor, logical, or numeric vector with observations
pred factor, logical, or numeric vector with predictions. Must be of same type as obs with the exception that pred may be numeric if obs is factor or logical ('soft' classification).
Value

A list with (currently) the following components, depending on the type of prediction problem:

- ‘hard’ classification: Misclassification error, overall accuracy; if two classes, sensitivity, specificity, positive predictive value (PPV), negative predictive value (NPV), kappa
- ‘soft’ classification: area under the ROC curve, error and accuracy at a obs>0.5 dichotomization, false-positive rate (FPR; 1-specificity) at 70, 80 and 90 percent sensitivity, true-positive rate (sensitivity) at 80, 90 and 95 percent specificity.
- regression: Bias, standard deviation, mean squared error, MAD (mad), median, interquartile range (IQR) of residuals

Note

NA values are currently not handled by this function, i.e. they will result in an error.

See Also

ROCR

Examples

obs <- rnorm(1000)
# Two mock (soft) classification examples:
err_default(obs > 0, rnorm(1000)) # just noise
err_default(obs > 0, obs + rnorm(1000)) # some discrimination
# Three mock regression examples:
err_default(obs, rnorm(1000)) # just noise, but no bias
err_default(obs, obs + rnorm(1000)) # some association, no bias
err_default(obs, obs + 1) # perfect correlation, but with bias

get_small_tiles

Identify small partitions that need to be fixed.

Description

get_small_tiles identifies partitions (tiles) that are too small according to some defined criterion / criteria (minimum number of samples in tile and/or minimum fraction of entire dataset).

Usage

get_small_tiles(tile, min_n = NULL, min_frac = 0, ignore = c())
**partition_cv**

Partition the data for a (non-spatial) cross-validation

**Description**

`partition_cv` creates a represampling object for `length(repetition)`-repeated `nfold`-fold cross-validation.

**Arguments**

- **tile**: factor: tile/partition names for all samples; names must be coercible to class `tilename`, i.e. of the form 'X4:Y2' etc.
- **min_n**: integer (optional): minimum number of samples per partition
- **min_frac**: numeric >0, <1: minimum relative size of partition as percentage of sample.
- **ignore**: character vector: names of tiles to be ignored, i.e. to be retained even if the inclusion criteria are not met.

**Value**

character vector: names of tiles that are considered 'small' according to these criteria

**See Also**

`partition_tiles, tilename`

**Examples**

```r
# Muenchow et al. (2012), see ?ecuador
# Rectangular partitioning without removal of small tiles:
parti <- partition_tiles(ecuador, nsplit = c(10, 10), reassign = FALSE)
summary(parti)
length(parti[[1]])
# Same in factor format for the application of get_small_tiles:
parti_fac <- partition_tiles(ecuador, nsplit = c(10, 10), reassign = FALSE,
                           return_factor = TRUE)
get_small_tiles(parti_fac[[1]], min_n = 20) # tiles with less than 20 samples
parti2 <- partition_tiles(ecuador, nsplit = c(10, 10), reassign = TRUE,
                          min_n = 20, min_frac = 0)
length(parti2[[1]]) # < length(parti[[1]])
```
Usage

```r
partition_cv(
  data,
  coords = c("x", "y"),
  nfold = 10,
  repetition = 1,
  seed1 = NULL,
  return_factor = FALSE
)
```

Arguments

data  data.frame containing at least the columns specified by coords
coords (ignored by partition_cv)
nfold  number of partitions (folds) in nfold-fold cross-validation partitioning
repetition numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use repetition = c(1:100) to obtain (the 'first') 100 repetitions, and repetition = c(101:200) to obtain a different set of 100 repetitions.
seed1  seed1+i is the random seed that will be used by set.seed in repetition i (i in repetition) to initialize the random number generator before sampling from the data set.
return_factor  if FALSE (default), return a represampling object; if TRUE (used internally by other sperrorest functions), return a list containing factor vectors (see Value)

Details

This function does not actually perform a cross-validation or partition the data set itself; it simply creates a data structure containing the indices of training and test samples.

Value

If return_factor = FALSE (the default), a represampling object. Specifically, this is a (named) list of length(repetition) resampling objects. Each of these resampling objects is a list of length nfold corresponding to the folds. Each fold is represented by a list containing the components train and test, specifying the indices of training and test samples (row indices for data). If return_factor = TRUE (mainly used internally), a (named) list of length length(repetition). Each component of this list is a vector of length nrow(data) of type factor, specifying for each sample the fold to which it belongs. The factor levels are factor(1:nfold).

See Also

sperrorest, represampling
Examples

```r
data(ecuador)
## non-spatial cross-validation:
resamp <- partition_cv(ecuador, nfold = 5, repetition = 5)
# plot(resamp, ecuador)
# first repetition, second fold, test set indices:
idx <- resamp[["1"]][[2]]$test
# test sample used in this particular repetition and fold:
ecuador[idx, ]
```

---

**partition_cv_strat**  
*Partition the data for a stratified (non-spatial) cross-validation*

### Description

`partition_cv_strat` creates a set of sample indices corresponding to cross-validation test and training sets.

### Usage

```r
partition_cv_strat(
  data, 
  coords = c("x", "y"), 
  nfold = 10, 
  return_factor = FALSE, 
  repetition = 1, 
  seed1 = NULL, 
  strat
)
```

### Arguments

- **data**: data.frame containing at least the columns specified by `coords`
- **coords**: vector of length 2 defining the variables in `data` that contain the x and y coordinates of sample locations
- **nfold**: number of partitions (folds) in `nfold`-fold cross-validation partitioning
- **return_factor**: if FALSE (default), return a `represampling` object; if TRUE (used internally by other `sperrorest` functions), return a list containing factor vectors (see Value)
- **repetition**: numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use `repetition = c(1:100)` to obtain (the 'first') 100 repetitions, and `repetition = c(101:200)` to obtain a different set of 100 repetitions.
- **seed1**: `seed1+i` is the random seed that will be used by `set.seed` in repetition `i` (i in `repetition`) to initialize the random number generator before sampling from the data set.
- **strat**: character: column in `data` containing a factor variable over which the partitioning should be stratified; or factor vector of length `nrow(data)`: variable over which to stratify
Value

A represampling object, see also \texttt{partition_cv()} and \texttt{partition_strat_cv}, however, stratified with respect to the variable \texttt{data[,strat]}; i.e., cross-validation partitioning is done within each set \texttt{data[data[,strat]==i,]} (\texttt{i} in levels(\texttt{data[, strat]})), and the \texttt{i}th folds of all levels are combined into one cross-validation fold.

See Also

\texttt{sperrorest()}, \texttt{as.resampling()}, \texttt{resample_strat_uniform()}

Examples

\begin{verbatim}
  data(ecuador)
  parti <- partition_cv_strat(ecuador, strat = "slides", nfold = 5, repetition = 1)
  idx <- parti["1"][[1]]$train
  mean(ecuador$slides[idx] == "TRUE") / mean(ecuador$slides == "TRUE")
# always == 1
# Non-stratified cross-validation:
  parti <- partition_cv(ecuador, nfold = 5, repetition = 1)
  idx <- parti[["1"]][[1]]$train
  mean(ecuador$slides[idx] == "TRUE") / mean(ecuador$slides == "TRUE")
# close to 1 because of large sample size, but with some random variation
\end{verbatim}

---

\texttt{partition_disc} \hspace{1cm} \textit{Leave-one-disc-out cross-validation and leave-one-out cross-validation}

Description

\texttt{partition_disc} partitions the sample into training and tests set by selecting circular test areas (possibly surrounded by an exclusion buffer) and using the remaining samples as training samples (leave-one-disc-out cross-validation). \texttt{partition_loo} creates training and test sets for leave-one-out cross-validation with (optional) buffer.

Usage

\begin{verbatim}
  partition_disc(
    data, 
    coords = c("x", "y"),
    radius, 
    buffer = 0,
    ndisc = nrow(data),
    seed1 = NULL,
    return_train = TRUE,
    prob = NULL,
  )
\end{verbatim}
replace = FALSE, 
repetition = 1 
)

partition_loo(data, ndisc = nrow(data), replace = FALSE, ...)

Arguments

data data frame containing at least the columns specified by coords
coords vector of length 2 defining the variables in data that contain the x and y coordinates of sample locations.
radius radius of test area discs; performs leave-one-out resampling if radius <0.
buffer radius of additional 'neutral area' around test area discs that is excluded from training and test sets; defaults to 0, i.e. all samples are either in the test area or in the training area.
ndisc Number of discs to be randomly selected; each disc constitutes a separate test set. Defaults to nrow(data), i.e. one disc around each sample.
seed1 seed1+i is the random seed that will be used by set.seed in repetition i (i in repetition) to initialize the random number generator before sampling from the data set.
return_train If FALSE, returns only test sample; if TRUE, also the training area.
prob optional argument to sample.
replace optional argument to sample: sampling with or without replacement?
repetition see partition_cv; however, see Note below: repetition should normally be = 1 in this function.
... arguments to be passed to partition_disc

Value

A repersampling object. Contains length(repetition) resampling objects. Each of these contains ndisc lists with indices of test and (if return_train = TRUE) training sets.

Note

Test area discs are centered at (random) samples, not at general random locations. Test area discs may (and likely will) overlap independently of the value of replace. replace only controls the replacement of the center point of discs when drawing center points from the samples.

radius < 0 does leave-one-out resampling with an optional buffer. radius = 0 is similar except that samples with identical coordinates would fall within the test area disc.

References

**partition_factor**

Partition the data for a (non-spatial) leave-one-factor-out cross-validation based on a given, fixed partitioning

### Description

`partition_factor` creates a represampling object, i.e. a set of sample indices defining cross-validation test and training sets.

### Usage

```r
partition_factor(
  data, 
  coords = c("x", "y"), 
  fac, 
  return_factor = FALSE, 
  repetition = 1
)
```

### Arguments

- **data** `data.frame` containing at least the columns specified by `coords`
- **coords** vector of length 2 defining the variables in `data` that contain the x and y coordinates of sample locations.
- **fac** either the name of a variable (column) in `data`, or a vector of type factor and length `nrow(data)` that contains the partitions to be used for defining training and test samples.
- **return_factor** if `FALSE` (default), return a represampling object; if `TRUE` (used internally by other `sperrorest` functions), return a list containing factor vectors (see Value)

### See Also

`sperrorest`, `partition_cv`, `partition_kmeans`

### Examples

```r
data(ecuador)
parti <- partition_disc(ecuador,
  radius = 200, buffer = 200,
  ndisc = 5, repetition = 1:2
)
# plot(parti, ecuador)
summary(parti)

# leave-one-out with buffer:
parti.loo <- partition_loo(ecuador, buffer = 200)
summary(parti)
```
partition_factor_cv

**partition_factor_cv**

Partition the data for a (non-spatial) k-fold cross-validation at the group level

**Description**

partition_factor_cv creates a represampling object, i.e. a set of sample indices defining cross-validation test and training sets, where partitions are obtained by resampling at the level of groups of observations as defined by a given factor variable. This can be used, for example, to resample agricultural data that is grouped by fields, at the agricultural field level in order to preserve spatial autocorrelation within fields.

**Usage**

```r
partition_factor_cv(
  data,
  coords = c("x", "y"),
  fac,
  nfold = 10,
  repetition = 1,
)```

**repetition** numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use repetition = c(1:100) to obtain (the 'first') 100 repetitions, and repetition = c(101:200) to obtain a different set of 100 repetitions.

**Value**

A represampling object, see also partition_cv for details.

**Note**

In this partitioning approach, all repetitions are identical and therefore pseudo-replications.

**See Also**

sperrorest, partition_cv, as.resampling.factor

**Examples**

```r
data(ecuador)
# I don't recommend using this partitioning for cross-validation,
# this is only for demonstration purposes:
breaks <- quantile(ecuador$dem, seq(0, 1, length = 6))
ecuador$zclass <- cut(ecuador$dem, breaks, include.lowest = TRUE)
summary(ecuador$zclass)
parti <- partition_factor(ecuador, fac = "zclass")
# plot(parti,ecuador)
summary(parti)
```
Arguments

data data.frame containing at least the columns specified by coords
cords vector of length 2 defining the variables in data that contain the x and y coordinates of sample locations.
fac either the name of a variable (column) in data, or a vector of type factor and length nrow(data) that defines groups or clusters of observations.
nfold number of partitions (folds) in nfold-fold cross-validation partitioning
repetition numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use repetition = c(1:100) to obtain (the 'first') 100 repetitions, and repetition = c(101:200) to obtain a different set of 100 repetitions.
seed1 seed1+i is the random seed that will be used by set.seed in repetition i (i in repetition) to initialize the random number generator before sampling from the data set.
return_factor if FALSE (default), return a represampling object; if TRUE (used internally by other sperrorest functions), return a list containing factor vectors (see Value)

Value

A represampling object, see also partition_cv for details.

Note

In this partitioning approach, the number of factor levels in fac must be large enough for this factor-level resampling to make sense.

See Also

sperrorest, partition_cv, partition_factor, as.resampling.factor

partition_kmeans Partition samples spatially using k-means clustering of the coordinates

Description

partition_kmeans divides the study area into irregularly shaped spatial partitions based on k-means (kmeans) clustering of spatial coordinates.
Usage

```r
partition_kmeans(
  data,
  coords = c("x", "y"),
  nfold = 10,
  repetition = 1,
  seed1 = NULL,
  return_factor = FALSE,
  balancing_steps = 1,
  order_clusters = TRUE,
  ...
)
```

Arguments

- **data**: `data.frame` containing at least the columns specified by `coords`.
- **coords**: vector of length 2 defining the variables in `data` that contain the `x` and `y` coordinates of sample locations.
- **nfold**: number of cross-validation folds, i.e. parameter `k` in `k-means` clustering.
- **repetition**: numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use `repetition = c(1:100)` to obtain (the 'first') 100 repetitions, and `repetition = c(101:200)` to obtain a different set of 100 repetitions.
- **seed1**: `seed1+i` is the random seed that will be used by `set.seed` in repetition `i` (i in `repetition`) to initialize the random number generator before sampling from the data set.
- **return_factor**: if FALSE (default), return a `represampling` object; if TRUE (used internally by other `sperrorest` functions), return a list containing factor vectors (see Value).
- **balancing_steps**: if > 1, perform `nfold`-means clustering `balancing_steps` times, and pick the clustering that minimizes the Gini index of the sample size distribution among the partitions. The idea is that 'degenerate' partitions will be avoided, but this also has the side effect of reducing variation among partitioning repetitions. More meaningful constraints (e.g., minimum number of positive and negative samples within each partition should be added in the future.
- **order_clusters**: if TRUE, clusters are ordered by increasing `x` coordinate of center point.
- **...**: additional arguments to `kmeans`.

Value

A `represampling` object, see also `partition_cv` for details.

Note

Default parameter settings may change in future releases.
partition_tiles

References


See Also

sperrorest, partition_cv, partition_disc, partition_tiles, kmeans

Examples

data(ecuador)
resamp <- partition_kmeans(ecuador, nfold = 5, repetition = 2)
# plot(resamp, ecuador)

partition_tiles

Partition the study area into rectangular tiles

Description

partition_tiles divides the study area into a specified number of rectangular tiles. Optionally small partitions can be merged with adjacent tiles to achieve a minimum number or percentage of samples in each tile.

Usage

partition_tiles(
  data,
  coords = c("x", "y"),
  dsplit = NULL,
  nsplit = NULL,
  rotation = c("none", "random", "user"),
  user_rotation,
  offset = c("none", "random", "user"),
  user_offset,
  reassign = TRUE,
  min_frac = 0.025,
  min_n = 5,
  iterate = 1,
  return_factor = FALSE,
  repetition = 1,
  seed1 = NULL
)
Arguments

**data**  
data.frame containing at least the columns specified by **coords**

**coords**  
vector of length 2 defining the variables in **data** that contain the x and y coordinates of sample locations

**dsplit**  
optional vector of length 2: equidistance of splits in (possibly rotated) x direction (**dsplit[1]**) and y direction (**dsplit[2]**) used to define tiles. If **dsplit** is of length 1, its value is recycled. Either **dsplit** or **nsplit** must be specified.

**nsplit**  
optional vector of length 2: number of splits in (possibly rotated) x direction (**nsplit[1]**) and y direction (**nsplit[2]**) used to define tiles. If **nsplit** is of length 1, its value is recycled.

**rotation**  
indicates whether and how the rectangular grid should be rotated; random rotation is only between -45 and +45 degrees.

**user_rotation**  
if **rotation**='user', angles (in degrees) by which the rectangular grid is to be rotated in each repetition. Either a vector of same length as **repetition**, or a single number that will be replicated length(**repetition**) times.

**offset**  
indicates whether and how the rectangular grid should be shifted by an offset.

**user_offset**  
if **offset**='user', a list (or vector) of two components specifying a shift of the rectangular grid in (possibly rotated) x and y direction. The offset values are relative values, a value of 0.5 resulting in a one-half tile shift towards the left, or upward. If this is a list, its first (second) component refers to the rotated x (y) direction, and both components must have same length as **repetition** (or length 1). If a vector of length 2 (or list components have length 1), the two values will be interpreted as relative shifts in (rotated) x and y direction, respectively, and will therefore be recycled as needed (length(**repetition**) times each).

**reassign**  
logical (default TRUE): if TRUE, 'small' tiles (as per **min_frac** and **min_n** arguments and **get_small_tiles**) are merged with (smallest) adjacent tiles. If FALSE, small tiles are 'eliminated', i.e. set to NA.

**min_frac**  
numeric >=0, <1: minimum relative size of partition as percentage of sample; argument passed to **get_small_tiles**. Will be ignored if NULL.

**min_n**  
integer >=0: minimum number of samples per partition; argument passed to **get_small_tiles**. Will be ignored if NULL.

**iterate**  
argument to be passed to **tile_neighbors**

**return_factor**  
if FALSE (default), return a **represampling** object; if TRUE (used internally by other sperrorest functions), return a list containing factor vectors (see Value)

**repetition**  
numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use repetition = c(1:100) to obtain (the 'first') 100 repetitions, and repetition = c(101:200) to obtain a different set of 100 repetitions.

**seed**  
seed1+i is the random seed that will be used by **set.seed** in repetition i (i in repetition) to initialize the random number generator before sampling from the data set.
plot.represampling

Description

plot.represampling displays the partitions or samples corresponding arising from the resampling of a data set.

Value

A represampling object. Contains length(repetition) resampling objects as repetitions. The exact number of folds / test-set tiles within each resampling objects depends on the spatial configuration of the data set and possible cleaning steps (see min_frac, min_n).

Note

Default parameter settings may change in future releases. This function, especially the rotation and shifting part of it and the algorithm for cleaning up small tiles is still a bit experimental. Use with caution. For non-zero offsets (offset!=’none’), the number of tiles may actually be greater than nsplit[1]*nsplit[2] because of fractional tiles lurking into the study region. reassign=TRUE with suitable thresholds is therefore recommended for non-zero (including random) offsets.

See Also

sperrorest, as.resampling.factor, get_small_tiles, tile_neighbors

Examples

data(ecuador)
set.seed(42)
parti <- partition_tiles(ecuador, nsplit = c(4, 3), reassign = FALSE)
  # plot(parti,ecuador)
  # tile A4 has only 55 samples
  # same partitioning, but now merge tiles with less than 100 samples to
  # adjacent tiles:
parti2 <- partition_tiles(ecuador,
  nsplit = c(4, 3), reassign = TRUE,
  min_n = 100
)
  # plot(parti2,ecuador)
summary(parti2)
  # tile B4 (in 'parti') was smaller than A3, therefore A4 was merged with B4,
  # not with A3
  # now with random rotation and offset, and tiles of 2000 m length:
parti3 <- partition_tiles(ecuador,
  dsplit = 2000, offset = "random",
  rotation = "random", reassign = TRUE, min_n = 100
)
  # plot(parti3, ecuador)
summary(parti3)
## S3 method for class 'represampling'
plot(x, data, coords = c("x", "y"), pch = "+", wiggle_sd = 0, ...)

## S3 method for class 'resampling'
plot(x, ...)

### Arguments

- **x**: a represampling resp. resampling object.
- **data**: a data.frame of samples containing at least the x and y coordinates of samples as specified by coords.
- **coords**: vector of length 2 defining the variables in data that contain the x and y coordinates of sample locations.
- **pch**: point symbol (to be passed to points).
- **wiggle_sd**: ‘wiggle’ the point locations in x and y direction to avoid overplotting of samples drawn multiple times by bootstrap methods; this is a standard deviation (in the units of the x/y coordinates) of a normal distribution and defaults to 0 (no wiggling).
- **...**: additional arguments to plot.

### Note

This function is not intended for samples obtained by resampling with replacement (e.g., bootstrap) because training and test points will be overplotted in that case. The size of the plotting region will also limit the number of maps that can be displayed at once, i.e., the number of rows (repetitions) and fields (columns).

### Examples

```r
data(ecuador)
# non-spatial cross-validation:
resamp <- partition_cv(ecuador, nfold = 5, repetition = 1:2)
# plot(resamp, ecuador)
# spatial cross-validation using k-means clustering:
resamp <- partition_kmeans(ecuador, nfold = 5, repetition = 1:2)
# plot(resamp, ecuador)
```

---

**represampling_bootstrap**

*Non-spatial bootstrap resampling*

**Description**

represampling_bootstrap draws a bootstrap random sample (with replacement) from data.
Usage

represampling_bootstrap(
  data,
  coords = c("x", "y"),
  nboot = nrow(data),
  repetition = 1,
  seed1 = NULL,
  oob = FALSE
)

Arguments

data data.frame containing at least the columns specified by coords
coords vector of length 2 defining the variables in data that contain the x and y coordinates of sample locations.
nboot Size of bootstrap sample
repetition numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use repetition = c(1:100) to obtain (the 'first') 100 repetitions, and repetition = c(101:200) to obtain a different set of 100 repetitions.
seed1 seed1+i is the random seed that will be used by set.seed in repetition i (i in repetition) to initialize the random number generator before sampling from the data set.
oob logical (default FALSE): if TRUE, use the out-of-bag sample as the test sample; if FALSE, draw a second bootstrap sample of size nboot independently to obtain a test sample.

Value

A represampling object. This is a (named) list containing length(repetition). resampling objects. Each of these contains only one list with indices of training and test samples. Indices are row indices for data.

Examples

data(ecuador)
# only 10 bootstrap repetitions, normally use >=100:
parti <- represampling_bootstrap(ecuador, repetition = 10)
# plot(parti, ecuador) # careful: overplotting occurs
# because some samples are included in both the training and
# the test sample (possibly even multiple times)
represampling_disc_bootstrap

*Overlapping spatial block bootstrap using circular blocks*

**Description**

`represampling_disc_bootstrap` performs a spatial block bootstrap by resampling at the level of rectangular partitions or 'tiles' generated by `partition_tiles`.

**Usage**

```r
represampling_disc_bootstrap(
  data, coords = c("x", "y"), nboot, repetition = 1, seed1 = NULL, oob = FALSE, ...
)
```

**Arguments**

- **data** `data.frame` containing at least the columns specified by `coords`
- **coords** vector of length 2 defining the variables in `data` that contain the x and y coordinates of sample locations.
- **nboot** number of bootstrap samples; you may specify different values for the training sample (`nboot[1]`) and for the test sample (`nboot[2]`).
- **repetition** numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use `repetition = c(1:100)` to obtain (the 'first') 100 repetitions, and `repetition = c(101:200)` to obtain a different set of 100 repetitions.
- **seed1** `seed1+i` is the random seed that will be used by `set.seed` in repetition `i` (`i` in `repetition`) to initialize the random number generator before sampling from the data set.
- **oob** logical (default `FALSE`): if `TRUE`, use the out-of-bag sample as the test sample (the complement of the `nboot[1]` test set discs, minus the buffer area as specified in the ... arguments to `partition_disc`); if `FALSE`, draw a second bootstrap sample of size `nboot` independently to obtain a test sample (sets of overlapping discs drawn with replacement).
- **...** additional arguments to be passed to `partition_disc`; note that a `buffer` argument has not effect if `oob=FALSE`; see example below

**Note**

Performs `nboot` out of `nrow(data)` resampling of circular discs. This is an *overlapping* spatial block bootstrap where the blocks are circular.
Examples

```r
data(ecuador)
# Overlapping disc bootstrap:
parti <- represampling_disc_bootstrap(ecuador, 
  radius = 200, nboot = 20, 
  oob = FALSE 
)  
# plot(parti, ecuador)
# Note that a 'buffer' argument would make no difference because boostrap
# sets of discs are drawn independently for the training and test sample.
#
# Overlapping disc bootstrap for training sample, out-of-bag sample as test
# sample:
parti <- represampling_disc_bootstrap(ecuador, 
  radius = 200, buffer = 200, 
  nboot = 10, oob = TRUE 
)  
# plot(parti,ecuador)
```

---

**represampling_factor_bootstrap**

*Bootstrap at an aggregated level*

**Description**

represampling_factor_bootstrap resamples partitions defined by a factor variable. This can be used for non-overlapping block bootstraps and similar.

**Usage**

```r
represampling_factor_bootstrap(
  data, 
  fac, 
  repetition = 1, 
  nboot = -1, 
  seed1 = NULL, 
  oob = FALSE 
)
```

**Arguments**

data frame containing at least the columns specified by coords

fac defines a grouping or partitioning of the samples in data; three possible types: (1) the name of a variable in data (coerced to factor if not already a factor variable); (2) a factor variable (or a vector that can be coerced to factor); (3) a list of factor variables (or vectors that can be coerced to factor); this list must be of length length(repetition), and if it is named, the names must be equal to as.character(repetition); this list will typically be generated by a partition.* function with return_factor = TRUE (see Examples below)
represampling_factor_bootstrap

repetition numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use repetition = c(1:100) to obtain (the 'first') 100 repetitions, and repetition = c(101:200) to obtain a different set of 100 repetitions.

nboot number of bootstrap replications used for generating the bootstrap training sample (nboot[1]) and the test sample (nboot[2]); nboot[2] is ignored (with a warning) if oob = TRUE. A value of -1 will be substituted with the number of levels of the factor variable, corresponding to an \( n \) out of \( n \) bootstrap at the grouping level defined by fac.

seed1 seed1+i is the random seed that will be used by set.seed in repetition i (i in repetition) to initialize the random number generator before sampling from the data set.

oob if TRUE, the test sample will be the out-of-bag sample; if FALSE (default), the test sample is an independently drawn bootstrap sample of size nboot[2].

Details

nboot refers to the number of groups (as defined by the factors) to be drawn with replacement from the set of groups. I.e., if fac is a factor variable, nboot would normally not be greater than nlevels(fac), nlevels(fac) being the default as per nboot = -1.

See Also

represampling_disc_bootstrap, represampling_tile_bootstrap

Examples

data(ecuador)
# a dummy example for demonstration, performing bootstrap
# at the level of an arbitrary factor variable:
parti <- represampling_factor_bootstrap(ecuador,
  factor(floor(ecuador$dem / 100)),
  oob = TRUE
)
# plot(parti,ecuador)

# using the factor bootstrap for a non-overlapping block bootstrap
# (see also represampling_tile_bootstrap):
fac <- partition_tiles(ecuador,
  return_factor = TRUE, repetition = c(1:3),
  dsplit = 500, min_n = 200, rotation = "random",
  offset = "random"
)
parti <- represampling_factor_bootstrap(ecuador, fac,
  oob = TRUE,
  repetition = c(1:3)
)
# plot(parti, ecuador)
represampling_tile_bootstrap

*Spatial block bootstrap using rectangular blocks*

**Description**

represampling_tile_bootstrap performs a non-overlapping spatial block bootstrap by resampling at the level of rectangular partitions or 'tiles' generated by partition_tiles.

**Usage**

```r
represampling_tile_bootstrap(
  data,
  coords = c("x", "y"),
  repetition = 1,
  nboot = -1,
  seed1 = NULL,
  oob = FALSE,
  ...
)
```

**Arguments**

- `data` *data.frame* containing at least the columns specified by `coords`
- `coords` vector of length 2 defining the variables in `data` that contain the x and y coordinates of sample locations.
- `repetition` numeric vector: cross-validation repetitions to be generated. Note that this is not the number of repetitions, but the indices of these repetitions. E.g., use `repetition = c(1:100)` to obtain (the 'first') 100 repetitions, and `repetition = c(101:200)` to obtain a different set of 100 repetitions.
- `nboot` see `represampling_factor_bootstrap`
- `seed1` `seed1+i` is the random seed that will be used by `set.seed` in repetition `i` (i in repetition) to initialize the random number generator before sampling from the data set.
- `oob` see `represampling_factor_bootstrap`
- `...` additional arguments to be passed to `partition_tiles`
resample_factor

**Draw uniform random (sub)sample at the group level**

**Description**

resample_factor draws a random (sub)sample (with or without replacement) of the groups or clusters identified by the fac argument.

**Usage**

```r
resample_factor(data, param = list(fac = "class", n = Inf, replace = FALSE))
```

**Arguments**

- `data` a data.frame, rows represent samples
- `param` a list with the following components: fac is a factor variable of length nrow(data) or the name of a factor variable in data; n is a numeric value specifying the size of the subsample (in terms of groups, not observations); replace determines if resampling of groups is to be done with or without replacement.

**Details**

If param$replace=FALSE, a subsample of min(param$n,nlevel(data[,fac])) groups will be drawn from data. If param$replace=TRUE, the number of groups to be drawn is param$n.

**Value**

a data.frame containing a subset of the rows of data.

**See Also**

resample_strat_uniform(), sample()

---

resample_strat_uniform

**Draw stratified random sample**

**Description**

resample_strat_uniform draws a stratified random sample (with or without replacement) from the samples in data. Stratification is over the levels of data[, param$response]. The same number of samples is drawn within each level.
resample_uniform

Usage

resample_strat_uniform(
  data,
  param = list(strat = "class", nstrat = Inf, replace = FALSE)
)

Arguments

data       a data.frame, rows represent samples
param      a list with the following components: strat is either the name of a factor variable in data that defines the stratification levels, or a vector of type factor and length nrow(data); n is a numeric value specifying the size of the subsample; replace determines if sampling is with or without replacement

Details

If param$replace=FALSE, a subsample of size min(param$n,nrow(data)) will be drawn from data. If param$replace=TRUE, the size of the subsample is param$n.

Value

a data.frame containing a subset of the rows of data.

See Also

resample_uniform(), sample()

Examples

data(ecuador) # Muenchow et al. (2012), see ?ecuador
d <- resample_strat_uniform(ecuador,
  param = list(strat = "slides", nstrat = 100)
)
nrow(d) # == 200
sum(d$slides == "TRUE") # == 100
Arguments

data a data.frame, rows represent samples
param a list with the following components: n is a numeric value specifying the size of
the subsample; replace determines if sampling is with or without replacement

Details

If param$replace=FALSE, a subsample of size min(param$n,nrow(data)) will be drawn from
data. If param$replace=TRUE, the size of the subsample is param$n.

Value

a data.frame containing a subset of the rows of data.

See Also

resample_strat_uniform(), sample()

Examples

# Muenchow et al. (2012), see ?ecuador
d <- resample_uniform(ecuador, param = list(strat = "slides", n = 200))
# == 200
sum(d$slides == "TRUE")
train_param = NULL,
test_fun = NULL,
err_fun = err_default,
imp_variables = NULL,
imp_permutations = 1000,
imp_sample_from = c("test", "train", "all"),
importance = !is.null(imp_variables),
distance = FALSE,
do_gc = 1,
progress = "all",
benchmark = FALSE,
mode_rep = c("future", "sequential", "loop"),
mode_fold = c("sequential", "future", "loop"),
verbose = 0
)

Arguments

formula A formula specifying the variables used by the model. Only simple formulas without interactions or nonlinear terms should be used, e.g. \( y \sim x_1 + x_2 + x_3 \) but not \( y \sim x_1 \times x_2 + \log(x_3) \). Formulas involving interaction and nonlinear terms may possibly work for error estimation but not for variable importance assessment, but should be used with caution. The formula \( y \sim \ldots \) is not supported, but \( y \sim 1 \) (i.e. no predictors) is.

data a data.frame with predictor and response variables. Training and test samples will be drawn from this data set by train_fun and test_fun, respectively.

cords vector of length 2 defining the variables in data that contain the x and y coordinates of sample locations.

model_fun Function that fits a predictive model, such as glm or rpart. The function must accept at least two arguments, the first one being a formula and the second a data.frame with the learning sample.

model_args Arguments to be passed to model_fun (in addition to the formula and data argument, which are provided by sperrorest).

pred_fun Prediction function for a fitted model object created by model. Must accept at least two arguments: the fitted object and a data.frame newdata with data on which to predict the outcome.

pred_args (optional) Arguments to pred_fun (in addition to the fitted model object and the newdata argument, which are provided by sperrorest).

smp_fun A function for sampling training and test sets from data. E.g. partition_kmeans for spatial cross-validation using spatial k-means clustering.

smp_args (optional) Arguments to be passed to smp_fun.

train_fun (optional) A function for resampling or subsampling the training sample in order to achieve, e.g., uniform sample sizes on all training sets, or maintaining a certain ratio of positives and negatives in training sets. E.g. resample_uniform or resample_strat_uniform.
train_param  (optional) Arguments to be passed to resample_fun.

test_fun   (optional) Like train_fun but for the test set.

test_param (optional) Arguments to be passed to test_fun.

err_fun    A function that calculates selected error measures from the known responses in
data and the model predictions delivered by pred_fun. E.g. err_default (the
default).

imp_variables (optional; used if importance = TRUE). Variables for which permutation-based
variable importance assessment is performed. If importance = TRUE and imp_variables
== NULL, all variables in formula will be used.

imp_permutations (optional; used if importance = TRUE). Number of permutations used for vari-
able importance assessment.

imp_sample_from (default: "test"): specified if the permuted feature values should be taken
from the test set, the training set (a rather unlikely choice), or the entire sam-
ple ("all"). The latter is useful in leave-one-out resampling situations where
the test set is simply too small to perform any kind of resampling. In any case
importances are always estimates on the test set. (Note that resampling with
replacement is used if the test set is larger than the set from which the permuted
values are to be taken.)

importance logical (default: FALSE): perform permutation-based variable importance assess-
ment?

distance logical (default: FALSE): if TRUE, calculate mean nearest-neighbour distances
from test samples to training samples using add.distance.represampling.

do_gc numeric (default: 1): defines frequency of memory garbage collection by calling
gc; if < 1, no garbage collection; if >= 1, run a gc after each repetition; if >= 2,
after each fold.

progress character (default: all): Whether to show progress information (if possible).
Default shows repetition, fold and (if enabled) variable importance progress. Set
to "rep" for repetition information only or FALSE for no progress information.

benchmark (optional) logical (default: FALSE): if TRUE, perform benchmarking and return
sperrorestbenchmark object.

mode_rep, mode_fold character (default: "future" and "sequential", respectively): specifies whether
to parallelize the execution at the repetition level, at the fold level, or not at all.
Parallel execution uses future.apply::future_lapply() (see details below).
It is only possible to parallelize at the repetition level or at the fold level. The
"loop" option uses a for loop instead of an lapply function; this option is for
debugging purposes.

verbose Controls the amount of information printed while processing. Defaults to 0 (no
output).

Details
Custom predict functions passed to pred_fun, which consist of multiple child functions, must be
defined in one function.
Value

A list (object of class sperrorest) with (up to) six components:

- error_rep: sperrorestrepeerror containing predictive performances at the repetition level
- error_fold: sperroresterror object containing predictive performances at the fold level
- represampling: represampling object
- importance: sperrorestimportance object containing permutation-based variable importances at the fold level
- benchmark: sperrorestbenchmark object containing information on the system the code is running on, starting and finishing times, number of available CPU cores and runtime performance
- package_version: sperrorestpackageversion object containing information about the sperrorest package version

Parallelization

Running in parallel is supported via package future. Have a look at vignette("future-1-overview", package = "future"). In short: Choose a backend and specify the number of workers, then call sperrorest() as usual. Example:

```r
future::plan(future.callr::callr, workers = 2)
sperrorest()
```

Parallelization at the repetition is recommended when using repeated cross-validation. If the 'granularity' of parallelized function calls is too fine, the overall runtime will be very poor since the overhead for passing arguments and handling environments becomes too large. Use fold-level parallelization only when the processing time of individual folds is very large and the number of repetitions is small or equals 1.

Note that nested calls to future are not possible. Therefore a sequential sperrorest call should be used for hyperparameter tuning in a nested cross-validation.

References


Examples

```r
## Classification tree example using non-spatial partitioning

# Muenchow et al. (2012), see ?ecuador
fo <- slides ~ dem + slope + hcurv + vcurv + log.carea + cslope

library(rpart)
mypred_part <- function(object, newdata) predict(object, newdata)[, 2]
ctrl <- rpart.control(cp = 0.005) # show the effects of overfitting
fit <- rpart(fo, data = ecuador, control = ctrl)

### Non-spatial cross-validation:
mypred_part <- function(object, newdata) predict(object, newdata)[, 2]
nsp_res <- sperrorest(  
data = ecuador, formula = fo,  
model_fun = rpart,  
model_args = list(control = ctrl),  
pred_fun = mypred_part,  
progress = TRUE,  
smp_fun = partition_cv,  
smp_args = list(repetition = 1:2, nfold = 3)
)
summary(nsp_res$error_rep)
summary(nsp_res$error_fold)
summary(nsp_res$represampling)
# plot(nsp_res$represampling, ecuador)

### Spatial cross-validation:
sp_res <- sperrorest(  
data = ecuador, formula = fo,  
model_fun = rpart,  
model_args = list(control = ctrl),  
pred_fun = mypred_part,  
progress = TRUE,  
smp_fun = partition_kmeans,  
smp_args = list(repetition = 1:2, nfold = 3)
)
summary(sp_res$error_rep)
summary(sp_res$error_fold)
summary(sp_res$represampling)
# plot(sp_res$represampling, ecuador)
```
summary.represampling

Description

Calculates sample sizes of training and test sets within repetitions and folds of a resampling or represampling object.

Usage

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

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

Arguments

object A resampling or represampling object.

... currently ignored.

Value

A list of data.frames summarizing the sample sizes of training and test sets in each fold of each repetition.
Summary of Error Statistics Obtained by sperrorest

Description

summary.sperroresterror calculates mean, standard deviation, median etc. of the calculated error measures at the specified level (overall, repetition, or fold). summary.sperrorestrepperror does the same with the pooled error, at the overall or repetition level.

Usage

```r
## S3 method for class 'sperroresterror'
summary(object, level = 0, pooled = TRUE, na.rm = TRUE, ...)
```

Arguments

- `object`: sperroresterror resp. sperrorestcombinederror error object calculated by sperrorest
- `level`: Level at which errors are summarized: 0: overall (i.e. across all repetitions); 1: repetition; 2: fold
- `pooled`: If TRUE (default), mean and standard deviation etc are calculated between fold-level error estimates. If FALSE, apply first a weighted.mean among folds before calculating mean, standard deviation etc among repetitions. See also Details.
- `na.rm`: Remove NA values? See mean etc.
- `...`: additional arguments (currently ignored)

Details

Let’s use an example to explain the `error_rep` argument. E.g., assume we are using 100-repeated 10-fold cross-validation. If `error_rep = TRUE` (default), the mean and standard deviation calculated when summarizing at `level = 0` are calculated across the error estimates obtained for each of the 100*10 = 1000 folds. If `error_rep = FALSE`, mean and standard deviation are calculated across the 100 repetitions, using the weighted average of the fold-level errors to calculate an error value for the entire sample. This will essentially not affect the mean value but of course the standard deviation of the error.

`error_rep = FALSE` is not recommended, it is mainly for testing purposes; when the test sets are small (as in leave-one-out cross-validation, in the extreme case), consider running sperrorest with `error_rep = TRUE` and examine only the `error_rep` component of its result.

Value

Depending on the level of aggregation, a list or data.frame with mean, and at level 0 also standard deviation, median and IQR of the error measures.
See Also

sperrorest

summary.sperrorestimportance

Summary and print methods for sperrorest results

Description

Summary methods provide varying level of detail while print methods provide full details.
tile_neighbors

Determine the names of neighbouring tiles in a rectangular pattern

Description

This based on 'counting' up and down based on the tile name.

Usage

```r
## S3 method for class 'sperrorestrepeerror'
summary(object, level = 0, na.rm = TRUE, ...)

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

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

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

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

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

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

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

Arguments

- `object` a `sperrorest` object
- `level` Level at which errors are summarized: 0: overall (i.e. across all repetitions); 1: repetition; 2: fold
- `na.rm` Remove NA values? See `mean` etc.
- `...` additional arguments for `summary.sperroresterror` or `summary.sperrorestimportance`
- `x` Depending on method, a `sperrorest`, `sperroresterror` or `sperrorestimportance` object

See Also

- `sperrorest`, `summary.sperroresterror`, `summary.sperrorestimportance`
tile_neighbors

Usage

    tile_neighbors(nm, tileset, iterate = 0, diagonal = FALSE)

Arguments

  nm  Character string or factor: name of a tile, e.g., 'X4:Y6'
  tileset  Admissible tile names; if missing and nm is a factor variable, then levels(nm) is used as a default for tileset.
  iterate  internal - do not change default: to control behaviour in an interactive call to this function.
  diagonal  if TRUE, diagonal neighbours are also considered neighbours.

Value

  Character string.
Index

add.distance, 3, 9
add.distance.represampling, 33
add.distance.resampling, 3
as.character.tilename (as.tilename), 7
as.numeric.tilename (as.tilename), 7
as.represampling, 4
as.represampling_list
  (as.represampling), 4
as.resampling, 5
as.resampling(), 14
as.resampling.factor, 17, 18, 22
as.resampling.default (as.resampling), 5
as.resampling_list (as.resampling), 5
as.tilename, 7
as.tilename_character (as.tilename), 7
as.tilename_numeric (as.tilename), 7
dataset_distance, 3, 4, 8
derr_default, 9, 33
gc, 33
get_small_tiles, 10, 21, 22
IQR, 10
is.resampling (as.resampling), 5
is_represampling (as.represampling), 4
kmeans, 18–20
mad, 10
mean, 37–39
partition_cv, 5, 7, 11, 16–20
partition_cv(), 14
partition_disc_strat, 13
partition_disc, 14, 20, 25
partition_factor, 16, 18
partition_factor_cv, 17
partition_kmeans, 5, 7, 16, 18, 32
partition_loo (partition_disc), 14
partition_tiles, 7, 8, 11, 20, 20, 28
plot, 23
plot.represampling, 22
plot.resampling (plot.represampling), 22
points, 23
print.represampling (as.represampling), 4
print.resampling (as.resampling), 5
print.sperrorest
  (summary.sperrorestrepeerror), 38
print.sperrorestbenchmarks
  (summary.sperrorestrepeerror), 38
print.sperroresterror
  (summary.sperrorestrepeerror), 38
print.sperrorestimportance
  (summary.sperrorestrepeerror), 38
print.sperrorestpackageversion
  (summary.sperrorestrepeerror), 38
print.sperrorestrepeerror
  (summary.sperrorestrepeerror), 38
print.tilename (as.tilename), 7
rep, 6
represampling, 3, 4, 7, 8, 11–19, 21–24, 34
represampling (as.represampling), 4
represampling_bootstrap, 6, 7, 23
represampling_disc_bootstrap, 5, 25, 27
represampling_factor_bootstrap, 26, 28
represampling_tile_bootstrap, 7, 8, 27, 28
resample_factor, 29
resample_strat_uniform, 29, 32
resample_strat_uniform(), 14, 29, 31
resample_uniform, 30, 32
resample_uniform(). 30
resampling, 3–5, 12, 22–24
resampling (as.resampling), 5

sample, 15
sample(), 29–31
set.seed, 12, 13, 15, 18, 19, 21, 24, 25, 27, 28
sperrorest, 12, 16–18, 20, 22, 31, 37–39
sperrorest(), 14
sperrorest-package, 2
summary.represampling, 36
summary.resampling
  (summary.represampling), 36
summary.sperrorest
  (summary.sperrorestreparerror), 38
summary.sperroresterror, 37, 39
summary.sperrorestimportance, 38, 39
summary.sperrorestreparerror, 38

tile_neighbors, 21, 22, 39
tilename, 11
tilename (as.tilename), 7

validate.resampling (as.resampling), 5

weighted.mean, 37