Package 'tidysdm'

March 5, 2024

Title Species Distribution Models with Tidymodels

Version 0.9.4

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, as well as a number of specialised functions to process presences and pseudoabsences 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 7.2.3

Depends tidymodels, spatialsample, R (>= 3.50)

Imports dials, DALEX, DALEXtra, dplyr, ggplot2, lubridate, magrittr, maxnet, 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, rmarkdown, spelling, stacks, testthat (>= 3.0.0), tidyterra, xgboost, V8

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

NeedsCompilation no

Author Michela Leonardi [aut], Margherita Colucci [aut], Andrea Manica [aut, cre]

Maintainer Andrea Manica <am315@cam.ac.uk>

Repository CRAN

Date/Publication 2024-03-05 20:30:02 UTC

R topics documented:

add_member . . . 3
add_repeat . . . 4
autoplot.simple_ensemble . . . 4
autoplot.spatial_initial_split . . . 6
blockcv2rsample . . . 7
boyce_cont . . . 8
calib_class_thresh . . . 10
check_sdm_presence . . . 10
check_splits_balance . . . 11
collect_metrics.simple_ensemble . . . 12
control_ensemble_grid . . . 13
dist_pres_vs_bg . . . 13
explain_tidysdm . . . 14
filter_high_cor . . . 17
gam_formula . . . 18
geom_split_violin . . . 19
grid_cellsize . . . 21
grid_offset . . . 21
horses . . . 22
kap_max . . . 22
km2m . . . 24
lacerta . . . 25
lacerta_ensemble . . . 25
lacerta_rep_ens . . . 26
maxent . . . 26
maxent_params . . . 27
optim_thresh . . . 28
plot_pres_vs_bg . . . 29
predict.repeat_ensemble . . . 30
predict.simple_ensemble . . . 31
predict_raster . . . 32
prob_metrics_sf . . . 33
recipe.sf . . . 34
repeat_ensemble . . . 35
sample_pseudoabs . . . 35
sample_pseudoabs_time . . . 36
sdm_metric_set . . . 38
sdm_spec_boost_tree . . . 39
sdm_spec_gam . . . 39
sdm_spec_glm . . . 40
sdm_spec_maxent . . . 41

sdm_spec_rand_forest 41
simple_ensemble 42
spatial_initial_split 43
thin_by_cell 44
thin_by_cell_time 45
thin_by_dist 46
thin_by_dist_time 47
tidysdm 48
tss 48
tss_max 50
y2d 52

Index **53**

`add_member` *Add best member of workflow to a simple ensemble*

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

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

## Default S3 method:
add_member(x, member, ...)

## S3 method for class 'tune_results'
add_member(x, member, metric = NULL, id = NULL, ...)

## S3 method for class 'workflow_set'
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)

autoplot.simple_ensemble *Plot the results of a simple ensemble*

Description

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

Usage

```
## S3 method for class 'simple_ensemble'
autoplot(
  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 autoplot(). 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

```
# 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

```
## 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

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

`blockcv2rsample` *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

```
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

```
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_cont *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

```
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

Boyce, M.S., P.R. Vernier, S.E. Nielsen and F.K.A. Schmiegelow. 2002. Evaluating resource selection functions. Ecol. Model., 157, 281-300.

Hirzel, A.H., G. Le Lay, V. Helfer, C. Randin and A. Guisan. 2006. Evaluating the ability of habitat suitability models to predict species presences. Ecol. Model., 199, 142-152.

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

```
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. Examples are c("accuracy",0.8) or c("boyce_cont",0.7)

Value

a simple_ensemble object

Examples

```
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

`check_splits_balance` *Check the balance of presences vs pseudoabsences among splits*

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_bloc`
`.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)
```

`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.

Usage

```
## 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 a 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

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

control_ensemble_grid *Control wrappers*

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, is 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

```
control_ensemble_grid()

control_ensemble_resamples()

control_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 bioclimate 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

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

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

---

filter_high_cor *Filter to retain only variables below a given correlation threshold*

Description

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.

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)

filter_high_cor_algorithm(x, cutoff = 0.7, verbose = FALSE)
```

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

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

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).

---

gam_formula *Create a formula for gam*

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 `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`stat`	Use to override the default connection between `geom_violin()` and `stat_ydensity()`.
`position`	Position adjustment, either as a string naming the adjustment (e.g. `"jitter"` to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
`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 layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
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

```
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(
  x = "",
  y = cld6190_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

```
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).
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

Cohen, J. (1960). "A coefficient of agreement for nominal scales". *Educational and Psychological Measurement*. 20 (1): 37-46.

Cohen, J. (1968). "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit". *Psychological Bulletin*. 70 (4): 213-220.

See Also

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

Examples

```
kap_max(two_class_example, truth, Class1)
```

---

km2m *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 functions 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)
```

`lacerta` *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` *A simple ensemble for the lacerta data*

Description

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

Usage

```
lacerta_ensemble
```

Format

A `simple_ensemble` object

lacerta_rep_ens *A repeat ensemble for the lacerta data*

Description

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

Usage

```
lacerta_rep_ens
```

Format

A repeat_ensemble object

maxent *Maxent model*

Description

maxent defines a MaxEnt model for binary outcomes 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 "lqpht" (for example, "lh")
regularization_multiplier	numeric, a constant to adjust regularization

Value

a model_spec for a maxent model

Examples

```
# format the data
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)

# fit the model, and make some predictions
maxent_spec <- maxent(feature_classes = "lq")
maxent_fitted <- maxent_spec %>%
  fit(presence ~ ., data = bradypus_tb)
pred_prob <- predict(maxent_fitted, new_data = bradypus[, -1], type = "prob")
pred_class <- predict(maxent_fitted, new_data = bradypus[, -1], type = "class")

# Now with tuning
maxent_spec <- maxent(
  regularization_multiplier = tune(),
  feature_classes = tune()
)
set.seed(452)
cv <- vfold_cv(bradypus_tb, v = 2)
maxent_tune_res <- maxent_spec %>%
  tune_grid(presence ~ ., cv, grid = 3)
show_best(maxent_tune_res, metric = "roc_auc")
```

---

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", "lqp", "lqph", "lqpht"))
```

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.
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 "lqpht" (for example, "lh")

Value

a `param` object that can be used for tuning.

Examples

```
regularization_multiplier()
feature_classes()
```

---

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

```
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

```
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

```
plot_pres_vs_bg(.data, .col)
```

Arguments

`.data`	a `data.frame` (or derived object, such as `tibble::tibble`, or `sf::st_sf`) with values for the bioclimate 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,
  ...
)
```

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

`predict.simple_ensemble` *Predict for a simple ensemble set*

Description

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

Usage

```
## S3 method for class 'simple_ensemble'
predict(
  object,
  new_data,
  type = "prob",
  fun = "mean",
  metric_thresh = NULL,
  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.
...	not used in this method.

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::linear_stack`

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.

Value

a `terra::SpatRaster` with the predictions

prob_metrics_sf *Probability metrics for* sf *objects*

Description

tidysdm provides methods to handle sf::sf objects for the following 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

```
## 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, ...)

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

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

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

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

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

Arguments

data	an sf::sf object
...	any other parameters to pass to the data.frame version of the metric.

Value

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

---

recipe.sf *Recipe for* sf *objects*

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

```
## 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 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 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 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.

repeat_ensemble *Repeat ensemble*

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_pseudoabs *Sample pseudo-absence (or background) points for SDM analysis*

Description

This function samples pseudo-absence (or background, the naming is a matter of semantics) 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. The following methods are implemented:

- 'random': pseudo-absences/background randomly sampled from the region covered by the raster (i.e. not NAs).
- 'dist_min': pseudo-absences/background 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/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".
- 'dist_disc': pseudo-absences/background randomly sampled from the unioned discs around presences with the two values of 'dist_disc' defining the minimum and maximum distance from presences.

Usage

```
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/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_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/background 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 (or background) points for SDM analysis for points with a time point.*

Description

This function samples pseudo-absence (or background, the naming is a matter of semantics) 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. The following methods are implemented:

- 'random': pseudo-absences/background randomly sampled from the region covered by the raster (i.e. not NAs).

- 'dist_min': pseudo-absences/background 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/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".
- 'dist_disc': pseudo-absences/background randomly sampled from the unioned discs around presences with the two values of 'dist_disc' defining the minimum and maximum distance from presences.

Usage

```
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/background points to sample for each presence
`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 `pseudoabs`
`return_pres`	return presences together with pseudoabsences/background in a single tibble

time_buffer — the buffer on the time axis around presences that defines their effect when sampling pseudoabsences. 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)

---

sdm_metric_set *Metric set for SDM*

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

```
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.

Examples

```
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())
```

sdm_spec_gam *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

```
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.

Examples

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

---

sdm_spec_glm *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

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

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.

Examples

```
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 as a classifier of presences and absences in Species Distribution Models.

Usage

```
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 hyperparameters 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.

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.

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 consists each of the best version of a given model type following turning. 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)
```

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
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")
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 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".

thin_by_cell_time *Thin point dataset to have 1 observation per raster cell per time slice*

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

```
thin_by_cell_time(
  data,
  raster,
  coords = NULL,
  time_col = "time",
  lubridate_fun = c,
  drop_na = TRUE,
  agg_fact = 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).
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".

`thin_by_dist` *Thin points dataset based on geographic distance*

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

```
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".

`tidysdm` *tidysdm*

Description

This `R` library facilitates the fitting of Species Distribution Models with `tidymodels`.

Details

The functionalities of `tidysdm` are described in Leonardi et al. (2023) doi:10.1101/2023.07.24.550358. Please cite it if you use `tidysdm` in your research.

On its dedicated website, you can find Articles giving you a step-by-step overview of the package, how to use `tidysdm` on palaeodata, examples of advanced modelling approaches using `tidymodels` features, and a troubleshooting guide for when models fail. There is also a development version of the site updated for the `dev` version (on the top left, the version number is in red, and will be in the format x.x.x.9xxx, indicating it is a development version).

`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

sensitivity+*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, ...)

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

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

tss_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).

`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

sensitivity+*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 takes a time interval in years and converts into days, the unit commonly used in time operations in R. The simple conversion x * 365 does not work for large number of years, due to the presence of leap years.

Usage

```
y2d(x)
```

Arguments

x	the number of years of the interval

Value

a difftime object (in days)

Examples

```
y2d(1)
y2d(1000)
```

Index

∗ **class probability metrics**
  boyce_cont, 8
  kap_max, 22
  tss_max, 50
∗ **datasets**
  horses, 22
  lacerta, 25
  lacerta_ensemble, 25
  lacerta_rep_ens, 26

add_member, 3
add_repeat, 4
aes(), *19*
autoplot.simple_ensemble, 4
autoplot.spatial_initial_split, 6
average_precision.sf *(*prob_metrics_sf*)*, 33

blockcv2rsample, 7
borders(), *20*
boyce_cont, 8, *24*, *51*
boyce_cont(), *38*
boyce_cont_vec *(*boyce_cont*)*, 8
brier_class.sf *(*prob_metrics_sf*)*, 33

calib_class_thresh, 10
check_sdm_presence, 10
check_splits_balance, 11
classification_cost.sf *(*prob_metrics_sf*)*, 33
collect_metrics.repeat_ensemble *(*collect_metrics.simple_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(), *14*
data.frame, *17*, *29*, *34*, *44*, *46*, *48*
dist_pres_vs_bg, 13

explain_tidysdm, 14

feature_classes *(*maxent_params*)*, 27
filter_high_cor, 17
filter_high_cor_algorithm *(*filter_high_cor*)*, 17
fortify(), *19*

gain_capture.sf *(*prob_metrics_sf*)*, 33
gam_formula, 18
geom_split_violin, 19
geom_violin(), *19*
ggplot(), *19*
ggplot2::geom_sf(), *6*
ggplot2::layer, *20*
grid_cellsize, 21
grid_offset, 21

horses, 22

kap_max, *9*, 22, *28*, *51*
kap_max_vec *(*kap_max*)*, 22
km2m, 24

lacerta, 25
lacerta_ensemble, 25
lacerta_rep_ens, 26
layer(), *20*

maxent, *26*, 26
maxent(), *41*
maxent_params, 27
mn_log_loss.sf *(*prob_metrics_sf*)*, 33
model_spec, *27*

optim_thresh, 28

parsnip::boost_tree(), *39*

parsnip::gen_additive_mod(), *40*
parsnip::logistic_reg(), *40*
parsnip::model_spec, *39–42*
parsnip::rand_forest(), *42*
plot_pres_vs_bg, 29
pr_auc.sf *(*prob_metrics_sf*)*, 33
predict.repeat_ensemble, 30
predict.simple_ensemble, 31
predict_raster, 32
prob_metrics_sf, 33

recipe, *34*
recipe(), *34*
recipe.sf, 34
recipes, *34*
recipes::recipe, *18*, *34*
recipes::recipe(), *34*
regularization_multiplier *(*maxent_params*)*, 27
repeat_ensemble, *4*, *12*, *26*, 35
roc_auc.sf *(*prob_metrics_sf*)*, 33
roc_aunp.sf *(*prob_metrics_sf*)*, 33
roc_aunu.sf *(*prob_metrics_sf*)*, 33
rsample::testing, *43*
rsample::training, *43*

sample_pseudoabs, 35
sample_pseudoabs_time, 36
sdm_metric_set, 38
sdm_spec_boost_tree, 39
sdm_spec_gam, 39
sdm_spec_glm, 40
sdm_spec_maxent, 41
sdm_spec_rand_forest, 41
sdm_spec_rf *(*sdm_spec_rand_forest*)*, 41
sf::sf, *21*, *33*, *34*, *36*, *37*, *44–48*
sf::st_make_grid(), *21*
sf::st_sf, *29*
simple_ensemble, *3–5*, *10*, *12*, *25*, *32*, *35*, 42
simple_ensemble(), *3*
spatial_initial_split, 43
spatial_initial_split(), *21*
spatial_recipe *(*recipe.sf*)*, 34
spatialsample, *43*
spatialsample::spatial_block_cv, *21*
spatialsample::spatial_block_cv(), *11*
stat_ydensity(), *19*

terra::aggregate(), *44*, *46*
terra::SpatRaster, *17*, *18*, *32*, *36*, *37*, *44*, *45*
terra::SpatRasterDataset, *37*, *45*
terra::time(), *45*
thin_by_cell, 44
thin_by_cell_time, 45
thin_by_dist, 46
thin_by_dist_time, 47
tibble::tibble, *29*, *36*, *38*
tidymodels, *32*
tidysdm, 48
tss, 48
tss_max, *9*, *24*, *28*, 50
tss_max(), *38*, *49*
tss_max_vec *(*tss_max*)*, 50
tune::collect_metrics(), *12*
tune::control_bayes, *13*
tune::control_grid, *13*
tune::control_resamples, *13*
tune::fit_resamples(), *13*
tune::tune_bayes(), *13*
tune::tune_grid(), *13*

workflowsets::workflow_set, *3*
workflowsets::workflow_set(), *3*

y2d, 52
y2d(), *47*
yardstick, *33*
yardstick::accuracy(), *22*
yardstick::average_precision(), *33*
yardstick::brier_class(), *33*
yardstick::classification_cost(), *33*
yardstick::gain_capture(), *33*
yardstick::j_index(), *49*
yardstick::kap(), *22*
yardstick::metric_set, *38*
yardstick::metric_set(), *38*
yardstick::mn_log_loss(), *33*
yardstick::pr_auc(), *33*
yardstick::roc_auc(), *33*, *38*
yardstick::roc_aunp(), *33*
yardstick::roc_aunu(), *33*