Package ‘tidysdm’

June 23, 2024

Title Species Distribution Models with Tidymodels
Version 0.9.5
Description Fit species distribution models (SDMs) using the 'tidymodels' framework, which provides a standardised interface to define models and process their outputs. 'tidysdm' expands 'tidymodels' by providing methods for spatial objects, models and metrics specific to SDMs, as well as a number of specialised functions to process occurrences for contemporary and palaeo datasets. The full functionalities of the package are described in Leonardi et al. (2023) <doi:10.1101/2023.07.24.550358>.
License AGPL (>= 3)
Encoding UTF-8
Language en-GB
URL https://github.com/EvolEcolGroup/tidysdm,
https://evolecolgroup.github.io/tidysdm/
BugReports https://github.com/EvolEcolGroup/tidysdm/issues
RoxygenNote 7.3.1
Depends tidymodels, spatialsample, R (>= 3.50)
Imports dials, DALEX, DALEXtra, dplyr, ggplot2, lubridate, magrittr, maxnet, methods, parsnip, patchwork, recipes, rsample, rlang (>= 1.0.0), stats, sf, terra, tibble, tune, workflows, workflowsets, yardstick
Suggests blockCV, data.table, doParallel, earth, kernlab, knitr, overlapping, pastclim (>= 2.0.0), ranger, rgbif, rmarkdown, spelling, stacks, testthat (>= 3.0.0), tidyterra, vdiffr, xgboost
VignetteBuilder knitr
Config/testthat/edition 3
LazyData true
NeedsCompilation no
Description

This function adds member(s) to a `simple_ensemble()` object, taking the best member from each workflow provided. It is possible to pass individual `tune_results` objects from a tuned workflow, or a `workflowsets::workflow_set()`.

Usage

```r
add_member(x, member, ...)
```

## Default S3 method:
```r
default_method <- add_member(x, member, ...)
```

## S3 method for class 'tune_results'
```r
default_method <- add_member(x, member, metric = NULL, id = NULL, ...)
```

## S3 method for class 'workflow_set'
```r
default_method <- add_member(x, member, metric = NULL, ...)
```
Arguments

x       a simple_ensemble to which member(s) will be added
member   a tune_results, or a workflowsets::workflow_set
...      not used at the moment.
metric   A character string (or NULL) for which metric to optimize. If NULL, the first metric is used.
id       the name to be given to this workflow in the wflow_id column.

Value

a simple_ensemble with additional member(s)

add_repeat  Add repeat(s) to a repeated ensemble

Description

This function adds repeat(s) to a repeat_ensemble object, where each repeat is a simple_ensemble. All repeats must contain the same members, selected using the same metric.

Usage

add_repeat(x, rep, ...)

## Default S3 method:
add_repeat(x, rep, ...)

## S3 method for class 'simple_ensemble'
add_repeat(x, rep, ...)

## S3 method for class 'list'
add_repeat(x, rep, ...)

Arguments

x       a repeat_ensemble to which repeat(s) will be added
rep      a repeat, as a single simple_ensemble, or a list of simple_ensemble objects
...      not used at the moment.

Value

a repeat_ensemble with additional repeat(s)
**autplot.simple_ensemble**

Plot the results of a simple ensemble

---

**Description**

This autplot() method plots performance metrics that have been ranked using a metric.

**Usage**

```r
## S3 method for class 'simple_ensemble'
autplot(
  object,
  rank_metric = NULL,
  metric = NULL,
  std_errs = stats::qnorm(0.95),
  ...
)
```

**Arguments**

- **object**: A `simple_ensemble` whose elements have results.
- **rank_metric**: A character string for which metric should be used to rank the results. If none is given, the first metric in the metric set is used (after filtering by the `metric` option).
- **metric**: A character vector for which metrics (apart from `rank_metric`) to be included in the visualization. If NULL (the default), all available metrics will be plotted.
- **std_errs**: The number of standard errors to plot (if the standard error exists).
- **...**: Other options to pass to autplot(). Currently unused.

**Details**

This function is intended to produce a default plot to visualize helpful information across all possible applications of a `simple_ensemble`. More sophisticated plots can be produced using standard `ggplot2` code for plotting.

The x-axis is the workflow rank in the set (a value of one being the best) versus the performance metric(s) on the y-axis. With multiple metrics, there will be facets for each metric, with the `rank_metric` first (if any was provided; otherwise the metric used to create the `simple_ensemble` will be used).

If multiple resamples are used, confidence bounds are shown for each result (95% confidence, by default).

**Value**

A ggplot object.
Examples

```r
# we use the two_class_example from `workflowsets`
two_class_ens <- simple_ensemble() %>%
  add_member(two_class_res, metric = "roc_auc")
autoplot(two_class_ens)
```

## autoplot spatial initial split

Create a ggplot for a spatial initial rsplit.

Description

This method provides a good visualization method for a spatial initial rsplit.

Usage

```r
## S3 method for class 'spatial_initial_split'
autoplot(object, ..., alpha = 0.6)
```

Arguments

- **object**
  A spatial_initial_rsplit object. Note that only resamples made from sf objects create spatial_initial_rsplit objects; this function will not work for resamples made with non-spatial tibbles or data.frames.
- **...**
  Options passed to ggplot2::geom_sf().
- **alpha**
  Opacity, passed to ggplot2::geom_sf(). Values of alpha range from 0 to 1, with lower values corresponding to more transparent colors.

Details

This plot method is a wrapper around the standard spatial_rsplit method, but it re-labels the folds as Testing and Training following the convention for a standard initial_split object.

Value

A ggplot object with each fold assigned a color, made using ggplot2::geom_sf().

Examples

```r
set.seed(123)
block_initial <- spatial_initial_split(boston_canopy, prop = 1 / 5, spatial_block_cv)
autoplot(block_initial)
```
Convert an object created with blockCV to an rsample object

**Description**

This function creates objects created with blockCV to rsample objects that can be used by tidysdm. BlockCV provides more sophisticated sampling options than the spatialsample library. For example, it is possible to stratify the sampling to ensure that presences and absences are evenly distributed among the folds (see the example below).

**Usage**

```r
blockcv2rsample(x, data)
```

**Arguments**

- `x`: a object created with a blockCV function
- `data`: the sf object used to create x

**Details**

Note that currently only objects of type `cv_spatial` and `cv_cluster` are supported.

**Value**

an rsample object

**Examples**

```r
library(blockCV)
points <- read.csv(system.file("extdata/", "species.csv", package = "blockCV"))
pa_data <- sf::st_as_sf(points, coords = c("x", "y"), crs = 7845)
sb1 <- cv_spatial(
  x = pa_data,
  column = "occ", # the response column to balance the folds
  k = 5, # number of folds
  size = 350000, # size of the blocks in metres
  selection = "random", # random blocks-to-fold
  iteration = 10
) # find evenly dispersed folds
sb1_rsample <- blockcv2rsample(sb1, pa_data)
class(sb1_rsample)
autoplot(sb1_rsample)
```
**Boyce Continuous Index (BCI)**

**Description**

This function the Boyce Continuous Index, a measure of model accuracy appropriate for Species Distribution Models with presence only data (i.e. using pseudoabsences or background). The algorithm used here comes from the package `enmSdm`, and uses multiple overlapping windows.

**Usage**

```r
boyce_cont(data, ...)

## S3 method for class 'data.frame'
boyce_cont(
  data,
  truth,
  ...,
  estimator = NULL,
  na.rm = TRUE,
  event_level = "first",
  case_weights = NULL
)

## S3 method for class 'sf'
boyce_cont(data, ...)

boyce_cont_vec(
  truth,
  estimate,
  estimator = NULL,
  na.rm = TRUE,
  event_level = "first",
  case_weights = NULL,
  ...
)
```

**Arguments**

- `data` Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
- `...` A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
truth

The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimator

One of "binary", "hand_till", "macro", or "macro_weighted" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The others are general methods for calculating multiclass metrics. The default will automatically choose "binary" if truth is binary, "hand_till" if truth has >2 levels and case_weights isn’t specified, or "macro" if truth has >2 levels and case_weights is specified (in which case "hand_till" isn’t well-defined).

na_rm

A logical value indicating whether NA values should be stripped before the computation proceeds.

event_level

A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first"

case_weights

The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

estimate

If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.

Details

There is no multiclass version of this function, it only operates on binary predictions (e.g. presences and absences in SDMs).

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups.

References


See Also

Other class probability metrics: kap_max(), tss_max()

Examples

boyce_cont(two_class_example, truth, Class1)
**calib_class_thresh**  
*Calibrate class thresholds*

**Description**

Predict for a new dataset by using a simple ensemble. Predictions from individual models are combined according to `fun`.

**Usage**

```r
calib_class_thresh(object, class_thresh, metric_thresh = NULL)
```

**Arguments**

- `object`  
an `simple_ensemble` object

- `class_thresh`  
probability threshold used to convert probabilities into classes. It can be a number (between 0 and 1), or a character metric (currently "tss_max", "kap_max" or "sensitivity"). For sensitivity, an additional target value is passed along as a second element of a vector, e.g. `c("sensitivity",0.8)`.

- `metric_thresh`  
a vector of length 2 giving a metric and its threshold, which will be used to prune which models in the ensemble will be used for the prediction. The 'metrics' need to have been computed when the workflow was tuned. The metric’s threshold needs to match the value used during prediction. Examples are `c("accuracy",0.8)` or `c("boyce_cont",0.7)`.

**Value**

- a `simple_ensemble` object

**Examples**

```r
test_ens <- simple_ensemble() %>%
  add_member(two_class_res[1:3, ], metric = "roc_auc")
test_ens <- calib_class_thresh(test_ens, class_thresh = "tss_max")
test_ens <- calib_class_thresh(test_ens, class_thresh = "kap_max")
test_ens <- calib_class_thresh(test_ens, class_thresh = c("sens", 0.9))
```

**check_sdm_presence**  
*Check that the column with presences is correctly formatted*

**Description**

In `tidysdm`, the string defining presences should be the first level of the response factor. This function checks that the column is correctly formatted.
Usage

check_sdm_presence(.data, .col, presence_level = "presence")

Arguments

.data a data.frame or tibble, or a derived object such as an sf data.frame
.col the column containing the presences
.presence_level the string used to define the presence level of .col

Value

TRUE if correctly formatted

Description

Check the balance of presences vs pseudoabsences among splits

Usage

check_splits_balance(splits, .col)

Arguments

splits the data splits (an rset or split object), generated by a function such as spatialsample::spatial_block_cv()
.col the column containing the presences

Value

a tibble of the number of presences and pseudoabsences in the assessment and analysis set of each split (or training and testing in an initial split)

Examples

lacerta_thin <- readRDS(system.file("extdata/lacerta_climate_sf.RDS", package = "tidysdm")
lacerta_cv <- spatial_block_cv(lacerta_thin, v = 5)
check_splits_balance(lacerta_cv, class)
clamp_predictors  
Requires the predictors to match values in training set

Description

This function clamps the environmental variables in a `terra::SpatRaster` or `terra::SpatRasterDataset` so that their minimum and maximum values do not exceed the range in the training dataset.

Usage

```r
clamp_predictors(x, training, .col, use_na)

# Default S3 method:
clamp_predictors(x, training, .col, use_na)

# S3 method for class 'SpatRaster'
clamp_predictors(x, training, .col, use_na = FALSE)

# S3 method for class 'SpatRasterDataset'
clamp_predictors(x, training, .col, use_na = FALSE)
```

Arguments

- `x`: a `terra::SpatRaster` or `terra::SpatRasterDataset` to clamp.
- `training`: the training dataset (a `data.frame` or a `sf::sf` object).
- `col`: the column containing the presences (optional). If specified, it is excluded from the clamping.
- `use_na`: a boolean determining whether values outside the range of the training dataset are removed (set to NA). If FALSE (the default), values outside the training range are replaced with the extremes of the training range.

Value

A `terra::SpatRaster` or `terra::SpatRasterDataset` clamped to the ranges in training.

---

collect_metrics.simple_ensemble  
Obtain and format results produced by tuning functions for ensemble objects

Description

Return a tibble of performance metrics for all models.
## S3 method for class 'simple_ensemble'
collect_metrics(x, ...)

## S3 method for class 'repeat_ensemble'
collect_metrics(x, ...)

### Arguments

- **x**: A `simple_ensemble` or `repeat_ensemble` object
- **...**: Not currently used.

### Details

When applied to an ensemble, the metrics that are returned do not contain the actual tuning parameter columns and values (unlike when these collect functions are run on other objects). The reason is that ensembles contain different types of models or models with different tuning parameters.

### Value

A tibble.

### See Also

`tune::collect_metrics()`

### Examples

```r
collect_metrics(lacerta_ensemble)
collect_metrics(lacerta_rep_ens)
```

---

### Description

Supply these light wrappers as the control argument in a `tune::tune_grid()`, `tune::tune_bayes()`, or `tune::fit_resamples()` call to return the needed elements for use in an ensemble. These functions will return the appropriate control grid to ensure that assessment set predictions and information on model specifications and preprocessors, are supplied in the resampling results object!

To integrate ensemble settings with your existing control settings, note that these functions just call the appropriate `tune::control_*` function with the arguments `save_pred = TRUE`, `save_workflow = TRUE`. These wrappers are equivalent to the ones used in the `stacks` package.
Usage

ccontrol_ensemble_grid()

ccontrol_ensemble_resamples()

ccontrol_ensemble_bayes()

Value

A tune::control_grid, tune::control_bayes, or tune::control_resamples object.

See Also

See the vignettes for examples of these functions used in context.

---

dist_pres_vs_bg | Distance between the distribution of climate values for presences vs background

Description

For each environmental variable, this function computes the density functions of presences and absences and returns (1-overlap), which is a measure of the distance between the two distributions. Variables with a high distance are good candidates for SDMs, as species occurrences are confined to a subset of the available background.

Usage

dist_pres_vs_bg(.data, .col)

Arguments

.data a data.frame (or derived object, such as tibble, or sf) with values for the bioclimatic variables for presences and background

.col the column containing the presences; it assumes presences to be the first level of this factor

Value

a name vector of distances
explain_tidysdm

Examples

# This should be updated to use a dataset from tidysdm
data("bradypus", package = "maxnet")
bradypus_tb <- tibble::as_tibble(bradypus) %>%
dplyr::mutate(presence = relevel(
    factor(
      dplyr::case_match(presence, 1 ~ "presence", 0 ~ "absence")
    ),
    ref = "presence"
  )) %>%
select(-ecoreg)
bradypus_tb %>% dist_pres_vs_bg(presence)

explain_tidysdm

Create explainer from your tidysdm ensembles.

Description

DALEX is designed to explore and explain the behaviour of Machine Learning methods. This function creates a DALEX explainer (see DALEX::explain()), which can then be queried by multiple function to create explanations of the model.

Usage

explain_tidysdm(
  model,
  data,
  y,
  predict_function,
  predict_function_target_column,
  residual_function,
  ...,
  label,
  verbose,
  precalculate,
  colorize,
  model_info,
  type,
  by_workflow
)

## Default S3 method:
explain_tidysdm(
  model,
  data = NULL,
y = NULL,
predict_function = NULL,
predict_function_target_column = NULL,
residual_function = NULL,
..., label = NULL,
verbose = TRUE,
precalculate = TRUE,
colorize = !isTRUE(getOption("knitr.in.progress")),
model_info = NULL,
type = "classification",
by_workflow = FALSE
)

## S3 method for class 'simple_ensemble'
explain_tidysdm(
  model,
  data = NULL,
  y = NULL,
  predict_function = NULL,
predict_function_target_column = NULL,
  residual_function = NULL,
  ..., label = NULL,
  verbose = TRUE,
  precalculate = TRUE,
colorize = !isTRUE(getOption("knitr.in.progress")),
model_info = NULL,
type = "classification",
by_workflow = FALSE
)

## S3 method for class 'repeat_ensemble'
explain_tidysdm(
  model,
  data = NULL,
  y = NULL,
predict_function = NULL,
predict_function_target_column = NULL,
  residual_function = NULL,
  ..., label = NULL,
  verbose = TRUE,
  precalculate = TRUE,
colorize = !isTRUE(getOption("knitr.in.progress")),
model_info = NULL,
type = "classification",
by_workflow = FALSE
)
Arguments

- **model**: object - a model to be explained
- **data**: data.frame or matrix - data which will be used to calculate the explanations. If not provided, then it will be extracted from the model. Data should be passed without a target column (this shall be provided as the `y` argument). NOTE: If the target variable is present in the data, some of the functionalities may not work properly.
- **y**: numeric vector with outputs/scores. If provided, then it shall have the same size as `data`
- **predict_function**: function that takes two arguments: model and new data and returns a numeric vector with predictions. By default it is `yhat`
- **predict_function_target_column**: Character or numeric containing either column name or column number in the model prediction object of the class that should be considered as positive (i.e. the class that is associated with probability 1). If NULL, the second column of the output will be taken for binary classification. For a multiclass classification setting, that parameter cause switch to binary classification mode with one vs others probabilities.
- **residual_function**: function that takes four arguments: model, data, target vector `y` and predict function (optionally). It should return a numeric vector with model residuals for given data. If not provided, response residuals ($y - \hat{y}$) are calculated. By default it is `residual_function_default`
- **...**: other parameters
- **label**: character - the name of the model. By default it’s extracted from the ‘class’ attribute of the model
- **verbose**: logical. If TRUE (default) then diagnostic messages will be printed
- **precalculate**: logical. If TRUE (default) then `predicted_values` and `residual` are calculated when explainer is created. This will happen also if `verbose` is TRUE. Set both `verbose` and `precalculate` to FALSE to omit calculations.
- **colorize**: logical. If TRUE (default) then WARNINGS, ERRORS and NOTES are colorized. Will work only in the R console. Now by default it is FALSE while knitting and TRUE otherwise.
- **model_info**: a named list (package, version, type) containing information about model. If NULL, DALEX will seek for information on it’s own.
- **type**: type of a model, either classification or regression. If not specified then type will be extracted from `model_info`.
- **by_workflow**: boolean determining whether a list of explainer, one per model, should be returned instead of a single explainer for the ensemble

Value

explainer object `DALEX::explain` ready to work with DALEX
### Examples

```r
# using the whole ensemble
lacerta_explainer <- explain_tidysdm(tidysdm::lacerta_ensemble)
# by workflow
explainer_list <- explain_tidysdm(tidysdm::lacerta_ensemble,
by_workflow = TRUE
)
```

---

#### extrapol_mess

**Multivariate environmental similarity surfaces (MESS)**

#### Description

Compute multivariate environmental similarity surfaces (MESS), as described by Elith et al., 2010.

#### Usage

```r
extrapol_mess(x, training, .col, ...)
```

- **x**: terra::SpatRaster, terra::SpatRasterDataset or data.frame
- **training**: matrix or data.frame or sf object containing the reference values; each column should correspond to one layer of the terra::SpatRaster object, with the exception of the presences column defined in .col (optional).
- **.col**: the column containing the presences (optional). If specified, it is excluded when computing the MESS scores.
- **...**: additional arguments as for terra::writeRaster()
- **filename**: character. Output filename (optional)
Details
This function is a modified version of mess in package predicts, with a method added to work on terra::SpatRasterDataset. Note that the method for terra::SpatRasterDataset assumes that each variables is stored as a terra::SpatRaster with time information within x. Time is also assumed to be in years. If these conditions are not met, it is possible to manually extract a terra::SpatRaster for each time step, and use extrapol_mess on those terra::SpatRasters.

Value
a terra::SpatRaster (data.frame) with the MESS values.

Author(s)
Jean-Pierre Rossi, Robert Hijmans, Paulo van Breugel, Andrea Manica

References

Description
This method finds a subset of variables that have low collinearity. It provides three methods: cor_caret, a stepwise approach to remove variables with a pairwise correlation above a given cutoff, choosing the variable with the greatest mean correlation (based on the algorithm in caret::findCorrelation); vif_step, a stepwise approach to remove variables with an variance inflation factor above a given cutoff (based on the algorithm in usdm::vifstep), and vif_cor, a stepwise approach that, at each step, find the pair of variables with the highest correlation above the cutoff and removes the one with the largest vif. such that all have a correlation below a certain cutoff. There are methods for terra::SpatRaster, data.frame and matrix. For terra::SpatRaster and data.frame, only numeric variables will be considered.

Usage
filter_collinear(
  x,
  cutoff = NULL,
  verbose = FALSE,
  names = TRUE,
  to_keep = NULL,
  method = "cor_caret",
  cor_type = "pearson",
  max_cells = Inf,
  ...
)
filter_collinear

## Default S3 method:
filter_collinear(
  x,
  cutoff = NULL,
  verbose = FALSE,
  names = TRUE,
  to_keep = NULL,
  method = "cor_caret",
  cor_type = "pearson",
  max_cells = Inf,
  ...
)

## S3 method for class 'SpatRaster'
filter_collinear(
  x,
  cutoff = NULL,
  verbose = FALSE,
  names = TRUE,
  to_keep = NULL,
  method = "cor_caret",
  cor_type = "pearson",
  max_cells = Inf,
  exhaustive = FALSE,
  ...
)

## S3 method for class 'data.frame'
filter_collinear(
  x,
  cutoff = NULL,
  verbose = FALSE,
  names = TRUE,
  to_keep = NULL,
  method = "cor_caret",
  cor_type = "pearson",
  max_cells = Inf,
  ...
)

## S3 method for class 'matrix'
filter_collinear(
  x,
  cutoff = NULL,
  verbose = FALSE,
  names = TRUE,
to_keep = NULL,
method = "cor_caret",
cor_type = "pearson",
max_cells = Inf,
...
)

Arguments

- **x** A `terra::SpatRaster` object, a data.frame (with only numeric variables)
- **cutoff** A numeric value used as a threshold to remove variables. For, "cor_caret" and "vif_cor", it is the pair-wise absolute correlation cutoff, which defaults to 0.7. For "vif_step", it is the variable inflation factor, which defaults to 10
- **verbose** A boolean whether additional information should be provided on the screen
- **names** a logical; should the column names be returned TRUE or the column index FALSE?
- **to_keep** A vector of variable names that we want to force in the set (note that the function will return an error if the correlation among any of those variables is higher than the cutoff).
- **method** character. One of "cor_caret", "vif_cor" or "vif_step".
- **cor_type** character. For methods that use correlation, which type of correlation: "pearson", "kendall", or "spearman". Defaults to "pearson"
- **max_cells** positive integer. The maximum number of cells to be used. If this is smaller than ncell(x), a regular sample of x is used
- **...** additional arguments specific to a given object type
- **exhaustive** boolean. Used only for `terra::SpatRaster` when downsampling to max_cells, if we require the exhaustive approach in `terra::spatSample()`. This is only needed for rasters that are very sparse and not too large, see the help page of `terra::spatSample()` for details.

Value

A vector of names of columns that are below the correlation threshold (when names = TRUE), otherwise a vector of indices. Note that the indices are only for numeric variables (i.e. if factors are present, the indices do not take them into account).

Author(s)

for cor_caret: Original R code by Dong Li, modified by Max Kuhn and Andrea Manica; for vif_step and vif_cor, original algorithm by Babak Naimi, rewritten by Andrea Manica for tidysdm

References

filter_high_cor

Deprecated: Filter to retain only variables below a given correlation threshold

Description

THIS FUNCTION IS DEPRECATED. USE filter_collinear with method=cor_caret instead

Usage

filter_high_cor(x, cutoff = 0.7, verbose = FALSE, names = TRUE, to_keep = NULL)

## Default S3 method:
filter_high_cor(x, cutoff = 0.7, verbose = FALSE, names = TRUE, to_keep = NULL)

## S3 method for class 'SpatRaster'
filter_high_cor(x, cutoff = 0.7, verbose = FALSE, names = TRUE, to_keep = NULL)

## S3 method for class 'data.frame'
filter_high_cor(x, cutoff = 0.7, verbose = FALSE, names = TRUE, to_keep = NULL)

## S3 method for class 'matrix'
filter_high_cor(x, cutoff = 0.7, verbose = FALSE, names = TRUE, to_keep = NULL)

Arguments

x A terra::SpatRaster object, a data.frame (with only numeric variables), or a correlation matrix
cutoff A numeric value for the pair-wise absolute correlation cutoff
verbose A boolean for printing the details
names a logical; should the column names be returned TRUE or the column index FALSE)?
to_keep A vector of variable names that we want to force in the set (note that the function will return an error if the correlation among any of those variables is higher than the cutoff).

Details

This method finds a subset of variable such that all have a correlation below a certain cutoff. There are methods for terra::SpatRaster, data.frame, and to work directly on a correlation matrix that was previously estimated. For data.frame, only numeric variables will be considered. The algorithm is based on caret::findCorrelation, using the exact option. The absolute values of pair-wise correlations are considered. If two variables have a high correlation, the function looks at the mean absolute correlation of each variable and removes the variable with the largest mean absolute correlation.

There are several function in the package subselect that can also be used to accomplish the same goal but tend to retain more predictors.
**gam_formula**

Create a formula for gam

**Value**

A vector of names of columns that are below the correlation threshold (when `names = TRUE`), otherwise a vector of indices. Note that the indices are only for numeric variables (i.e. if factors are present, the indices do not take them into account).

**Description**

This function takes the formula from a recipe, and turns numeric predictors into smooths with a given k. This formula can be passed to a workflow or workflow set when fitting a gam.

**Usage**

```
gam_formula(object, k = 10)
```

**Arguments**

- `object`: a `recipes::recipe`, already trained
- `k`: the k value for the smooth

**Value**

a formula

---

**geom_split_violin**

Split violin geometry for ggplots

**Description**

This geometry displays the density distribution of two groups side by side, as two halves of a violin. Note that an empty x aesthetic has to be provided even if you want to plot a single variable (see example below).

**Usage**

```
geom_split_violin(
  mapping = NULL,
  data = NULL,
  stat = "ydensity",
  position = "identity",
  nudge = 0,
  ..., 
  draw_quantiles = NULL,
```

trim = TRUE,
    scale = "area",
    na.rm = FALSE,
    show.legend = NA,
    inherit.aes = TRUE
)

Arguments

mapping  Set of aesthetic mappings created by \texttt{aes()}. If specified and \texttt{inherit.aes = TRUE} (the default), it is combined with the default mapping at the top level of the plot. You must supply \texttt{mapping} if there is no plot mapping.

data      The data to be displayed in this layer. There are three options:
            If \texttt{NULL}, the default, the data is inherited from the plot data as specified in the call to \texttt{ggplot()}.
            A \texttt{data.frame}, or other object, will override the plot data. All objects will be fortified to produce a data frame. See \texttt{fortify()} for which variables will be created.
            A function will be called with a single argument, the plot data. The return value must be a \texttt{data.frame}, and will be used as the layer data. A function can be created from a formula (e.g. \texttt{~ head(.x, 10)}).

stat      Use to override the default connection between \texttt{ggplot2::geom_violin()} and \texttt{ggplot2::stat_ydensity()}.

position  A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The \texttt{position} argument accepts the following:
            • The result of calling a position function, such as \texttt{position_jitter()}. This method allows for passing extra arguments to the position.
            • A string naming the position adjustment. To give the position as a string, strip the function name of the \texttt{position_} prefix. For example, to use \texttt{position_jitter()}, give the position as "jitter".
            • For more information and other ways to specify the position, see the \texttt{layer} \texttt{position} documentation.

nudge     Add space between the half-violin and the middle of the space allotted to a given factor on the x-axis.

...       Other arguments passed on to \texttt{layer()}’s \texttt{params} argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the \texttt{position} argument, or aesthetics that are required can \texttt{not} be passed through \texttt{...}. Unknown arguments that are not part of the 4 categories below are ignored.
            • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, \texttt{colour = "red"} or \texttt{linewidth = 3}. The geom’s documentation has an \texttt{Aesthetics} section that lists the available options. The ‘required’ aesthetics cannot be passed on to the \texttt{params}. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `geom_density(geom = "area", outline.type = "both")`. The `geom`’s documentation lists which parameters it can accept.

- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The `stat`’s documentation lists which parameters it can accept.

- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as `key glyphs`, to change the display of the layer in the legend.

- `draw_quantiles` If `not(NULL)` (default), draw horizontal lines at the given quantiles of the density estimate.

- `trim` If `TRUE` (default), trim the tails of the violins to the range of the data. If `FALSE`, don’t trim the tails.

- `scale` if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width.

- `na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

- `show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

- `inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

Details

The implementation is based on https://stackoverflow.com/questions/35717353/split-violin-plot-with-ggplot2. Credit goes to @jan-jlx for providing a complete implementation on StackOverflow, and to Trang Q. Nguyen for adding the nudge parameter.

Value

- A `ggplot2::layer` object

Examples

```r
data("bradypus", package = "maxnet")
bradypus_tb <- tibble::as_tibble(bradypus) %>% dplyr::mutate(presence = relevel(
  factor(  
    dplyr::case_match(presence, 1 ~ "presence", 0 ~ "absence")  
  ),
  ref = "presence"  
))
ggplot(bradypus_tb, aes(  
  ...  
))```
```r
x = "",
y = cl6190_ann,
fill = presence
}) +
geom_split_violin(nudge = 0.01)
```

---

**grid_cellsize**: Get default grid cellsize for a given dataset

**Description**

This function facilitates using `spatialsample::spatial_block_cv` multiple times in an analysis. `spatialsample::spatial_block_cv` creates a grid based on the object in `data`. However, if spatial blocks are generated multiple times in an analysis (e.g. for a `spatial_initial_split()`, and then subsequently for cross-validation on the training dataset), it might be desirable to keep the same grid. By applying this function to the largest dataset, usually the full dataset before `spatial_initial_split()`. The resulting cellsize can be used as an option in `spatialsample::spatial_block_cv`.

**Usage**

```r
grid_cellsize(data, n = c(10, 10))
```

**Arguments**

- `data` : a `sf::sf` dataset used to size the grid
- `n` : the number of cells in the grid, defaults to `c(10,10)`, which is also the default for `sf::st_make_grid()`

**Value**

the cell size

---

**grid_offset**: Get default grid cellsize for a given dataset

**Description**

This function facilitates using `spatialsample::spatial_block_cv` multiple times in an analysis. `spatialsample::spatial_block_cv` creates a grid based on the object in `data`. However, if spatial blocks are generated multiple times in an analysis (e.g. for a `spatial_initial_split()`, and then subsequently for cross-validation on the training dataset), it might be desirable to keep the same grid. By applying this function to the largest dataset, usually the full dataset before `spatial_initial_split()`. The resulting cellsize can be used as an option in `spatialsample::spatial_block_cv`. 
Usage

grid_offset(data)

Arguments

data a sf::sf dataset used to size the grid

Value

the grid offset

---

**horses**

*Coordinates of radiocarbon dates for horses*

**Description**

Coordinates for presences of horses from 22k to 8k YBP.

**Usage**

horses

**Format**

An tibble with 1,297 rows and 3 variables:

- **latitude** latitudes in degrees
- **longitude** longitudes in degrees
- **time_bp** time in years before present

---

**kap_max**

*Maximum Cohen's Kappa*

**Description**

Cohen’s Kappa (*yardstick::kap()*) is a measure similar to *yardstick::accuracy()*, but it normalises the observed accuracy by the value that would be expected by chance (this helps for unbalanced cases when one class is predominant).
Usage

kap_max(data, ...)

## S3 method for class 'data.frame'
kap_max(
  data,
  truth,
  ..., 
  estimator = NULL,
  na_rm = TRUE,
  event_level = "first",
  case_weights = NULL
)

## S3 method for class 'sf'
kap_max(data, ...)

kap_max_vec(
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  event_level = "first",
  case_weights = NULL,
  ...
)

Arguments

data        Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

...         A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

truth      The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For .vec() functions, a factor vector.

estimator   One of "binary", "hand_till", "macro", or "macro_weighted" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The others are general methods for calculating multiclass metrics. The default will automatically choose "binary" if truth is binary, "hand_till" if truth has >2 levels and case_weights isn’t specified, or "macro" if truth has >2 levels and case_weights is specified (in which case "hand_till" isn’t well-defined).
`kap_max`

**na_rm**

A logical value indicating whether NA values should be stripped before the computation proceeds.

**event_level**

A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first"

**case_weights**

The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

**estimate**

If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.

**Details**

This function calibrates the probability threshold to classify presences to maximises kappa.

There is no multiclass version of this function, it only operates on binary predictions (e.g. presences and absences in SDMs).

**Value**

A tibble with columns `.metric`, `.estimator`, and `.estimate` and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups.

**References**


**See Also**

Other class probability metrics: `boyce_cont()`, `tss_max()`

**Examples**

```
kap_max(two_class_example, truth, Class1)
```
Convert a geographic distance from km to m

Description
This function takes distance in km and converts it into meters, the units generally used by geographic operations in R. This is a trivial conversion, but this function ensures that no zeroes are lost along the way!

Usage
km2m(x)

Arguments
x the number of km

Value
the number of meters

Examples
km2m(10000)
km2m(1)

Coordinates of presences for Iberian emerald lizard

Description
Coordinates for presences of *Lacerta schreiberi*. The variables are as follows:

Usage
lacerta

Format
An tibble with 1,297 rows and 3 variables:

**ID** ids from GBIF

**latitude** latitudes in degrees

**longitude** longitudes in degrees
**lacerta_ensemble**

**Description**

Ensemble SDM for *Lacerta schreiberi*, as generated in the vignette.

**Usage**

lacerta_ensemble

**Format**

A *simple_ensemble* object

---

**lacerta_rep_ens**

**Description**

Ensemble SDM for *Lacerta schreiberi*, as generated in the vignette.

**Usage**

lacerta_rep_ens

**Format**

A *repeat_ensemble* object

---

**lacertidae_background**  *Coordinates of presences for lacertidae in the Iberian peninsula*

**Description**

Coordinates for presences of lacertidae, used as background for the *lacerta* dataset. The variables are as follows:

**Usage**

lacertidae_background
Format
An tibble with 1,297 rows and 3 variables:

ID ids from GBIF
latitude latitudes in degrees
longitude longitudes in degrees

maxent

Description
maxent defines the MaxEnt model as used in Species Distribution Models. A good guide to how options of a MaxEnt model work can be found in https://onlinelibrary.wiley.com/doi/full/10.1111/j.1600-0587.2013.07872.x

Usage
maxent(
  mode = "classification",
  engine = "maxnet",
  feature_classes = NULL,
  regularization_multiplier = NULL
)

Arguments
mode A single character string for the type of model. The only possible value for this model is "classification".
engine A single character string specifying what computational engine to use for fitting. Currently only "maxnet" is available.
feature_classes character, continuous feature classes desired, either "default" or any subset of "lqpt" (for example, "lh")
regularization_multiplier numeric, a constant to adjust regularization

Value
a parsnip::model_spec for a maxent model
maxent_params

Parameters for maxent models

Description

These parameters are auxiliary to MaxEnt models using the "maxnet" engine. These functions are used by the tuning functions, and the user will rarely access them directly.

Usage

regularization_multiplier(range = c(0.5, 3), trans = NULL)

feature_classes(values = c("l", "lq", "lpq", "lqph", "lqph")

Arguments

range A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the transformed units.
niche_overlap

trans  A trans object from the scales package, such as scales::log10_trans() or scales::reciprocal_trans(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

values  For feature_classes(), a character string of any subset of "lqph" (for example, "lh")

Value

a param object that can be used for tuning.

Examples

regularization_multiplier()

value

niche_overlap(x, y, method = c("Schoener", "Hellinger"))

Arguments

x  a terra::SpatRaster with a single layer

y  a terra::SpatRaster with a single layer

method  a string (or vector of strings) taking values "Schoener" and "Hellinger"

Details

Note that Hellinger's distance is normalised by dividing by square root of 2 (which is the correct asymptote for Hellinger's D), rather than the incorrect 2 used originally in Warren et al (2008), based on the Erratum for that paper.

Value

a list of overlap metrics, with slots D and I (depending on method)

References

**optim_thresh**

*Find threshold that optimises a given metric*

**Description**

This function returns the threshold to turn probabilities into binary classes whilst optimising a given metric. Currently available for `tss_max`, `kap_max` and sensitivity (for which a target sensitivity is required).

**Usage**

```r
optim_thresh(truth, estimate, metric, event_level = "first")
```

**Arguments**

- **truth**: The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a factor vector.
- **estimate**: the predicted probability for the event
- **metric**: character of metric to be optimised. Currently only "tss_max", "kap_max", and "sensitivity" with a given target (e.g. c("sensitivity",0.8))
- **event_level**: A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first"

**Value**

the probability threshold for the event

**Examples**

```r
optim_thresh(two_class_example$truth, two_class_example$Class1, metric = c("tss_max"))
optim_thresh(two_class_example$truth, two_class_example$Class1, metric = c("sens", 0.9))
```

**plot_pres_vs_bg**

*Plot presences vs background*

**Description**

Create a composite plots contrasting the distribution of multiple variables for presences vs the background.

**Usage**

```r
plot_pres_vs_bg(.data, .col)
```
predict.repeat_ensemble

Arguments
.data a data.frame (or derived object, such as tibble::tibble, or sf::st_sf) with values for the bioclimatic variables for presences and background
.col the column containing the presences; it assumes presences to be the first level of this factor

Value
a patchwork composite plot

Examples

data("bradypus", package = "maxnet")
bradypus_tb <- tibble::as_tibble(bradypus) %>%
dplyr::mutate(presence = relevel(
  factor(
    dplyr::case_match(presence, 1 ~ "presence", 0 ~ "absence")
  ),
  ref = "presence"
)) %>%
select(-ecoreg)
bradypus_tb %>% plot_pres_vs_bg(presence)

predict.repeat_ensemble
Predict for a repeat ensemble set

Description
Predict for a new dataset by using a repeat ensemble. Predictions from individual models are combined according to fun

Usage

## S3 method for class 'repeat_ensemble'
predict(
  object,                          
  new_data,                         
  type = "prob",                    
  fun = "mean",                     
  metric_thresh = NULL,             
  class_thresh = NULL,              
  members = FALSE,                  
  ...                               
)
**predict.simple_ensemble**

**Arguments**

- **object**
  - an `repeat_ensemble` object

- **new_data**
  - a data frame in which to look for variables with which to predict.

- **type**
  - the type of prediction, "prob" or "class".

- **fun**
  - string defining the aggregating function. It can take values `mean`, `median`, `weighted_mean`, `weighted_median` and `none`. It is possible to combine multiple functions, except for "none". If it is set to "none", only the individual member predictions are returned (this automatically sets `member` to TRUE).

- **metric_thresh**
  - a vector of length 2 giving a metric and its threshold, which will be used to prune which models in the ensemble will be used for the prediction. The 'metrics' need to have been computed when the workflow was tuned. Examples are `c("accuracy",0.8)` or `c("boyce_cont",0.7)`.

- **class_thresh**
  - probability threshold used to convert probabilities into classes. It can be a number (between 0 and 1), or a character metric (currently "tss_max" or "sensitivity"). For sensitivity, an additional target value is passed along as a second element of a vector, e.g. `c("sensitivity",0.8)`.

- **members**
  - boolean defining whether individual predictions for each member should be added to the ensemble prediction. The columns for individual members have the name of the workflow a a prefix, separated by "." from the usual column names of the predictions.

- ... not used in this method.

**Value**

- a tibble of predictions

---

**Usage**

```r
## S3 method for class 'simple_ensemble'
predict(
  object,
  new_data,
  type = "prob",
  fun = "mean",
  metric_thresh = NULL,
```

**Description**

Predict for a new dataset by using a simple ensemble. Predictions from individual models (i.e. workflows) are combined according to `fun`.
predict_raster

class_thresh = NULL,
members = FALSE,
...
)

Arguments

object an simple_ensemble object
new_data a data frame in which to look for variables with which to predict.
type the type of prediction, "prob" or "class".
fun string defining the aggregating function. It can take values mean, median, weighted_mean, weighted_median and none. It is possible to combine multiple functions, except for "none". If it is set to "none", only the individual member predictions are returned (this automatically sets member to TRUE)
metric_thresh a vector of length 2 giving a metric and its threshold, which will be used to prune which models in the ensemble will be used for the prediction. The 'metrics' need to have been computed when the workflow was tuned. Examples are c("accuracy",0.8) or c("boyce_cont",0.7)
class_thresh probability threshold used to convert probabilities into classes. It can be a number (between 0 and 1), or a character metric (currently "tss_max" or "sensitivity"). For sensitivity, an additional target value is passed along as a second element of a vector, e.g. c("sensitivity",0.8).
members boolean defining whether individual predictions for each member should be added to the ensemble prediction. The columns for individual members have the name of the workflow a a prefix, separated by "." from the usual column names of the predictions.
...

Value

a tibble of predictions

predict_raster Make predictions for a whole raster

Description

This function allows to use a raster as data to make predictions from a variety of tidymodels objects, such as simple_ensemble or stacks::stacks

Usage

predict_raster(object, raster, ...)

## Default S3 method:
predict_raster(object, raster, ...)
Arguments

- **object**: the tidymodels object of interest
- **raster**: the `terra::SpatRaster` with the input data. It has to include levels with the same names as the variables used in `object`
- ... parameters to be passed to the standard `predict()` function for the appropriate object type (e.g. `metric_thresh` or `class_thresh`).

Value

- a `terra::SpatRaster` with the predictions

**Description**

`tidysdm` provides specialised metrics for SDMs, which have their own help pages (`boyce_cont()`, `kap_max()`, and `tss_max()`). Additionally, it also provides methods to handle `sf::sf` objects for the following standard `yardstick` metrics:

- `yardstick::average_precision()`
- `yardstick::brier_class()
- `yardstick::classification_cost()
- `yardstick::gain_capture()
- `yardstick::mn_log_loss()
- `yardstick::pr_auc()
- `yardstick::roc_auc()
- `yardstick::roc_aunp()
- `yardstick::roc_aunu()

**Usage**

```r
## S3 method for class 'sf'
average_precision(data, ...)
## S3 method for class 'sf'
brier_class(data, ...)
## S3 method for class 'sf'
classification_cost(data, ...)
## S3 method for class 'sf'
gain_capture(data, ...)
```
## Recipe for sf objects

### Arguments

- **data**: an sf::sf object
- **...**: any other parameters to pass to the data.frame version of the metric. See the specific man page for the metric of interest.

### Details

Note that roc_aunp and roc_aunu are multiclass metrics, and as such are not relevant for SDMs (which work on a binary response). They are included for completeness, so that all class probability metrics from yardstick have an sf method, for applications other than SDMs.

### Value

A tibble with columns `.metric`, `.estimator`, and `.estimate` and 1 row of values.

### Description

This method for recipes::recipe() handles the case when x is an sf::sf object, as commonly used in Species Distribution Model, and generates a spatial_recipe.

### Usage

```r
## S3 method for class 'sf'
recipe(x, ...)

spatial_recipe(x, ...)
```
Arguments

- **x**: An sf::sf data frame.
- **...**: parameters to be passed to recipes::recipe()

Details

recipes::recipe() are not natively compatible with sf::sf objects. The problem is that the geometry column of sf::sf objects is a list, which is incompatible with the translation of formulae in recipes::recipe(). This method strips the geometry column from the data.frame and replaces it with a simple X and Y columns before any further operations, thus allowing the usual processing by recipes::recipe() to succeed (X and Y are give the role of coords in a spatial recipe). When prepping and baking a spatial_recipe, if a data.frame or tibble without coordinates is used as training or new_data, dummy X and Y columns are generated and filled with NAs. NOTE that order matters! You need to use the syntax recipe(x=sf_obj, formula=class~.) for the method to successfully detect the sf::sf object. Starting with formula will fail.

Value

An object of class spatial_recipe, which is a derived version of recipes::recipe(), see the manpage for recipes::recipe() for details.

Description

An ensemble based multiple sets of pseudoabsences/background. This object is a collection (list) of simple_ensemble objects for which predictions will be combined in a simple way (e.g. by taking either the mean or median). Each simple_ensemble contains the best version of a each given model type following turning; all simple ensembles will need to have the same metric estimated during the cv process.

Usage

repeat_ensemble(...)

Arguments

- **...**: not used, this function just creates an empty repeat_ensemble object. Members are added with add_best_candidates()

Value

an empty repeat_ensemble
sample_background  Sample background points for SDM analysis

Description

This function samples background points from a raster given a set of presences. The locations returned as the center points of the sampled cells, which can overlap with the presences (in contrast to pseudo-absences, see sample_pseudoabs). The following methods are implemented:

- 'random': background randomly sampled from the region covered by the raster (i.e. not NAs).
- 'dist_max': background randomly sampled from the unioned buffers of 'dist_max' from presences (distances in 'm' for lonlat rasters, and in map units for projected rasters). Using the union of buffers means that areas that are in multiple buffers are not oversampled. This is also referred to as "thickening".
- 'bias': background points are sampled according to a surface representing the biased sampling effort.

Usage

sample_background(
  data,
  raster,
  n,
  coords = NULL,
  method = "random",
  class_label = "background",
  return_pres = TRUE
)

Arguments

data An sf::sf data frame, or a data frame with coordinate variables. These can be defined in coords, unless they have standard names (see details below).
raster the terra::SpatRaster from which cells will be sampled (the first layer will be used to determine which cells are NAs, and thus can not be sampled). If sampling is "biased", then the sampling probability will be proportional to the values on the first layer (i.e. band) of the raster.
n number of background points to sample.
coords a vector of length two giving the names of the "x" and "y" coordinates, as found in data. If left to NULL, the function will try to guess the columns based on standard names c("x", "y"), c("X", "Y"), c("longitude", "latitude"), or c("lon", "lat").
method sampling method. One of 'random', 'dist_max', and 'targeted'. For dist_max, the maximum distance is set as an additional element of a vector, e.g c('dist_max',70000).
class_label the label given to the sampled points. Defaults to background
return_pres return presences together with background in a single tibble.
sample_background_time

Details

Note that the units of distance depend on the projection of the raster.

Value

An object of class tibble::tibble. If presences are returned, the presence level is set as the reference (to match the expectations in the yardstick package that considers the first level to be the event).

Description

This function samples background points from a raster given a set of presences. The locations returned as the center points of the sampled cells, which can overlap with the presences (in contrast to pseudo-absences, see sample_pseudoabs_time). The following methods are implemented:

- 'random': background points randomly sampled from the region covered by the raster (i.e. not NAs).
- 'dist_max': background points randomly sampled from the unioned buffers of 'dist_max' from presences (distances in 'm' for lonlat rasters, and in map units for projected rasters). Using the union of buffers means that areas that are in multiple buffers are not oversampled. This is also referred to as "thickening".
- 'bias': background points are sampled according to a surface representing the biased sampling effort. Note that the surface for each time step is normalised to sum to 1; use n_per_time_step to affect sampling effort within each time step.

Usage

```r
sample_background_time(
  data,
  raster,
  n_per_time_step,
  coords = NULL,
  time_col = "time",
  lubridate_fun = as,
  method = "random",
  class_label = "background",
  return_pres = TRUE,
  time_buffer = 0
)
```
Arguments

data  An sf::sf data frame, or a data frame with coordinate variables. These can be defined in coords, unless they have standard names (see details below).
raster the terra::SpatRaster or terra::SpatRasterDataset from which cells will be sampled. If a terra::SpatRasterDataset, the first dataset will be used to define which cells are valid, and which are NAs.
n_per_time_step number of background points to sample for each time step (i.e. a vector of length equal to the number of time steps in raster)
coords a vector of length two giving the names of the "x" and "y" coordinates, as found in data. If left to NULL, the function will try to guess the columns based on standard names c("x", "y"), c("X", "Y"), c("longitude", "latitude"), or c("lon", "lat")
time_col The name of the column with time; if time is not a lubridate object, use lubridate_fun to provide a function that can be used to convert appropriately
lubridate_fun function to convert the time column into a lubridate object
method sampling method. One of 'random', 'dist_min', 'dist_max', or 'dist_disc'.
class_label the label given to the sampled points. Defaults to background
return_pres return presences together with background in a single tibble
time_buffer the buffer on the time axis around presences that defines their effect when sampling background with method 'max_dist'. If set to zero, presences have an effect only on the time step to which they are assigned in raster; if a positive value, it defines the number of days before and after the date provided in the time column for which the presence should be considered (e.g. 20 days means that a presence is considered in all time steps equivalent to plus and minus twenty days from its date).

Value

An object of class tibble::tibble. If presences are returned, the presence level is set as the reference (to match the expectations in the yardstick package that considers the first level to be the event)

Description

This function samples pseudo-absence points from a raster given a set of presences. The locations returned as the center points of the sampled cells, which can not overlap with the presences (in contrast to background points, see sample_background). The following methods are implemented:

- 'random': pseudo-absences randomly sampled from the region covered by the raster (i.e. not NAs).
• ‘dist_min’: pseudo-absences randomly sampled from the region excluding a buffer of 'dist_min'
from presences (distances in ‘m’ for lonlat rasters, and in map units for projected rasters).
• ‘dist_max’: pseudo-absences randomly sampled from the unioned buffers of 'dist_max' from
presences (distances in ‘m’ for lonlat rasters, and in map units for projected rasters). Using
the union of buffers means that areas that are in multiple buffers are not oversampled. This is
also referred to as "thickening".
• ‘dist_disc’: pseudo-absences randomly sampled from the unioned discs around presences with
the two values of 'dist_disc’ defining the minimum and maximum distance from presences.

Usage

```r
sample_pseudoabs(
  data, 
  raster, 
  n, 
  coords = NULL, 
  method = "random", 
  class_label = "pseudoabs", 
  return_pres = TRUE
)
```

Arguments

data An sf::sf data frame, or a data frame with coordinate variables. These can be
defined in coords, unless they have standard names (see details below).
raster the terra::SpatRaster from which cells will be sampled
n number of pseudoabsence points to sample
coords a vector of length two giving the names of the "x" and "y" coordinates, as found
in data. If left to NULL, the function will try to guess the columns based on
standard names c("x", "y"), c("X", "Y"), c("longitude", "latitude"), or
c("lon", "lat")
method sampling method. One of 'random', 'dist_min', 'dist_max', or 'dist_disc'. Threshold
distances are set as additional elements of a vector, e.g c('dist_min',70000)
or c('dist_disc',50000,200000).
class_label the label given to the sampled points. Defaults to pseudoabs
return_pres return presences together with pseudoabsences in a single tibble

Value

An object of class tibble::tibble. If presences are returned, the presence level is set as the reference
(to match the expectations in the yardstick package that considers the first level to be the event)
sample_pseudoabs_time  

Sample pseudo-absence points for SDM analysis for points with a time point.

Description

This function samples pseudo-absence points from a raster given a set of presences. The locations returned as the center points of the sampled cells, which can not overlap with the presences (in contrast to background points, see sample_background_time). The following methods are implemented:

- 'random': pseudo-absences randomly sampled from the region covered by the raster (i.e. not NAs).
- 'dist_min': pseudo-absences randomly sampled from the region excluding a buffer of 'dist_min' from presences (distances in 'm' for lonlat rasters, and in map units for projected rasters).
- 'dist_max': pseudo-absences randomly sampled from the unioned buffers of 'dist_max' from presences (distances in 'm' for lonlat rasters, and in map units for projected rasters). Using the union of buffers means that areas that are in multiple buffers are not oversampled. This is also referred to as "thickening".
- 'dist_disc': pseudo-absences randomly sampled from the unioned discs around presences with the two values of 'dist_disc' defining the minimum and maximum distance from presences.

Usage

```r
sample_pseudoabs_time(
  data,  
  raster,  
  n_per_presence,  
  coords = NULL,  
  time_col = "time",  
  lubridate_fun = c,  
  method = "random",  
  class_label = "pseudoabs",  
  return_pres = TRUE,  
  time_buffer = 0  
)
```

Arguments

data  
An sf::sf data frame, or a data frame with coordinate variables. These can be defined in coords, unless they have standard names (see details below).

raster  
the terra::SpatRaster or terra::SpatRasterDataset from which cells will be sampled. If a terra::SpatRasterDataset, the first dataset will be used to define which cells are valid, and which are NAs.

n_per_presence  
number of pseudoabsence points to sample for each presence
sdm_metric_set

Description

This function returns a yardstick::metric_set that includes boyce_cont(), yardstick::roc_auc() and tss_max(), the most commonly used metrics for SDM.

Usage

sdm_metric_set(...)

Arguments

... additional metrics to be added to the yardstick::metric_set. See the help to yardstick::metric_set() for constraints on the type of metrics that can be mixed.

Value

a yardstick::metric_set object.

Examples

sdm_metric_set()
sdm_metric_set(accuracy)
sdm_spec_boost_tree  

Model specification for a Boosted Trees model for SDM

Description

This function returns a `parsnip::model_spec` for a Boosted Trees model to be used as a classifier of presences and absences in Species Distribution Model. It uses the library `xgboost` to fit boosted trees; to use another library, simply build the `parsnip::model_spec` directly.

Usage

```r
sdm_spec_boost_tree(..., tune = c("sdm", "all", "custom", "none"))
```

Arguments

- `...` parameters to be passed to `parsnip::boost_tree()` to customise the model. See the help of that function for details.
- `tune` character defining the tuning strategy. Valid strategies are:
  - "sdm" chooses hyperparameters that are most important to tune for an sdm (for `boost_tree`: 'mtry', 'trees', 'tree_depth', 'learn_rate', 'loss_reduction', and 'stop_iter')
  - "all" tunes all hyperparameters (for `boost_tree`: 'mtry', 'trees', 'tree_depth', 'learn_rate', 'loss_reduction', 'stop_iter', 'min_n' and 'sample_size')
  - "custom" passes the options from `...`
  - "none" does not tune any hyperparameter

Value

a `parsnip::model_spec` of the model.

See Also

Other "sdm model specifications": `sdm_spec_gam()`, `sdm_spec_glm()`, `sdm_spec_maxent()`, `sdm_spec_rand_forest()`

Examples

```r
standard_bt_spec <- sdm_spec_boost_tree()
full_bt_spec <- sdm_spec_boost_tree(tune = "all")
custom_bt_spec <- sdm_spec_boost_tree(tune = "custom", mtry = tune())
```
Model specification for a GAM for SDM

Description

This function returns a `parsnip::model_spec` for a General Additive Model to be used as a classifier of presences and absences in Species Distribution Model.

Usage

```r
sdm_spec_gam(..., tune = "none")
```

Arguments

- `...`: parameters to be passed to `parsnip::gen_additive_mod()` to customise the model. See the help of that function for details.
- `tune`: character defining the tuning strategy. As there are no hyperparameters to tune in a `gam`, the only valid option is "none". This parameter is present for consistency with other `sdm_spec_*` functions, but it does nothing in this case.

Value

A `parsnip::model_spec` of the model.

See Also

Other "sdm model specifications": `sdm_spec_boost_tree()`, `sdm_spec_glm()`, `sdm_spec_maxent()`, `sdm_spec_rand_forest()`

Examples

```r
my_gam_spec <- sdm_spec_gam()
```

---

Model specification for a GLM for SDM

Description

This function returns a `parsnip::model_spec` for a Generalised Linear Model to be used as a classifier of presences and absences in Species Distribution Model.

Usage

```r
sdm_spec_glm(..., tune = "none")
```
sdm_spec_maxent

Arguments

... parameters to be passed to `parsnip::logistic_reg()` to customise the model. See the help of that function for details.

tune character defining the tuning strategy. As there are no hyperparameters to tune in a `glm`, the only valid option is "none". This parameter is present for consistency with other `sdm_spec_` functions, but it does nothing in this case.

Value

a `parsnip::model_spec` of the model.

See Also

Other "sdm model specifications": `sdm_spec_boost_tree()`, `sdm_spec_gam()`, `sdm_spec_maxent()`, `sdm_spec_rand_forest()`

Examples

```r
my_spec_glm <- sdm_spec_glm()
```

---

sdm_spec_maxent Model specification for a MaxEnt for SDM

Description

This function returns a `parsnip::model_spec` for a MaxEnt model to be used in Species Distribution Models.

Usage

```r
sdm_spec_maxent(..., tune = c("sdm", "all", "custom", "none"))
```

Arguments

... parameters to be passed to `maxent()` to customise the model. See the help of that function for details.

tune character defining the tuning strategy. Valid strategies are:

- "sdm" chooses hyper-parameters that are most important to tune for an sdm (for `maxent`, 'mtry')
- "all" tunes all hyperparameters (for `maxent`, 'mtry', 'trees' and 'min')
- "custom" passes the options from `...`
- "none" does not tune any hyperparameter

Value

a `parsnip::model_spec` of the model.
sdm_spec_rand_forest

See Also

Other "sdm model specifications": sdm_spec_boost_tree(), sdm_spec_gam(), sdm_spec_glm(), sdm_spec_rand_forest()

Examples

test_maxent_spec <- sdm_spec_maxent(tune = "sdm")
test_maxent_spec
# setting specific values
sdm_spec_maxent(tune = "custom", feature_classes = "lq")

sdm_spec_rand_forest  Model specification for a Random Forest for SDM

Description

This function returns a parsnip::model_spec for a Random Forest to be used as a classifier of presences and absences in Species Distribution Models. It uses the library ranger to fit boosted trees; to use another library, simply build the parsnip::model_spec directly.

Usage

sdm_spec_rand_forest(..., tune = c("sdm", "all", "custom", "none"))

sdm_spec_rf(..., tune = c("sdm", "all", "custom", "none"))

Arguments

... parameters to be passed to parsnip::rand_forest() to customise the model. See the help of that function for details.
tune character defining the tuning strategy. Valid strategies are:

• "sdm" chooses hyperparameters that are most important to tune for an sdm (for rf, 'mtry')
• "all" tunes all hyperparameters (for rf, 'mtry', 'trees' and 'min')
• "custom" passes the options from '...
• "none" does not tune any hyperparameter

Details

sdm_spec_rf() is simply a short form for sm_spec_rand_forest().

Value

a parsnip::model_spec of the model.
simple_ensemble

See Also

Other "sdm model specifications": sdm_spec_boost_tree(), sdm_spec_gam(), sdm_spec_glm(), sdm_spec_maxent()

Examples

test_rf_spec <- sdm_spec_rf(tune = "sdm")
test_rf_spec
# combining tuning with specific values for other hyperparameters
sdm_spec_rf(tune = "sdm", trees = 100)

---

simple_ensemble  Simple ensemble

Description

A simple ensemble is a collection of workflows for which predictions will be combined in a simple way (e.g. by taking either the mean or median). Usually these workflows will consist of the best version of a given model algorithm following tuning. The workflows are fitted to the full training dataset before making predictions.

Usage

simple_ensemble(...)

Arguments

...  not used, this function just creates an empty simple_ensemble object. Members are added with add_best_candidates()

Value

an empty simple_ensemble. This is a tibble with columns:

- wflow_id: the name of the workflows for which the best model was chosen
- workflow: the trained workflow objects
- metrics: metrics based on the crossvalidation resampling used to tune the models
spatial_initial_split

Simple Training/Test Set Splitting for spatial data

Description

spatial_initial_split creates a single binary split of the data into a training set and testing set. All strategies from the package spatialsample are available; a random split from that strategy will be used to generate the initial split.

Usage

spatial_initial_split(data, prop, strategy, ...)

Arguments

data: A dataset (data.frame or tibble)
prop: The proportion of data to be retained for modelling/analysis.
strategy: A sampling strategy from spatialsample
...
parameters to be passed to the strategy

Value

An rsplit object that can be used with the rsample::training and rsample::testing functions to extract the data in each split.

Examples

set.seed(123)
block_initial <- spatial_initial_split(boston_canopy, prop = 1 / 5, spatial_block_cv)
testing(block_initial)
training(block_initial)

thin_by_cell

Thin point dataset to have 1 observation per raster cell

Description

This function thins a dataset so that only one observation per cell is retained.

Usage

thin_by_cell(data, raster, coords = NULL, drop_na = TRUE, agg_fact = NULL)
thin_by_cell_time

Arguments

data  An \texttt{sf::sf} data frame, or a data frame with coordinate variables. These can be defined in \texttt{coords}, unless they have standard names (see details below).
raster A \texttt{terra::SpatRaster} object that defined the grid
coords a vector of length two giving the names of the "x" and "y" coordinates, as found in \texttt{data}. If left to \texttt{NULL}, the function will try to guess the columns based on standard names \texttt{c("x", "y")}, \texttt{c("X", "Y")}, \texttt{c("longitude", "latitude")}, or \texttt{c("lon", "lat")}
drop_na boolean on whether locations that are NA in the raster should be dropped.
agg_fact positive integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor) or three integers (when also aggregating over layers). Defaults to \texttt{NULL}, which implies no aggregation (i.e. thinning is done on the grid of \texttt{raster})

Details

Further thinning can be achieved by aggregating cells in the raster before thinning, as achieved by setting \texttt{agg_fact} > 1 (aggregation works in a manner equivalent to \texttt{terra::aggregate()}).

Value

An object of class \texttt{sf::sf} or \texttt{data.frame}, the same as "data".

Description

This function thins a dataset so that only one observation per cell per time slice is retained. We use a raster with layers as time slices to define the data cube on which thinning is enforced (see details below on how time should be formatted).

Usage

\begin{verbatim}
thin_by_cell_time(
data,
raster,
coords = NULL,
time_col = "time",
lubridate_fun = \texttt{c},
drop_na = \texttt{TRUE},
agg_fact = \texttt{NULL})
\end{verbatim}
Arguments

data  An `sf::sf` data frame, or a data frame with coordinate variables. These can be defined in `coords`, unless they have standard names (see details below).

raster  A `terra::SpatRaster` object that defined the grid with layers corresponding to the time slices (times should be set as either POSIXlt or "years", see `terra::time()` for details), or a `terra::SpatRasterDataset` where the first dataset will be used (again, times for that dataset should be set as either POSIXlt or "years") `terra::time()`

coords  a vector of length two giving the names of the "x" and "y" coordinates, as found in `data`. If left to NULL, the function will try to guess the columns based on standard names `c("x", "y")`, `c("X", "Y")`, `c("longitude", "latitude")`, or `c("lon", "lat")`

time_col  The name of the column with time; if time is not a lubridate object, use lubridate_fun to provide a function that can be used to convert appropriately

lubridate_fun  function to convert the time column into a lubridate object

drop_na  boolean on whether locations that are NA in the raster should be dropped.

agg_fact  positive integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor) or three integers (when also aggregating over layers). Defaults to NULL, which implies no aggregation (i.e. thinning is done on the grid of raster)

Details

Further spatial thinning can be achieved by aggregating cells in the raster before thinning, as achieved by setting `agg_fact > 1` (aggregation works in a manner equivalent to `terra::aggregate()`).

Value

An object of class `sf::sf` or `data.frame`, the same as "data".

Description

This function thins a dataset so that only observations that have a distance from each other greater than "dist_min" are retained.

Usage

`thin_by_dist(data, dist_min, coords = NULL)`
Arguments

- **data**: An `sf::sf` data frame, or a data frame with coordinate variables. These can be defined in `coords`, unless they have standard names (see details below).
- **dist_min**: Minimum distance between points (in units appropriate for the projection, or meters for lonlat data).
- **coords**: A vector of length two giving the names of the "x" and "y" coordinates, as found in `data`. If left to NULL, the function will try to guess the columns based on standard names `c("x", "y")`, `c("X", "Y")`, `c("longitude", "latitude")`, or `c("lon", "lat")`

Details

Distances are measured in the appropriate units for the projection used. In case of raw latitude and longitude (e.g. as provided in a data.frame), the crs is set to WGS84, and units are set to meters.

This function is a modified version of the algorithm in spThin, adapted to work on sf objects.

Value

An object of class `sf::sf` or `data.frame`, the same as "data".

---

thin_by_dist_time  Thin points dataset based on geographic and temporal distance

---

Description

This function thins a dataset so that only observations that have a distance from each other greater than "dist_min" in space and "interval_min" in time are retained.

Usage

```r
thin_by_dist_time(
  data,
  dist_min,
  interval_min,
  coords = NULL,
  time_col = "time",
  lubridate_fun = c
)
```

Arguments

- **data**: An `sf::sf` data frame, or a data frame with coordinate variables. These can be defined in `coords`, unless they have standard names (see details below).
- **dist_min**: Minimum distance between points (in units appropriate for the projection, or meters for lonlat data).
interval_min  Minimum time interval between points, in days.
coords       A vector of length two giving the names of the "x" and "y" coordinates, as found in data. If left to NULL, the function will try to guess the columns based on standard names c("x", "y"), c("X", "Y"), c("longitude", "latitude"), or c("lon", "lat")
time_col     The name of the column with time; if time is not a lubridate object, use lubridate_fun to provide a function that can be used to convert appropriately
lubridate_fun function to convert the time column into a lubridate object

Details

Geographic distances are measured in the appropriate units for the projection used. In case of raw latitude and longitude (e.g. as provided in a data.frame), the crs is set to WGS84, and units are set to meters. Time interval are estimated in days. Note that for very long time period, the simple conversion x years = 365 * x days might lead to slightly shorter intervals than expected, as it ignores leap years. The function y2d() provides a closer approximation.

This function an algorithm analogous to spThin, with the exception that neighbours are defined in terms of both space and time.

Value

An object of class sf::sf or data.frame, the same as "data".

tss

TSS - True Skill Statistics

Description

The True Skills Statistic, which is defined as

Usage

tss(data, ...)

  # S3 method for class 'data.frame'
  tss(
    data,
    truth,
    estimate,
    estimator = NULL,
    na.rm = TRUE,
    case_weights = NULL,
    event_level = "first",
    ...
Arguments

data
Either a data.frame containing the columns specified by the truth and estimate arguments, or a table(matrix where the true class results should be in the columns of the table.

... Not currently used.

truth
The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate
The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

estimator
One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na.rm
A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights
The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

event_level
A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default is "first".

Details

\[ \text{sensitivity} + \text{specificity} + 1 \]

This function is a wrapper around yardstick::j_index(), another name for the same quantity. Note that this function takes the classes as predicted by the model without any calibration (i.e. making a split at 0.5 probability). This is usually not the metric used for Species Distribution Models, where the threshold is recalibrated to maximise TSS; for that purpose, use tss_max().

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups.

Examples

# Two class
data("two_class_example")
tss(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)
# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  tss(obs, pred)

tss_max

Maximum TSS - True Skill Statistics

Description

The True Skills Statistic, which is defined as

Usage

\[
tss\_max(data, \ldots)
\]

\[
\text{## S3 method for class 'data.frame'}
\]

tss_max(
  data,
  truth,
  \ldots,
  estimator = NULL,
  na.rm = TRUE,
  event_level = "first",
  case_weights = NULL
)

\[
\text{## S3 method for class 'sf'}
\]

tss_max(data, \ldots)

Arguments

- **data**: Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
- **\ldots**: A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level.
Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

**truth**
The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

**estimator**
One of "binary", "hand_till", "macro", or "macro_weighted" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The others are general methods for calculating multiclass metrics. The default will automatically choose "binary" if truth is binary, "hand_till" if truth has >2 levels and case_weights isn’t specified, or "macro" if truth has >2 levels and case_weights is specified (in which case "hand_till" isn’t well-defined).

**na_rm**
A logical value indicating whether NA values should be stripped before the computation proceeds.

**event_level**
A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first"

**case_weights**
The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

**estimate**
If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.

**Details**

\[ \text{sensitivity} + \text{specificity} + 1 \]

This function calibrates the probability threshold to classify presences to maximise the TSS.

There is no multiclass version of this function, it only operates on binary predictions (e.g. presences and absences in SDMs).

**Value**

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups.

**See Also**

Other class probability metrics: `boyce_cont()`, `kap_max()`

**Examples**

tss_max(two_class_example, truth, Class1)
y2d

Convert a time interval from years to days

Description

This function takes a time interval in years and converts into days, the unit commonly used in time operations in R. The simple conversion \( x \times 365 \) does not work for large number of years, due to the presence of leap years.

Usage

\[
y2d(x)
\]

Arguments

\[
x \quad \text{the number of years of the interval}
\]

Value

a \texttt{difftime} object (in days)

Examples

\[
y2d(1)
y2d(1000)
\]
Index

* class probability metrics
  boyce_cont, 8
  kap_max, 27
  tss_max, 59

* datasets
  horses, 27
  lacerta, 30
  lacerta_ensemble, 31
  lacerta_rep_ens, 31
  lacertidae_background, 31

* ensemble
  add_member, 3
  add_repeat, 4
  autoplot.simple_ensemble, 5

* extrapolation
  clamp_predictors, 12
  extrapol_mess, 18

* predict
  calib_class_thresh, 10
  predict.repeat_ensemble, 36
  predict.simple_ensemble, 37
  predict_raster, 38

add_member, 3
add_repeat, 4
aes(), 24
autoplot.simple_ensemble, 5
autoplot.spatial_initial_split, 6
average_precision.sf (prob_metrics_sf), 39

blockcv2rsample, 7
borders(), 25
boyce_cont, 8, 29, 60
boyce_cont(), 39, 47
boyce_cont_vec (boyce_cont), 8
brier_class.sf (prob_metrics_sf), 39

check_class_thresh, 10
check_sdm_presence, 10
check_splits_balance, 11
clamp_predictors, 12
classification_cost.sf
  (prob_metrics_sf), 39
collect_metrics.repeat_ensemble
  (collect_metrics.repeat_ensemble), 12
collect_metrics.simple_ensemble, 12
control_ensemble_bayes
  (control_ensemble_grid), 13
control_ensemble_grid, 13
control_ensemble_resamples
  (control_ensemble_grid), 13

DALEX::explain, 17
DALEX::explain(), 15
data.frame, 12, 18, 19, 22, 36, 41, 54–57
dist_pres_vs_bg, 14

explain_tidysdm, 15
extrapol_mess, 18

feature_classes (maxent_params), 33
filter_collinear, 19
filter_high_cor, 22
fortify(), 24

gain_capture.sf (prob_metrics_sf), 39
gam_formula, 23
geom_split_violin, 23
ggplot(), 24
ggplot2::geom_sf(), 6
ggplot2::geom_violin(), 24
ggplot2::layer, 25
ggplot2::stat_ydensity(), 24
grid_cells, 26
grid_offset, 26

horses, 27
kap_max, 9, 27, 35, 60
yardstick::accuracy().27
yardstick::average_precision().39
yardstick::brier_class().39
yardstick::classification_cost().39
yardstick::gain_capture().39
yardstick::j_index().58
yardstick::kap().27
yardstick::metric_set.47
yardstick::metric_set().47
yardstick::mn_log_loss().39
yardstick::pr_auc().39
yardstick::roc_auc().39, 47
yardstick::roc_aunp().39
yardstick::roc_aunu().39