Package 'timesift'

November 7, 2026

Title Learn Predictive Representations of Time-Varying Data

Version 0.6.1

Description Fits and compares representations of time-varying data against a prediction target. Given a table of targets and a table of time-stamped series belonging to them, it builds each candidate representation, from the record unreduced through a calendar grain such as a week or a month to a lookback anchored on each target, fits the requested learners on each, scores every candidate on one set of held-out folds, and stacks the out-of-fold predictions into an ensemble. Calendar-aware binning keeps a bin a real week or month rather than a fixed block of hours. Learners, response heads and metrics are registered rather than hard-coded, so adding one is a registration and not a fork of the fitting code. The penalised baseline is an elastic net fitted by cyclic coordinate descent along a warm-started path, following Friedman, Hastie and Tibshirani (2010) <doi:10.18637/jss.v033.i01>. The shipped default is presence-absence with a joint multi-label head, the setting used for species distribution modelling from microclimate loggers, scored by the area under the receiver operating characteristic (ROC) curve with the true skill statistic of Allouche, Tsoar and Kadmon (2006) <doi:10.1111/j.1365-2664.2006.01214.x> beside it.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

URL https://gillescolling.com/timesift/, https://github.com/gcol33/timesift

BugReports https://github.com/gcol33/timesift/issues

Depends R (>= 4.1)

LinkingTo cpp11

Imports grDevices, graphics, rlang, stats, tidyselect (>= 1.2.0), tools, utils

Suggests testthat (>= 3.0.0), torch, glmnet, lme4, lmerTest, emmeans, knitr, rmarkdown, spelling, terra, withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Gilles Colling [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-3070-6066>)

Maintainer Gilles Colling <gilles.colling051@gmail.com>

Repository CRAN

Date/Publication 2026-11-07 16:50:02 UTC

Contents

additive . . . 4
artifacts . . . 5
as.matrix.timesift_matrix . . . 6
average_precision . . . 7
bind_channels . . . 8
boosting . . . 8
boyce_index . . . 10
build_representation . . . 11
calendar_channels . . . 12
combine . . . 13
coverage . . . 14
cv . . . 15
digest_array . . . 16
discriminant . . . 17
elasticnet . . . 19
ensemble . . . 21
ensemble_combine . . . 22
ensemble_fit . . . 23
ensemble_spread . . . 24
ensemble_weights . . . 25
envelope . . . 26
feature_matrix . . . 27
fit_learner . . . 27
fold_map . . . 29
forest . . . 30
grains . . . 31
grain_contrasts . . . 32
grain_ladder . . . 33
grain_matrix . . . 35
hierarchical . . . 38
implied_skill . . . 40
kappa_score . . . 41
learner . . . 43

linear . . . 45
lookback_matrix . . . 46
mars . . . 49
maxent . . . 51
native . . . 53
occlusion . . . 54
ordinal_metric . . . 56
paired_contrast . . . 57
perceptron . . . 59
plot.timesift . . . 61
plot.timesift_ladder . . . 62
plot.timesift_range_change . . . 63
plot.timesift_response_curve . . . 63
plot.timesift_selection . . . 64
positive_weights . . . 65
predict.timesift . . . 66
project . . . 68
pseudo_absences . . . 69
range_change . . . 71
register_learner . . . 72
register_metric . . . 73
register_response . . . 73
register_tuning . . . 74
regression_metric . . . 75
response_curve . . . 76
roc_auc . . . 78
scorable_cells . . . 78
score_predictions . . . 79
select_grain . . . 80
simulate_records . . . 84
table_metric . . . 87
timesift . . . 89
timesift_report . . . 92
timesift_set . . . 94
torch_learners . . . 94
train_control . . . 96
tree . . . 99
tss . . . 100
tss_inflation . . . 101
tune . . . 102

Index **104**

`additive` *Generalised additive model on the flattened representation*

Description

One additive model per response, a smooth function of every bin-by-channel column of the representation, fitted as mgcv's `gam(y ~ s(x1) + s(x2) + ..., method = "GCV.Cp")` fits it and as biomod2 fits `GAM`. Each column enters as a thin plate regression spline of `k` basis functions (Wood 2003): a cubic radial function centred on each of the column's distinct values, reduced to its `k - 2` directions of greatest eigenvalue, together with the linear function, which the penalty on the spline's squared second derivative leaves free. Each smooth sums to zero over the units, beside one intercept.

Usage

```
additive(data = NULL, k = 10L, gamma = 1, max_knots = 2000L, threads = 1L)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`k`	The basis dimension of a column's smooth, the constant included, at least 3.
`gamma`	The criterion's charge per effective degree of freedom.
`max_knots`	The most distinct values of a column the radial functions are centred on.
`threads`	Columns and responses worked at once. The model is the same on any number.

Details

The coefficients maximise the penalised likelihood, by penalised iteratively reweighted least squares, and the smoothing parameters, one per column, minimise the unbiased risk estimator under presence-absence and a count head and the generalised cross-validation score under a squared-error head, by Newton's method with the exact derivatives (Wood 2008). `gamma` multiplies the charge each effective degree of freedom adds to that criterion, as mgcv's `gamma` does; above one it gives smoother fits. Over columns as alike as neighbouring weeks the criterion can have more than one local minimum; the one the search settles in then depends on where it starts, and mgcv, starting from a rule of its own parametrisation, can settle in another.

The defaults are mgcv's, which biomod2 passes unchanged: `k = 10` and a criterion charge of one. Above `max_knots` distinct values, the radial functions are centred on that many of them, drawn as mgcv draws them, so that the basis is the one mgcv fits. A column of fewer than `k` distinct values takes as many basis functions as it holds values, a column of two enters linearly, and a column of one is left out; mgcv refuses the first and the last of those. A column whose linear part the columns before it already span keeps its penalised part and loses the linear one.

The case weights are the response head's, `positive_weights()` under presence-absence, and enter the likelihood as mgcv's prior weights. The model holds at most as many coefficients as there are units, as mgcv's does: one for the intercept and `k - 1` per column. Under the shipped presence-absence head those weights are on, so a default `additive()` is mgcv's specification fitted under them; a head registered without `weights` fits it unweighted.

The basis, the fit and the choice of the smoothing parameters live in the core the Python package calls, pinned against mgcv in the fixtures, so the two languages fit the same model. A response holding one value is predicted its mean and named in `unfitted`; one whose smoothing parameter search stopped short of its tolerance is named in `stopped`.

Value

A `learner()`.

Examples

```
additive()
additive(data = grain("season"), k = 5)
```

`artifacts` *Read and write the artifacts that cross the language boundary*

Description

The representation is built twice, once per language, from one shared core. The response matrix, the fold map and the mask of scorable cells are not: they are built once and read wherever they are needed, because a fold map drawn from a seed in R and one drawn from the same seed in Python are different maps, and aligning the two random streams would be the wrong fix.

Usage

```
write_folds(x, file)

read_folds(file, units = NULL)

write_response(y, file)

read_response(file, units = NULL)

write_cells(cells, file)

read_cells(file)
```

Arguments

`x`	A fold map from `fold_map()`, or any named integer vector.
`file`	Path to write to or read from.
`units`	The units to align to, in the order they are wanted. A unit the file has no row for is an error; a unit the file carries beyond these is dropped. `NULL`, the default, returns the file's own rows in the file's own order.
`y`	A response matrix, as `scorable_cells()` takes one.
`cells`	A mask from `scorable_cells()`.

Details

These six functions are the handover. The Python side carries `write_folds()` and its siblings under the same names; they write the same bytes from the same artifact and read the same files, so a fold map built in either language is usable in the other without the caller knowing the format.

The format is normative and is given in `inst/spec/representation.md`: CSV, UTF-8, a header row, no quoting, LF line endings on every platform, numbers at twelve significant digits, and rows ordered by the identifier under C collation.

Value

The writers return `file`, invisibly. `read_folds()` returns a named integer vector of class `timesift_folds`, `read_response()` a numeric matrix with the units in its row names, and `read_cells()` a `timesift_cells` data frame.

Examples

```
set.seed(1)
y <- matrix(rbinom(120, 1, 0.3), nrow = 20,
            dimnames = list(sprintf("p%02d", 1:20), paste0("sp", 1:6)))
f <- fold_map(y, v = 4)

path <- tempfile(fileext = ".csv")
write_folds(f, path)
back <- read_folds(path, names(f))
identical(names(back), names(f)) && identical(as.integer(back), as.integer(f))
unlink(path)
```

---

as.matrix.timesift_matrix

A representation as a block of predictors

Description

Lays a `[unit, bin, channel]` array out as one row per unit and one column per bin of each channel, which is what a model reading a table of predictors takes. It is what the learners that read a tabular block fit on, so a learner of your own that calls it reads the same columns they do.

Usage

```
## S3 method for class 'timesift_matrix'
as.matrix(x, ...)
```

Arguments

`x`	A `grain_matrix()`, `lookback_matrix()`, `feature_matrix()` or `build_representation()` result.
`...`	Ignored.

Details

Columns are named `channel@bin`, channel by channel and bin by bin within each. A channel that holds the same number in every bin, as a `static` predictor does, is one column named by the channel alone.

Value

A numeric `[unit, predictor]` matrix, the units as row names.

Examples

```
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 60)
d <- data.frame(plot = rep(c("a", "b"), each = length(t)), t = rep(t, 2),
                temp = sin(seq_len(2 * length(t)) / 24))
x <- grain_matrix(d, plot, t, temp, grain = "month", stats = c("min", "max"))
as.matrix(x)
```

`average_precision` *Average precision*

Description

The area under the precision-recall curve, as the step sum over the distinct predictions: at each, the precision of calling every unit at or above it a presence, weighted by the share of presences that cut adds. Units sharing a prediction enter together, so the order they arrived in does not move the score. Its floor is the prevalence rather than one half, which makes it the reading of how well presences are ranked above absences where presences are rare, and it is reported beside `roc_auc()` for that reason.

Usage

```
average_precision(y, p)
```

Arguments

`y`	Observed presence-absence, `0`/`1` or logical.
`p`	Predicted scores for the same units, in the same order. Higher means presence.

Value

One number, or `NA` where the cell defines none.

Examples

```
average_precision(c(0, 0, 1, 1), c(0.1, 0.2, 0.8, 0.9))
average_precision(c(0, 1, 0, 1), c(0.1, 0.2, 0.8, 0.9))
```

bind_channels *Put channels side by side*

Description

Joins representations of the same units and bins into one array, in the order given. It is how a temperature reading, an external product such as snow cover, and the calendar position of each bin reach a model as one input.

Usage

```
bind_channels(...)
```

Arguments

`...`	Two or more representations of shape `[unit, bin, channel]`, agreeing on their units and bins.

Value

One array carrying every channel, in the order the arguments are given and, inside each argument, in its own channel order. The attributes are the first argument's, with `stats` the joined names, except `static` and `position`, which name channels and so name those of every argument.

Examples

```
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 60)
d <- data.frame(plot = rep(c("a", "b"), each = length(t)), t = rep(t, 2),
                temp = rnorm(2 * length(t)))
x <- grain_matrix(d, plot, t, temp, grain = "week")
dimnames(bind_channels(x, calendar_channels(x)))[[3]]
```

boosting *Gradient boosted trees on the flattened representation*

Description

One boosted model per response, over every bin-by-channel column of the representation: a logistic model under a presence-absence head, a squared-error one under a head with a squared-error loss and a Poisson one with a log link under a count head. The score starts at the log-odds of the weighted share of presences, the weighted mean, or the log of the weighted mean count, and each tree is fitted to the loss's gradient at the current score and added to it scaled by `shrinkage`. Each tree is grown on a subsample of the units drawn without replacement, and reads a subsample of the columns.

Usage

```
boosting(
  data = NULL,
  method = c("gbm", "xgboost"),
  trees = NULL,
  depth = NULL,
  shrinkage = NULL,
  min_leaf = NULL,
  subsample = NULL,
  colsample = NULL,
  lambda = NULL,
  gamma = NULL,
  n_inner = NULL,
  preset = c("default", "bigboss"),
  seed = 1L,
  threads = 1L
)
```

Arguments

data	A representation the learner is pinned to, or NULL to run across every representation of the run.
method	"gbm" for first-order trees grown best first, "xgboost" for second-order trees grown level by level.
trees	Trees fitted.
depth	Splits in a tree under "gbm", its depth under "xgboost".
shrinkage	The scale each tree is added to the score at.
min_leaf	Units each side of a split keeps under "gbm", hessian under "xgboost".
subsample	Share of the units each tree is grown on.
colsample	Share of the columns each tree reads.
lambda, gamma	The L2 penalty on a leaf and the least gain of a split under "xgboost"; zero under "gbm".
n_inner	Folds of the inner cross-validation choosing how many trees are kept, or 0 to keep them all.
preset	Whose defaults the settings left NULL take: "default" or "bigboss".
seed	Seed for the subsamples and the inner folds.
threads	Responses fitted at once, or, with one response to fit, the fits of its inner cross-validation. The model is the same on any number.

Details

method picks the trees. Under "gbm" they are Friedman's gradient boosting machine as the gbm package grows it, which is what biomod2 fits as GBM: depth splits grown best first, each the one that most reduces the weighted squared error of the working response with at least min_leaf units on each side, and a leaf that takes one Newton step on the loss. Under "xgboost" they are XGBoost's

exact greedy trees (Chen and Guestrin 2016), which biomod2 fits as `XGBOOST`: grown level by level to `depth`, each split chosen by the second-order gain under the L2 penalty `lambda` with at least `min_leaf` of hessian on each side, pruned where a split gains less than `gamma`, and a leaf the step `-G / (H + lambda)`. Either way `depth` is the order of interaction a tree can hold. With `subsample = 1` the first-order fit is gbm's own to rounding, and the second-order one xgboost's to its single-precision storage; the model is grown by the core the Python package calls, so the two languages fit the same model.

`n_inner` folds, when above zero, choose how many trees are kept: the fit is repeated on each fold's complement, and the number of trees of least held-out deviance, summed over the folds and weighted by how many units each holds, is kept, as gbm's `cv.folds` chooses it. The folds are dealt for each response and stratified on it, as the elastic net's are.

`preset` says whose defaults the settings left `NULL` take. `"default"` is the fitting package's own, which is what biomod2's default option set fits: under gbm 100 trees of one split, `shrinkage = 0.1`, `min_leaf = 10` and `subsample = 0.5`; under xgboost 100 trees of depth 6, `shrinkage = 0.3`, `min_leaf = 1`, `lambda = 1` and every unit and column. `"bigboss"` is biomod2's tuned option set: under gbm 2500 trees of seven splits, `shrinkage = 0.001`, `min_leaf = 5`, `subsample = 0.5` and three inner folds; under xgboost four trees of depth 2 at `shrinkage = 1`. A setting given explicitly beats either.

The case weights are the response head's, `positive_weights()` under presence-absence, and weigh the gradient and every sum a tree is grown on; `min_leaf` counts units under gbm, as `n.minobsinnode` does. Under the shipped presence-absence head those weights are on, so a default `boosting()` is gbm's specification fitted under them; a head registered without `weights` fits it unweighted.

Value

A `learner()`.

Examples

```
boosting()
boosting(preset = "bigboss")
boosting(method = "xgboost", depth = 3L)
```

`boyce_index` *The continuous Boyce index*

Description

How consistently the predictions rise with the density of presences, without a cut and without absences (Hirzel et al. 2006). A window of `width` times the range of the predictions slides over that range in `resolution + 1` equal steps. In each, the share of presences falling inside divided by the share of all units falling inside is the predicted-to-expected ratio, undefined where no unit falls inside, and the index is the Spearman correlation of that ratio with the window's midpoint, between -1 and 1. Predictions that rank no better than chance read near 0.

Usage

```
boyce_index(y, p, resolution = 100L, width = 0.1)
```

Arguments

y	Observed presence-absence, 0/1 or logical.
p	Predicted scores for the same units, in the same order. Higher means presence.
resolution	Number of window steps, 100 by default.
width	Window width as a share of the range of p, 0.1 by default.

Details

The units of p are the background, which is how biomod2 reads BOYCE on a fit. The windows are closed at both ends and the ratio is not thinned of repeated values.

Value

One number, or NA where the cell defines none, where the predictions are all equal, or where the ratio is defined in fewer than three windows or takes one value.

Examples

```
boyce_index(c(0, 0, 0, 1, 1, 1, 0, 1), c(0.10, 0.20, 0.35, 0.40, 0.60, 0.90, 0.55, 0.70))
```

---

build_representation *Build one representation for a set of targets*

Description

Turns a representation and the two tables into the [target, bin, channel] array a learner is fitted on. It is the one place the fitting layer builds an array, so timesift() and predict.timesift() reach a record the same way and a candidate refitted on new targets is built from the settings its own arm was.

Usage

```
build_representation(rep, series, targets, spec)
```

Arguments

rep	A representation from native(), grain(), multigrain() or lookback().
series	The long table of readings, or NULL where the targets carry the whole predictor block.
targets	The table of prediction targets.
spec	The resolved settings timesift() carries: the identifier, time, value, anchor and static columns it settled on.

Details

The rows are the targets, in the order the fitting layer keeps them: sorted by identifier where one target row belongs to each unit, and in the targets' own order where `target_time` anchors them. Columns named in `static` are appended as channels holding one value per target, constant across the bins.

Value

A `timesift_matrix`.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 90)
units <- sprintf("p%02d", 1:30)
warmth <- rnorm(30)
logger <- data.frame(
  plot = rep(units, each = length(t)), datetime = rep(t, 30),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300), numeric(length(t)))))
plots <- data.frame(plot = units, sp_a = rbinom(30, 1, plogis(2 * warmth)))
fit <- timesift(plots, logger, y = sp_a, id = plot, time = datetime,
                sift = grains("month"), resampling = cv(v = 3L), ensemble = FALSE,
                verbose = FALSE)
x <- build_representation(grain("week"), logger, plots, fit$spec)
dim(x)
```

`calendar_channels` *Where in the year, or the day, each bin sits*

Description

An encoder that ends in global pooling discards when a thermal event happened, so the position of a bin in the year has to be given to it as input if it is to be used at all. These channels carry that position as the sine and cosine of the bin's fractional place in the year, which is continuous across the turn of the year where the fraction itself is not. On a record read finer than a day, the same pair for the place in the day carries where a reading sits in the daily cycle.

Usage

```
calendar_channels(x, cycles = "year")
```

Arguments

`x`	A `grain_matrix()` result. A `lookback_matrix()` result has no place in the calendar and is refused.
`cycles`	Which cycles to place each bin in, `"year"`, `"day"` or both, in the order given. The day cycle reads bins that sit less than a day apart; at a day or coarser every bin would sit at the same place in the day, and it is refused.

Details

They are the time index of each bin, not a summary of the readings, so adding them introduces no hand-built thermal feature: whatever a model does with them it could have done with a calendar.

The position is read at the midpoint of the record each bin holds, in UTC, so a bin the record only partly covers sits at the phase it was actually measured over. A site's longitude and a zone's offset move the day's phase by the same amount for every bin, which a model absorbs, where a local clock's summer time would move it twice a year. `inst/spec/representation.md` is the normative description.

Value

An array of the same units and bins with two channels per cycle, `year_sin` and `year_cos`, `day_sin` and `day_cos`, identical across units. Combine it with the readings using `bind_channels()`. The channels are recorded as positions in the `position` attribute, and the encoders of torch_learners read them at their own amplitude rather than standardise them.

Examples

```
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 400)
d <- data.frame(plot = "a", t = t, temp = sin(seq_along(t) / 500))
x <- grain_matrix(d, plot, t, temp, grain = "month")
round(calendar_channels(x)[1, 1:4, ], 3)

hourly <- grain_matrix(d, plot, t, temp, grain = "native")
round(calendar_channels(hourly, cycles = c("year", "day"))[1, 1:4, ], 3)
```

`combine` *Combine learners, or representations, into a set*

Description

`c()` on learners is the set of them, and on representations the set of those. A set handed to `c()` again splices, so a set can be added to rather than rewritten, which is what `list()` cannot do: `c(base, cnn())` where `base` is already a set.

Usage

```
## S3 method for class 'timesift_learner'
c(...)

## S3 method for class 'timesift_models'
c(...)

## S3 method for class 'timesift_representation'
c(...)

## S3 method for class 'timesift_sift'
c(...)
```

Arguments

...	Learners, representations, or sets of either.

Details

`learners` and `sift` take either form. A length-one string is the name of a registered learner.

Value

A `timesift_models` for learners and a `timesift_sift` for representations.

Examples

```
base <- c(elasticnet(), forest())
base
c(base, linear())

c(grains("day", "week"), lookback("30 days"))
```

---

coverage *Which units reach which bins*

Description

A representation needs every unit in every bin, and `grain_matrix()` refuses a record where one is missing rather than pad it. This is the same binning laid out so the gaps can be read: how many readings each unit has in each bin, over every bin the calendar tiles the record with from the first bin any unit touches to the last. A logger that started late, stopped early or lost a month is a row with zeros in it; a bin the whole record skips is a column of zeros.

Usage

```
coverage(data, id, time, grain = "day", year_start = "01-01")
```

Arguments

`data`	A data frame of readings in long form, one row per reading.
`id`	Column identifying the unit carrying the sensor. A bare column name or a string.
`time`	Column of reading instants, `POSIXct`. A bare column name or a string.
`grain`	One of `"native"`, `"halfday"`, `"day"`, `"week"`, `"month"`, `"season"`, `"year"`. The four coarse grains follow the calendar, so a bin is a real week or month rather than a fixed block of hours. Naming several grains returns one representation per grain, a `timesift_set()`. A function is called on the reading instants and must return the `POSIXct` start of each reading's bin, which is how a calendar the package does not carry, such as astronomical seasons, is binned.
`year_start`	`"MM-DD"` boundary of the year, used by `"season"` and `"year"`. Defaults to the calendar year, `"01-01"`; a hydrological year starting in September is `"09-01"`.

Details

What to do about a gap is the analyst's decision, and this is the table it is made on: drop the units that do not span the record, cut the record to the span every unit covers, or move to a grain the gap does not reach. Nothing here fills a cell.

Value

An integer matrix of reading counts, one row per unit and one column per bin, of class `timesift_coverage`, with the units and the ISO-8601 bin starts as dimnames and the `grain` and the `bin_start` instants as attributes.

Examples

```
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 40)
d <- data.frame(plot = rep(c("a", "b"), each = length(t)), t = rep(t, 2),
                temp = rnorm(2 * length(t)))
# Unit b loses the calendar week beginning Monday 6 September.
lost <- d$plot == "b" & d$t >= as.POSIXct("2021-09-06", tz = "UTC") &
  d$t < as.POSIXct("2021-09-13", tz = "UTC")
coverage(d[!lost, ], plot, t, grain = "week")
```

---

cv *How the folds are drawn*

Description

The resampling `timesift()` scores on. `cv()` deals units into folds balanced on the stratifying value; `grouped_cv()` deals whole groups, keeping every target sharing a group value on one side of each split. `block_cv()` and `env_cv()` hold out a block of units whole: a region of space for the first, a region of predictor space for the second.

Usage

```
cv(v = 10L, seed = 1L, strata = 5L, by = NULL, repeats = 1L)

block_cv(by, v = 4L)

env_cv(by, v = 4L)

grouped_cv(group, v = 10L, seed = 1L, repeats = 1L)
```

Arguments

`v`	Number of folds.
`seed`	Random seed, fixed so the map is reproducible.
`strata`	Number of strata, or 1 for no stratification.

by	What cv() stratifies on instead of the richness of the response: the name of a numeric column of targets, or a vector with one value per target. For block_cv() and env_cv(), the columns the blocks are cut on: names of columns of targets, or a numeric matrix with one row per target.
repeats	Number of times the split is drawn, each with its own seed (seed, seed + 1, and so on). In timesift() every repeat is a full run on its own fold map: a score stays one fit's held-out score, the report averages a response's scores over folds and repeats, and the combiner is fitted on the out-of-fold predictions of all the repeats together. block_cv() and env_cv() draw nothing, so repeat nothing.
group	The grouping: the name of a column of targets, or a vector with one value per target.

Details

The blocks are cut by halving. The units are split at the median of the column with the widest range (the first on a tie), the lower share going to the left and the units tied on that column ordered as they arrived, and each part is cut again until there are v of them, so a block holds as many units as another to within one and a map of v blocks has v folds. Nothing is drawn, so both languages return the same map. env_cv() centres and scales every column first, which a coordinate system does not need. Under the nested selection the inner folds keep blocks whole as well, so n_inner is at most v - 1.

resampling also accepts a fold vector or a fold_map() result directly, which is how a split the package has no constructor for – a spatial block, a season held out whole – reaches the same fitting path.

Value

A timesift_resampling.

Examples

```
cv(v = 5L)
cv(v = 5L, by = "elevation")
grouped_cv("site")
block_cv(c("x", "y"), v = 4L)
env_cv(c("elevation", "slope"), v = 5L)
```

---

digest_array *The cross-language digest of a representation*

Description

The MD5 the contract in inst/spec/representation.md defines, and what the fixtures in inst/spec/fixtures/ are checked against. timesift.digest_array() on the Python side is the same function: the same array gives the same string in either language, which is what makes a digest a statement about the representation rather than about the machine.

Usage

```
digest_array(x)
```

Arguments

x	A grain_matrix() result, or any numeric array.

Details

Values are traversed in the array's own order, unit fastest, then bin, then channel; each is formatted to twelve decimal places, joined with a line feed and terminated with one, and the UTF-8 bytes of that are hashed. The line ending is a line feed on every platform. It is written explicitly because `writeLines()` emits CRLF on Windows, which would make a digest depend on the machine that produced it.

The array has to be finite. `%.12f` writes an infinity as `Inf` here and as `inf` in Python, and R tells a missing value apart from a not-a-number where Python has one spelling for both, so a digest over such an array would say something about the language rather than about the representation. A representation never holds one: a reading that is not a finite number is refused where the record is read.

Value

The digest, a single string of 32 hexadecimal characters.

Examples

```
digest_array(array(c(1, -0.5), dim = c(2L, 1L, 1L)))
```

---

discriminant *Flexible discriminant analysis on the flattened representation*

Description

One discriminant per response, over every bin-by-channel column of the representation, fitted as mda's `fda(method = mars)` fits it and as biomod2 fits `FDA`. Optimal scoring gives presence and absence one score each and regresses the scored response on a MARS basis of the columns; the fitted score is the one canonical variate, and a unit's prediction is its posterior probability of presence under two normal classes around the class centroids on that variate, with the classes' shares among the fitting units as priors.

Usage

```
discriminant(
  data = NULL,
  degree = 1L,
  penalty = NULL,
  max_terms = NULL,
  min_gain = 0.001,
  prune = TRUE,
  calibrate = TRUE,
  threads = 1L
)
```

Arguments

data	A representation the learner is pinned to, or NULL to run across every representation of the run.
degree	The most hinges a term of the basis multiplies.
penalty	The generalised cross-validation's charge per term beyond its own degree of freedom, taken at half. NULL is mda's, 2 at degree one and 3 above.
max_terms	The most terms the forward pass reaches, the intercept included. NULL is max(21, 2 p + 1) for p columns; an even number is taken one lower.
min_gain	The least share of the residuals a forward step is kept for.
prune	Whether the pruning pass runs. Without it every term of the forward pass is kept.
calibrate	Whether the posterior is recalibrated by a probit regression, as biomod2 does.
threads	Responses fitted at once, or, with one response to fit, columns searched at once. The model is the same on any number.

Details

The basis is mda's own MARS, which biomod2 reaches through fda() and which differs from earth's that mars() reproduces. Each forward step adds a column linearly or a pair of hinges on it, max(0, x - t) and max(0, t - x), choosing by Friedman's running updates, and the pass stops when a step lowers the residuals by less than min_gain of them, when they fall to min_gain of the null model's, when the generalised cross-validation passes ten times the null model's, or at max_terms terms. The pruning pass drops the term of least t statistic, one at a time, and keeps the subset of least generalised cross-validation, which counts each term beyond the intercept as 1 + penalty / 2 degrees of freedom. The defaults are mda's, which biomod2 passes unchanged under its default option set and under "bigboss": degree one, penalty = 2, min_gain = 0.001 and max_terms = max(21, 2 p + 1) for p columns.

The case weights are the response head's, positive_weights() under presence-absence. They set the classes' scores and the variate, as fda() reads them, and not the basis: mda's forward pass sets them to one. Under the shipped presence-absence head those weights are on, so a default discriminant() is mda's specification fitted under them; a head registered without weights fits it unweighted.

biomod2 always recalibrates an FDA posterior by a probit regression of the response on it, under the case weights, and predicts through that regression. calibrate = TRUE does the same, on the units the discriminant was fitted on; FALSE predicts the posterior itself. biomod2 rounds the posterior to three decimals before recalibrating it, which is not reproduced.

The forward pass, the pruning, the scoring and the recalibration live in the core the Python package calls, pinned against mda and R's glm() in the fixtures, so the two languages keep the same terms and predict the same probabilities. A response holding one value is predicted its mean, and so is one whose scored response the basis does not reach; both are named in unfitted. The learner needs a presence-absence response, and a head whose loss is not the binary cross-entropy is refused.

Value

A learner().

Examples

```
discriminant()
discriminant(data = grain("season"), calibrate = FALSE)
```

---

elasticnet *Penalised regression on the flattened representation*

Description

One elastic net per variable, over every bin-by-channel column of the representation and, by default, their squares. There is no discrete selection step: the penalty path uses every column and shrinks, and the penalty itself is chosen by an inner cross-validation on the fitting units, so nothing about the model is decided outside the fold it is fitted in.

Usage

```
elasticnet(
  data = NULL,
  alpha = 0.5,
  n_inner = 10L,
  squares = TRUE,
  s = "lambda.1se",
  n_lambda = 100L,
  tol = 1e-08,
  threads = 1L,
  seed = 1L
)
```

Arguments

data	A representation the learner is pinned to, or NULL to run across every representation of the run.
alpha	Elastic-net mixing, 1 lasso and 0 ridge.
n_inner	Folds of the inner cross-validation that chooses the penalty, ten as in cv.glmnet().
squares	Add the square of every column, giving the same quadratic capacity a second-order polynomial term would.
s	Which penalty of the inner path to predict at: "lambda.min", "lambda.1se", or a penalty of its own, which is interpolated between the two points of the path around it. "lambda.1se", the largest penalty within one standard error of the least held-out deviance, is glmnet's default.
n_lambda	Points of the penalty path.
tol	Where the coordinate descent stops, read off the largest coefficient move of a pass. The default leaves the fit as close to the optimum as glmnet's own default does; a looser one is faster and a tighter one costs time roughly in proportion.
threads	How many responses are fitted at once, or, with one response to fit, how many fits of its inner cross-validation: the path on every fitting unit and the path of each inner fold are one independent fit each, so n_inner + 1 threads is as many as a lone response can use. The default is serial, because a package does not take a machine's cores without being asked. What comes back does not depend on it.
seed	Seed for the inner cross-validation's fold draw, which is random and would otherwise make the fit irreproducible.

Details

The family is the response head's: a binary cross-entropy loss fits a logistic model, a squared-error loss a linear one and a Poisson-deviance loss a log-linear one, so the learner is the same under a presence-absence head, a continuous one and a count one. So are the case weights: the head's weights, positive_weights() for presence-absence, are what every learner that ships fits under.

The inner folds are dealt for each response and stratified on it, so a rare outcome is spread over them as evenly as its count allows. A presence-absence response whose inner training sets cannot each hold two of each outcome, the fewest a logistic path is fitted to, has too few of one outcome to choose a penalty on. It is predicted its share among the fitting units, as a response holding one outcome is, and the fit names every such response in unfitted.

A reweighted fit that does not settle at a penalty ends its path there and keeps the points before it, which is what glmnet does with the same event, and the penalty is chosen over the points fitted. It happens where a rare outcome is nearly separable at the small end of the path. The fit names every response whose path on the fitting units, or on any inner fold, ended that way in stopped.

This is the aggregate-feature side of the comparison the package was built for, and it is the fair opponent for a network: a per-fold discrete selector pays selection variance a network never pays, so beating that one is not a matched result.

The path is fitted by the same core the Python package calls, so the two return the same coefficients for the same input. Its conventions are glmnet's, which is what the arm is measured against: weights

normalised to sum to one, columns centred and scaled by their weighted mean and weighted standard deviation, a hundred penalties down from the smallest that leaves every coefficient at zero, and the held-out deviance read fold by fold.

Value

A `learner()`.

Examples

```
elasticnet(alpha = 0.5)
```

`ensemble` *How the candidates are combined*

Description

Every candidate emits an out-of-fold prediction for every scorable cell over the same folds, so the combination is arithmetic on those predictions and nothing else. `ensemble()` says which arithmetic.

Usage

```
ensemble(
  method = c("stack", "mean", "median", "weighted", "committee"),
  scope = c("all", "learners", "representations"),
  metric = NULL,
  response = NULL,
  min_score = NULL,
  decay = NULL,
  rule = NULL
)
```

Arguments

`method`	How the members are combined.
`scope`	Which candidates are eligible.
`metric`	Name of the registered metric the eligibility, `min_score` and the `"weighted"` weights are read by, or `NULL` for the score the run already carries.
`response`	Name of the registered response head whose loss `"stack"` minimises, or `NULL` for the head the run was fitted under. Naming a head the run does not fit toward is an error rather than an override, and `ensemble_fit()` called on its own reads `NULL` as `"presence_absence"`.
`min_score`	`NULL`, or the mean score a candidate needs to be eligible.
`decay`	For `"weighted"`, `"proportional"` (the default) or a number of at least one, the ratio of one rank's weight to the next one's.
`rule`	For `"committee"`, the rule of `decision_threshold()` each member's cut is learned by; `"youden"` by default.

Details

"stack" fits non-negative weights summing to one on the out-of-fold predictions alone, never on in-sample ones, minimising the response head's loss over the scorable cells: binomial deviance for presence-absence. One weight vector covers every response, because per-response weights would be fitted on the handful of cells a rare response has. "mean" and "median" combine without fitting anything. "weighted" takes each candidate's own mean score, keeps its non-negative part and rescales those to sum to one, so a candidate scoring at or below zero is left out and the rest are weighted by how well they scored; where no candidate scores above zero the scores order nothing worth weighting by, and every candidate weighs the same.

decay changes how "weighted" turns scores into weights, as biomod2's EMwmean.decay does. Given a number d, the candidates scoring above zero are ranked from the best down, the one ranked r of K takes d^(K - r + 1), candidates on the same score share the mean of their ranks' weights, and the weights are rescaled to sum to one. Each rank down weighs 1 / d of the one above it, whatever the gap in score between them.

"committee" is biomod2's committee averaging. Each member cuts each response at its own threshold, learned by decision_threshold() under rule from that member's out-of-fold predictions of every target, and the combined prediction is the share of members voting presence. A member whose predictions of a response give no cut does not vote on it. Under timesift() the thresholds of each outer fold are learned from the inner out-of-fold predictions of its training targets, as a stack's weights are, so the estimate never reads a cut chosen on the targets it scores.

min_score is biomod2's metric.select.thresh: a candidate whose mean score is below it is not eligible, and the filter is applied before scope picks among what is left.

scope says which candidates are eligible. "all" is every candidate. "learners" keeps the several learners that read the representation of the best-scoring candidate, and "representations" keeps the one learner of the best-scoring candidate across the representations it ran on; both are read off the same mean scores the report shows.

Value

A timesift_ensemble.

Examples

```
ensemble()
ensemble("weighted", scope = "learners")
ensemble("weighted", decay = 1.6, min_score = 0.4)
ensemble("committee", rule = "kappa")
```

---

ensemble_combine *Combine one prediction per member into one prediction*

Description

Combine one prediction per member into one prediction

Usage

```
ensemble_combine(stack, preds)
```

Arguments

stack	A ensemble_fit() result.
preds	Named list of [target, response] matrices, one per member of the stack.

Value

One [target, response] matrix.

Examples

```
set.seed(1)
y <- matrix(rbinom(200, 1, 0.4), nrow = 50,
            dimnames = list(sprintf("p%02d", 1:50), paste0("sp", 1:4)))
folds <- fold_map(y, v = 5)
truth <- matrix(runif(200), nrow = 50, dimnames = dimnames(y))
oof <- list(good = 0.8 * y + 0.2 * truth, noise = truth)
st <- ensemble_fit(oof, y, scorable_cells(y, folds), folds)
dim(ensemble_combine(st, oof))
```

---

ensemble_fit *Fit the combiner on the out-of-fold predictions*

Description

The combiner sees the out-of-fold predictions, the response, the mask of scorable cells and the fold map, and never a model, so it cannot read anything a candidate fitted in-sample.

Usage

```
ensemble_fit(oof, y, cells, folds, spec = ensemble(), scores = NULL)
```

Arguments

oof	Named list of [target, response] matrices, one per candidate.
y	The response matrix.
cells	The scorable-cell mask from scorable_cells().
folds	The fold map.
spec	An ensemble() specification.
scores	The per-cell scores of the run, a data frame carrying candidate, variable, fold, score and scorable. Read only where spec names no metric of its own.

Details

The weights are fitted to the response on those predictions, so the combined prediction scored against the same response is scored on the data its weights were fitted to, and that score is optimistic. `timesift()` evaluates the stack the other way: each outer fold's weights are fitted on inner out-of-fold predictions of its training targets and applied to the outer test fold.

Value

A `timesift_stack`: `method`, the named `weights`, and what they were fitted on.

Examples

```
set.seed(1)
y <- matrix(rbinom(200, 1, 0.4), nrow = 50,
            dimnames = list(sprintf("p%02d", 1:50), paste0("sp", 1:4)))
folds <- fold_map(y, v = 5)
truth <- matrix(runif(200), nrow = 50, dimnames = dimnames(y))
oof <- list(good = 0.8 * y + 0.2 * truth, noise = truth)
ensemble_fit(oof, y, scorable_cells(y, folds), folds)
```

---

ensemble_spread *How far the members of an ensemble disagree*

Description

biomod2's `EMcv` and `EMci`, read on the members' predictions under the weights the stack carries. For each target and response: the weighted mean `m` of the members' predictions; their weighted standard deviation `s`, the square root of `sum(w * (p - m)^2) / (1 - sum(w^2))`, which is the sample standard deviation when the weights are equal; the coefficient of variation `s / m`; and the interval `m -+ qt(1 - alpha / 2, n - 1) * s * sqrt(sum(w^2))`, `n` the number of members carrying weight, which is the t interval of a mean of `n` members when the weights are equal. Under a response head whose predictions are probabilities, the interval is held inside zero and one. An uncertainty map is this on one target per map cell.

Usage

```
ensemble_spread(stack, preds, alpha = 0.05)
```

Arguments

`stack`	A `ensemble_fit()` result.
`preds`	Named list of `[target, response]` matrices, one per member of the stack.
`alpha`	One minus the interval's coverage.

Details

A committee's and a median's members are read at equal weight, and a committee's spread is that of the members' predictions rather than of their votes.

Value

A `[target, response, statistic]` array, the statistics being `mean`, `sd`, `cv`, `lower` and `upper`. `sd`, `cv` and the interval are `NA` where fewer than two members carry weight.

Examples

```
set.seed(1)
y <- matrix(rbinom(200, 1, 0.4), nrow = 50,
            dimnames = list(sprintf("p%02d", 1:50), paste0("sp", 1:4)))
folds <- fold_map(y, v = 5)
truth <- matrix(runif(200), nrow = 50, dimnames = dimnames(y))
oof <- list(good = 0.8 * y + 0.2 * truth, fair = 0.6 * y + 0.4 * truth, noise = truth)
st <- ensemble_fit(oof, y, scorable_cells(y, folds), folds, ensemble("mean"))
ensemble_spread(st, oof)[1:3, "sp1", ]
```

`ensemble_weights` *The weights the combiner fitted*

Description

The weights the combiner fitted

Usage

```
ensemble_weights(fit)
```

Arguments

`fit`	A `timesift` result, or the stack itself.

Value

A named numeric vector, or `NULL` where the run fitted no combiner.

Examples

```
set.seed(1)
y <- matrix(rbinom(200, 1, 0.4), nrow = 50,
            dimnames = list(sprintf("p%02d", 1:50), paste0("sp", 1:4)))
folds <- fold_map(y, v = 5)
truth <- matrix(runif(200), nrow = 50, dimnames = dimnames(y))
oof <- list(good = 0.8 * y + 0.2 * truth, noise = truth)
ensemble_weights(ensemble_fit(oof, y, scorable_cells(y, folds), folds))
```

`envelope` *biomod2's surface range envelope on the flattened representation*

Description

One envelope per response, over every bin-by-channel column of the representation: for each column, the `quantile` and `1 - quantile` quantiles of its readings over the units present, and a unit predicted present where every column lies between its two, the ends included. It is biomod2's `SRE`, and with the same quantile it draws the envelope `bm_SRE()` draws; the quantile is R's default, type 7.

Usage

```
envelope(data = NULL, quantile = 0.025)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`quantile`	The share of the presences left outside at each end of every column, in `[0, 0.5]`. biomod2's default is `0.025`.

Details

The prediction is zero or one. It enters an ensemble as that, and the combiner weighs it by its held-out loss like any other candidate; a threshold read off it has two values to choose from, so its TSS is the one its own zeros and ones give.

An envelope reads the presences and nothing else. The absences do not move it, and neither do the response head's case weights, which it takes no notice of.

Every column has to agree for a unit to be inside, so the more columns a representation has the fewer units any envelope holds: at the default quantile each column shuts out about one presence in twenty, and a weekly representation of three years has 157 columns per channel. A coarse grain is what an envelope is meant for, and pinning it there with `data = grain("season")` keeps it there while the rest of the run reads finer ones.

A response with no presence, or with nothing else, is predicted its share among the fitting units, and the fit names it in `unfitted`. The learner needs a presence-absence response, and a head whose loss is not the binary cross-entropy is refused.

Value

A `learner()`.

Examples

```
envelope()
envelope(data = grain("season"), quantile = 0.05)
```

feature_matrix *Bring an already-reduced feature table into a ladder*

Description

A representation the package did not build, such as a published set of hand-aggregated climate summaries, enters here. It becomes a one-channel `[unit, feature, 1]` array, which is what a learner reads, so a feature table and a temporal grain can be arms of the same `grain_ladder()` and be scored on the same cells by the same rule.

Usage

```
feature_matrix(m, label = "features")
```

Arguments

`m`	A matrix or data frame of units by features, with unit identifiers in the row names or in a leading character or factor column.
`label`	The name the arm is reported under.

Details

It carries no time axis, because it has none: the reduction already happened, elsewhere, and what reaches the model is a list of numbers per unit. That is the whole point of comparing against it.

Value

A `timesift_matrix` of shape `[unit, feature, 1]`.

Examples

```
m <- matrix(rnorm(30), nrow = 10,
            dimnames = list(sprintf("p%02d", 1:10), paste0("bio", 1:3)))
feature_matrix(m)
```

fit_learner *Fit one learner at one grain*

Description

Fit one learner at one grain

Usage

```
fit_learner(
  learner,
  x,
  y,
  response = "presence_absence",
  control = NULL,
  group = NULL,
  ...
)

## S3 method for class 'timesift_fit'
predict(object, newdata, ...)
```

Arguments

learner	A learner(), or the name of a registered one.
x	A grain_matrix() result.
y	The response for the same units.
response	Name of the registered response head. "presence_absence" ships.
control	The run's train_control(). The learner's own control overrides it on the settings that control names, and a setting given in ... overrides both.
group	One value per unit of x naming the group it belongs to, or NULL. A learner whose fit declares group draws its inner splits by it. Under grouped_cv() the run hands every fit the grouping its fold map carries.
...	Passed to the learner's fit.
object	A timesift_fit.
newdata	A representation of the same channels for the units to predict.

Value

A timesift_fit, which stats::predict() takes a new representation.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 120)
units <- sprintf("p%02d", 1:40)
d <- data.frame(plot = rep(units, each = length(t)), t = rep(t, length(units)),
                temp = as.numeric(replicate(length(units), rnorm(length(t)))))
x <- grain_matrix(d, plot, t, temp, grain = "month")
y <- matrix(rbinom(80, 1, 0.4), nrow = 40, dimnames = list(units, c("sp1", "sp2")))
fit <- fit_learner(elasticnet(), x, y)
dim(stats::predict(fit, x))
```

`fold_map` *Assign units to cross-validation folds*

Description

One fold map, built once and read by everything that scores. Every learner in a ladder is then fitted and scored on identical splits, which is what makes the comparison between them paired rather than a comparison of two clouds of numbers.

Usage

```
fold_map(y, v = 10L, seed = 1L, strata = 5L, by = NULL, group = NULL)
```

Arguments

`y`	The response: a matrix or data frame of units by variables, with unit identifiers in the row names or in a leading character or factor column.
`v`	Number of folds.
`seed`	Random seed, fixed so the map is reproducible.
`strata`	Number of strata, or `1` for no stratification.
`by`	A numeric vector of length `nrow(y)` to stratify on instead of richness.
`group`	A vector of length `nrow(y)`. Rows sharing a value land in one fold, which is what repeated targets on one unit through time need: a unit split across folds would put the same unit on both sides of a split and the held-out score would be read on rows the model had already seen a near-copy of. `NULL`, the default, is every row on its own.

Details

Units are held out singly. Where the input a model reads is measured at the unit itself, as a logger in each plot is, a held-out unit brings its own measured input with it and nothing of its neighbours' reaches the model.

Folds are balanced within strata: units are grouped into `strata` equal-count groups of the stratifying value, shuffled inside each group, and dealt round-robin by one counter that runs on from each stratum into the next, so each fold carries the same mix and the folds are equal in size to within one unit whatever `v` is, up to one unit per fold. With a multi-variable response the default stratifies on richness, the number of variables present at a unit, because one fold map has to serve every variable at once and cannot be stratified on any single one of them.

Value

An integer vector of fold numbers named by unit, of class `timesift_folds`. Any named integer vector of the same shape is accepted wherever this is.

Examples

```
y <- matrix(rbinom(300, 1, 0.3), nrow = 60,
            dimnames = list(sprintf("p%02d", 1:60), paste0("sp", 1:5)))
f <- fold_map(y, v = 5)
table(f)
```

`forest` *Random forest on the flattened representation*

Description

One forest per response, over every bin-by-channel column of the representation: a probability forest under a presence-absence head and a regression forest under a head with a squared-error loss or a count head, a count being cut on its variance and a leaf reporting its mean. Trees split on one column at a time and pay nothing for columns that carry nothing, so a forest reads a wide tabular representation without a penalty path and without a selection step, and it finds an interaction between two bins that a linear model would need the product term for.

Usage

```
forest(
  data = NULL,
  trees = NULL,
  mtry = NULL,
  min_node = NULL,
  balance = FALSE,
  preset = c("default", "bigboss"),
  seed = 1L,
  threads = 1L
)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`trees`	Trees in the forest.
`mtry`	Columns drawn for each node, the best of which it is split on.
`min_node`	Units each side of a split keeps.
`balance`	Whether each tree draws as many units from each class as the smaller holds.
`preset`	Whose defaults the settings left `NULL` take: `"default"` or `"bigboss"`.
`seed`	Seed for the bootstrap draws and the column draws.
`threads`	Responses' forests grown at once, or, with one response to fit, its trees. The forest is the same on any number.

Details

Each tree is grown on a bootstrap draw of the units, as many draws as there are units, and each node is split on the best of `mtry` columns drawn for it, by the Gini index or the sum of squares `tree()` splits by. A tree is grown out: a node is split while it holds at least twice `min_node` units and its responses differ. A leaf reports the share of presences among the draws it holds, or their mean, and the forest the mean over its trees. The forest is grown by the core the Python package calls, and every draw comes from one generator seeded per tree, so the two languages grow the same forest on any number of threads.

`balance = TRUE` is the down-sampled forest biomod2 fits as `RFd`: each tree draws as many units from each class as the smaller class holds, so a rare response's presences are half of every tree's draw.

`preset` says whose defaults the settings left `NULL` take. `"default"` is randomForest's own, which is what biomod2's default option set fits: 500 trees, `mtry` the square root of the column count under presence-absence and a third of it under a squared-error loss, and `min_node` 1 and 5 under the two. `"bigboss"` is biomod2's tuned option set: 500 trees, `mtry = 2` and `min_node = 5`. A setting given explicitly beats either, and an `mtry` above the column count is the column count.

The case weights are the response head's, `positive_weights()` under presence-absence, and weight the bootstrap draw: a unit is drawn in proportion to its weight, and within its class under `balance`. Under the shipped presence-absence head those weights are on, so a default `forest()` is randomForest's specification fitted under them; a head registered without `weights` fits it unweighted.

Value

A `learner()`.

Examples

```
forest(trees = 200L)
forest(preset = "bigboss", balance = TRUE)
```

---

grains *Several representations to run the same learners across*

Description

The set a learner without a `data =` of its own is fitted at every member of. `grains()` names calendar grains, `lookbacks()` names lookback spans, and either takes its arguments as separate names or as one vector.

Usage

```
grains(..., stats = "mean", year_start = "01-01")

lookbacks(..., lag = "0 days", bins = 1L, stats = "mean")

as_sift(x)
```

Arguments

...	Grain names for grains(), lookback spans for lookbacks(), given as separate arguments or as one vector.
stats	Statistics computed per bin, one channel each, in the order given. See grain_matrix() for the seven and for what separates an extreme reading from an extreme day.
year_start	"MM-DD" boundary of the year, used by "season" and "year".
lag	The gap between a target's instant and the end of its lookback.
bins	Sub-bins the lookback is cut into, oldest first. One gives a block of features, several give a sequence.
x	A named list of representations, a single representation, or a character vector of grain names.

Details

grains("auto") is the set of named grains the record gives at least two bins, in the order native, halfday, day, week, month, season, year, leaving out any grain the requested statistics are not defined at. There is no cap on it: over three years of hourly readings native is 26304 bins, and a caller who does not want that names the grains instead.

Value

A timesift_sift: a named list of representations, labelled by each one's own label where the list was not named.

Examples

```
grains("day", "week", "month")
grains(c("month", "year"), stats = c("cold_day", "mean", "warm_day"))
lookbacks("30 days", "90 days", bins = 3L)
```

---

grain_contrasts *Compare every grain against a learner's best one*

Description

A mixed model on the per-cell scores of one learner, score ~ grain + (1 | variable) + (1 | fold), fitted by restricted maximum likelihood. Every grain is then compared against the reference by Dunnett's many-to-one procedure, which corrects for the comparisons made without correcting for pairs nobody asked about.

Usage

```
grain_contrasts(ladder, learner = NULL, reference = NULL, adjust = "mvt")
```

Arguments

ladder	A grain_ladder() result.
learner	Which learner's grid to fit. The only one in the ladder by default.
reference	The grain every other is compared against. The learner's best by default.
adjust	Multiplicity adjustment passed to emmeans::contrast().

Details

The design is balanced across variables, folds and grains, so each grain is compared within a variable and within a fold and the variation between variables cancels from the comparison. That is what lets a difference of 0.015 hold up where absolute skill ranges across variables by ten times as much.

Taking the best-observed grain as the reference is a choice that favours the reference, so read this beside the paired differences paired_contrast() gives, which single out no grain.

Value

A data frame of one row per grain: the estimated marginal mean difference from the reference, its interval, and the adjusted p-value. Needs lme4, lmerTest and emmeans.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 200)
units <- sprintf("p%03d", 1:60)
warmth <- rnorm(60)
d <- data.frame(
  plot = rep(units, each = length(t)), t = rep(t, length(units)),
  temp = as.numeric(vapply(warmth, function(w) 1.5 * w + rnorm(length(t), sd = 20),
                           numeric(length(t)))))
y <- matrix(rbinom(60 * 6, 1, plogis(3 * as.numeric(outer(warmth, rep(c(1, -1), 3))))),
            ncol = 6, dimnames = list(units, paste0("sp", 1:6)))
x <- grain_matrix(d, plot, t, temp, grain = c("day", "week", "month"))
lad <- grain_ladder(x, y, elasticnet(), folds = fold_map(y, v = 5), verbose = FALSE)
grain_contrasts(lad)
```

---

grain_ladder *Fit at every grain and see where skill saturates*

Description

Cross-validates every learner at every grain of a representation set, on one fold map and one mask of scorable cells, and returns the score of each (grain, learner, variable, fold) cell. It is the measurement the package exists for: how much of a record a model needs, read off the point where making the record finer stops paying.

Usage

```
grain_ladder(
  x,
  y,
  learners,
  folds = NULL,
  response = "presence_absence",
  metric = NULL,
  control = train_control(),
  keep_fits = FALSE,
  interval = c("variables", "nested_cv"),
  repeats = 1L,
  seed = 1L,
  verbose = TRUE
)

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

Arguments

x	A grain_matrix() result, a timesift_set(), or a named list of representations.
y	The response for the same units.
learners	A learner, a set of them from c(), a list, or names of registered ones. An unnamed set is labelled by each learner's own name.
folds	A fold map from fold_map(), or any named integer vector. Built with the defaults of fold_map() when not given.
response	Name of the registered response head.
metric	Name of a registered metric, or a function of (y, p), or NULL for the response head's own. Whichever it is, it travels with the fit and is what every later rescoring reads; a function is reported as <function>.
control	train_control(), the training settings every neural learner of the ladder reads. A learner carrying a control of its own overrides it on the settings that control names.
keep_fits	Keep every per-fold fitted model, which is what lets occlusion() read a fitted model without refitting it.
interval	"nested_cv" refits every arm inside every outer training set, which is what paired_contrast() reads an interval for the difference in risk off; see select_grain() for what it estimates and what it costs. "variables", the default, fits nothing further, and a contrast is then read across the response variables.
repeats	Repetitions of the nested cross-validation, each on its own fold map. The first is the map the ladder was cross-validated on.
seed	Seed the repetitions' fold maps are drawn under. Two tables whose contrast is to be read take the same folds, repeats and seed.

verbose	Report each arm and each fold as it runs.
object	A ladder.
...	Ignored, so that summary() takes the arguments its generic declares.

Details

Every arm sees identical splits and is restricted to identical cells, so the arms' means share a denominator and any two of them can be compared cell by cell with paired_contrast().

Value

A data frame of one row per scored cell, of class timesift_ladder, carrying the grain, the learner, the variable, the fold and the score. The held-out prediction of every unit is kept in the predictions attribute, and the scorable-cell mask in cells.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 200)
units <- sprintf("p%02d", 1:60)
warmth <- rnorm(60)
d <- data.frame(
  plot = rep(units, each = length(t)), t = rep(t, length(units)),
  temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300) + rnorm(length(t)),
                           numeric(length(t)))))
y <- matrix(rbinom(120, 1, plogis(c(warmth, -warmth))), nrow = 60,
            dimnames = list(units, c("sp1", "sp2")))
x <- grain_matrix(d, plot, t, temp, grain = c("week", "month"))
lad <- grain_ladder(x, y, elasticnet(), folds = fold_map(y, v = 3), verbose = FALSE)
summary(lad)
```

---

grain_matrix *Reduce sensor series to a temporal grain*

Description

Bins each unit's readings by the calendar and summarises every bin by one or more statistics, returning the array a model is fitted on. The reduction is the choice the package exists to make explicit: grain sets how coarse the record becomes, stats sets what survives the reduction, and the two are not interchangeable.

Usage

```
grain_matrix(
  data,
  id,
  time,
```

```
  value,
  grain = "day",
  stats = "mean",
  year_start = "01-01",
  partial = c("keep", "drop")
)
```

Arguments

data	A data frame of readings in long form, one row per reading.
id	Column identifying the unit carrying the sensor. A bare column name or a string.
time	Column of reading instants, POSIXct. A bare column name or a string.
value	Column of readings, numeric. A bare column name or a string.
grain	One of "native", "halfday", "day", "week", "month", "season", "year". The four coarse grains follow the calendar, so a bin is a real week or month rather than a fixed block of hours. Naming several grains returns one representation per grain, a timesift_set(). A function is called on the reading instants and must return the POSIXct start of each reading's bin, which is how a calendar the package does not carry, such as astronomical seasons, is binned.
stats	Statistics to compute per bin, one channel each, in the order given. See Details.
year_start	"MM-DD" boundary of the year, used by "season" and "year". Defaults to the calendar year, "01-01"; a hydrological year starting in September is "09-01".
partial	What to do with a bin the record does not cover for its whole calendar span, which is what a record beginning or ending away from a bin boundary produces. "keep", the default, returns it alongside the full bins; "drop" removes it. See Partial bins.

Details

Seven statistics are available, and the distinction between an extreme reading, an extreme day and a typical day is deliberate rather than pedantic:

- mean: arithmetic mean of the readings in the bin.
- min, max: coldest and warmest single reading in the bin.
- cold_day, warm_day: coldest and warmest day, each day first reduced to its own mean. Defined for "day" and coarser.
- mean_daily_min, mean_daily_max: the bin's average daily minimum and average daily maximum, each day first reduced to its own extreme. Defined for "day" and coarser.

An extreme day is a state the unit was in; an extreme reading can be one hour; an average daily extreme is the exposure a typical day of the bin brought. On alpine soil temperature the day-level pair carries more predictive signal than the bin mean, and by more the coarser the bin.

Whether a grain is a day or coarser is decided from the bins rather than from the grain's name, so a supplied calendar that cuts inside a day is refused for these four as well, naming the day it splits.

Nothing is standardised here. Scaling belongs to the fold it is computed on, never to the representation, because computing it over all units would leak held-out units into the input.

Value

A numeric array of shape `[unit, bin, channel]`, with dimnames giving the sorted unit identifiers, the ISO-8601 start of each bin, and the statistic names. Attributes:

- `grain`: the grain name.
- `stats`: the statistic names in channel order.
- `year_start`: the boundary used.
- `bin_start`: the instant each bin begins on the calendar, which is earlier than the bin's first reading wherever the record does not reach the boundary.
- `bin_end`: the last reading instant assigned to each bin.
- `bin_n`: a `[unit, bin]` matrix of how many readings fell in each bin.
- `bin_partial`: a logical vector marking the bins the record does not cover for their whole calendar span.

Naming more than one grain returns a `timesift_set()` of those arrays.

Time zone

Bins follow the calendar the series is carried in, which is the `tzone` attribute of `time`; a column with none is read as UTC, and a name the zone database does not know is an error. The zone is resolved once, at the edge: below it the binning works in local time, where a day is 86400 seconds whatever the night did, so a zone that moves its clock at midnight has no midnight to lose. A bin start is a local time, so reporting it back as an instant needs a rule: one the clock skipped resolves to the instant the clock jumped to, one the clock repeated to the first of the two. Instants are read at whole seconds.

In a zone that keeps daylight saving time a day is therefore a wall-clock day: the day the clock goes forward spans 23 hours and the day it goes back 25, and the week and the month holding them one hour less or more. Hourly readings carried in `"Europe/Vienna"` give a 2021-03-28 day of 23 readings and a 2021-10-31 day of 25. A record kept on a fixed offset, as the Schrankogel deposit is (26,304 hourly readings, 1,096 days of 24), bins into days of 24 hours; a series carried in a daylight-saving zone bins that way too once its `tzone` is set to the fixed offset of standard time, such as `"Etc/GMT-1"` for Central European Time, whose days then run from 01:00 to 01:00 on the summer clock.

The `"native"` grain is the one grain not read on that clock: its bin is the reading itself, so the two readings of an hour a zone repeats are two bins, and the record read at `"native"` is the same array whichever zone it is carried in.

Bins that do not tile the record

Every unit must reach every bin, and consecutive bins must be one bin apart on the grain's own calendar. A bin no unit reaches is never built, so a month missing from the whole record would otherwise pass as four adjacent monthly bins with one simply gone. Neither the `"native"` grain, whose bin is the reading itself, nor a supplied calendar, which declares its own bin lengths, is held to the second rule. `coverage()` lays the same binning out as a count of readings per unit and bin, which is where a refused record's gaps are read off.

Partial bins

A bin is partial when the record does not cover its whole calendar span. Which bins those are follows from where the record starts and stops against the calendar, not from the grain alone: three years of hourly readings from 1 September carry no partial month and no partial season on a `"09-01"` boundary, but the same record carries a partial week at each end, because 1 September is a Wednesday. A record from an arbitrary deployment date carries one at each end of almost every grain.

A bin is partial if its start precedes the first reading of the record, or if its calendar span runs past the last reading plus the record's own sampling interval, taken as the smallest gap between consecutive distinct reading instants. Only a bin at an end of the record can satisfy either, because every unit is required to span every bin in between. The verdict is returned as the `bin_partial` attribute whichever way `partial` is set, so a kept partial bin is labelled rather than silent.

Keeping partial bins is the default because dropping them discards the record's ends: on a seasonal grain that is up to three months of readings at each end. The cost of keeping them is that such a bin's mean is taken over fewer readings and its extremes over fewer days, so `cold_day` and `warm_day` there are drawn from a shorter draw and sit closer to the bin mean than a full bin's would. `bin_n` gives the count the bin was actually reduced from.

A caller-supplied binning declares its own bins, so the package cannot know where the last one was meant to end and takes the record's end as its end. Such a final bin is never reported partial; its leading bin is judged as any other.

Examples

```
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 40)
d <- data.frame(plot = rep(c("a", "b"), each = length(t)),
                t = rep(t, 2),
                temp = c(sin(seq_along(t) / 24), cos(seq_along(t) / 24)))
x <- grain_matrix(d, plot, t, temp, grain = "week",
                  stats = c("cold_day", "mean", "warm_day"))
dim(x)
dimnames(x)[[3]]

ladder <- grain_matrix(d, plot, t, temp, grain = c("day", "week"))
names(ladder)
```

`hierarchical` *Bayesian logistic model with a spatial field*

Description

One model per response: a logistic regression on every bin-by-channel column of the representation, standardised so the prior on a coefficient means the same thing for each, optionally with an intercept for each unit and a Gaussian-process field over the targets' coordinates. A unit then carries what its own record says and what its neighbours' presences say, under the folds every other learner is scored on.

Usage

```
hierarchical(
  data = NULL,
  spatial = c("none", "nngp", "hsgp"),
  random = FALSE,
  cov = c("exponential", "matern32", "matern52", "gaussian"),
  neighbours = 15L,
  m = 6L,
  boundary = 1.5,
  nodes = 5L,
  threads = 1L
)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`spatial`	`"none"` for no field, `"nngp"` or `"hsgp"` for a Gaussian-process field over the coordinates.
`random`	An intercept for each unit.
`cov`	The covariance of the `"nngp"` field: `"exponential"`, `"matern32"`, `"matern52"` or `"gaussian"`, `sigma^2 exp(-d / range)` for the first.
`neighbours`	The neighbours each location of an `"nngp"` field is conditioned on.
`m`	Eigenfunctions per axis of an `"hsgp"` field.
`boundary`	The factor by which an `"hsgp"` field's box is wider than the coordinates.
`nodes`	Grid points per hyperparameter.
`threads`	Conditional fits over the hyperparameter grid run at once. What comes back does not depend on it.

Details

The coefficients, the intercept included, have a `N(0, 2.5^2)` prior. A unit's intercept is `N(0, sd^2)` and the field has a marginal standard deviation and a range, each standard deviation under a penalised-complexity prior (Simpson et al. 2017) with `P(sd > 3) = 0.01`, and the range under one anchored at a fifth of the coordinates' extent with `P(range < anchor) = 0.5` (Fuglstad et al. 2019). Coordinates are centred and divided by one factor, so distances keep their proportions.

The field is a Gaussian process on the two columns named by `coords` in `timesift()`. `"hsgp"` is the Hilbert-space approximation (Solin and Sarkka 2020) with `m` Laplacian eigenfunctions per axis, and `"nngp"` the nearest-neighbour process (Datta et al. 2016) over the distinct locations, each conditioned on its `neighbours` nearest among those before it in lexicographic order of the coordinates, with the covariance `cov`, whose range is the field's. Its sparse precision is factored once per set of hyperparameters, so the fit scales with the number of locations rather than their square.

Inference is Laplace's method over the coefficients, the intercepts and the field together, and the hyperparameters are integrated over on a grid of `nodes` points per hyperparameter, centred on the

mode of their posterior and weighted by it (Rue, Martino and Chopin 2009). With an intercept alone its standard deviation is set at the mode of its posterior; with neither the fit is the posterior mode of the coefficients, so at `spatial = "none"` and `random = FALSE` the learner is a penalised logistic model, every coefficient shrunk towards zero by its prior. A prediction at new units interpolates the field to their coordinates, so `predict.timesift()` is given targets that carry the same coordinate columns. The head's case weights enter the likelihood in every configuration. A response holding one value is predicted its mean and named in `unfitted`. The learner fits a presence-absence head.

With `random = TRUE` each unit, as named by `id` in `timesift()`, gets an intercept, which absorbs what its several targets share beyond the record: it is identified where a unit carries more than one target, as an anchored fit's units do. A prediction adds a unit's intercept where the unit was in the fit and leaves it at zero, the population level, where it was not, so a unit held out whole is predicted from its record and its place alone.

Both languages call one C++ core, `src/ts_hierarchical.cpp`.

Value

A `learner()`.

Examples

```
hierarchical()
hierarchical(data = grain("month"), spatial = "hsgp")
```

---

implied_skill *What population skill a reported level is consistent with*

Description

`tss_inflation()` maps a population skill to the level a design reports for it. This inverts that map: given a level actually read off a ladder, it solves for the population skill whose expected reported level equals it.

Usage

```
implied_skill(
  y,
  folds,
  observed,
  grid = seq(0, 0.95, by = 0.05),
  replicates = 200L,
  seed = 1L
)
```

Arguments

y	The response: a matrix or data frame of units by variables, with unit identifiers in the row names or in a leading character or factor column.
folds	A fold map from fold_map(), or any named integer vector of the same shape.
observed	Reported levels to invert.
grid	Population skills the forward map is measured on before interpolating between them.
replicates	Replicates per value.
seed	Random seed.

Details

It answers the question a level raises once the inflation is known, under the distribution of predictions tss_inflation() plants; a model whose predictions are distributed otherwise is inflated by a different amount. It does not correct a difference between two arms, whose inflations need not be equal.

Value

A data frame of one row per observed level: the level, the population skill it is consistent with, and whether that sits inside the grid the map was measured on.

Examples

```
set.seed(1)
y <- matrix(rbinom(1200, 1, 0.15), nrow = 200,
            dimnames = list(sprintf("p%03d", 1:200), paste0("sp", 1:6)))
implied_skill(y, fold_map(y, v = 5), observed = 0.71, replicates = 40)
```

---

kappa_score *Cohen's kappa, and where two models disagree*

Description

A chance-corrected agreement rate on a two-by-two table. Read against the observed response it is a skill score beside tss(); read between two models' decisions on the same units it says where the two part company.

Usage

```
kappa_score(y, p, rule = c("prevalence", "youden", "kappa", "mpa"))

decision_threshold(y, ...)

## Default S3 method:
```

```
decision_threshold(
  y,
  p,
  rule = c("youden", "kappa", "prevalence", "mpa"),
  perc = 0.9,
  ...
)

## S3 method for class 'timesift'
decision_threshold(
  y,
  candidate = "ensemble",
  rule = c("youden", "kappa", "prevalence", "mpa"),
  perc = 0.9,
  ...
)

model_agreement(y, p_a, p_b, rule = c("youden", "kappa", "prevalence", "mpa"))
```

Arguments

y	Observed presence-absence, 0/1 or logical; for decision_threshold(), also a timesift() fit, which carries its own response and held-out predictions.
p	Predicted scores for the same units, in the same order. Higher means presence.
rule	Threshold rule: "youden", "kappa", "prevalence" or "mpa". kappa_score() cuts at "prevalence" by default, the rule the registered kappa metric reads, and the other functions at "youden".
...	Ignored.
perc	For rule = "mpa", the share of presences the cut must keep, 0.9 by default. Where a rule is named without a call to decision_threshold() it takes that default.
candidate	For a timesift() fit, the candidate whose cut is learned: "ensemble", "selected" or the name of one, as predict() takes it.
p_a, p_b	Two models' predictions for the same units.

Details

Kappa is read at a threshold rather than maximised over one, so the rule that picks the threshold is part of the statistic. "youden" is the operating point tss() is defined at and inherits its selection bias; "kappa" maximises kappa itself and inherits the analogous bias; "prevalence" cuts at the observed presence rate, which selects nothing from the labels and is the rule to read an absolute level at. "mpa" is the minimum predicted area rule: the highest cut that still predicts presence at a share perc of the observed presences, so the labels enter through the presences alone.

Given a timesift() fit, decision_threshold() learns one cut per response from a candidate's out-of-fold predictions of the fit's own targets, which is the cut predict(type = "binary") applies to new targets.

Value

For `kappa_score()`, one number. For `decision_threshold()`, the cut itself, applied as `p >= threshold`, and on a fit one cut per response, named by it, `NA` where the response's held-out predictions give none. For `model_agreement()`, a one-row data frame carrying the agreement kappa between two models cut by the same rule, the share of units they decide differently, and how often each is the one that is right there.

Examples

```
y <- c(0, 0, 0, 1, 1, 1)
kappa_score(y, c(0.1, 0.2, 0.6, 0.4, 0.8, 0.9))
model_agreement(y, c(0.1, 0.2, 0.6, 0.4, 0.8, 0.9), c(0.2, 0.1, 0.3, 0.7, 0.9, 0.8))
```

`learner` *Define a learner*

Description

A learner is a pair of functions: one that fits a model to a representation and a response, and one that predicts from it. Everything the package fits goes through this pair, so a learner of your own sits beside the ones that ship and needs no change to the ladder, the folds or the scoring.

Usage

```
learner(
  name,
  fit,
  predict,
  data = NULL,
  reads = c("tabular", "sequence"),
  multi = c("separate", "joint"),
  control = NULL,
  needs = character(),
  params = list()
)
```

Arguments

`name`	Name the learner is reported under.
`fit`	A function of `(x, y, ...)`, where `x` is a `[unit, bin, channel]` array and `y` the response matrix for the same units, returning a fitted object. A `fit` that declares a `control` argument is handed the resolved `train_control()`, and one that declares a `head` argument is handed the registered response head, whose `loss` and `activation` say what it is fitting toward. The learners that ship read both from there and hold no response of their own. One that declares a `group` argument is handed the grouping the outer fold map keeps whole, one value per

	unit of `x` or `NULL`, so a split it draws inside the fit keeps the same groups whole. One that declares a `weights` argument is handed the head's case weights, a matrix of the response's shape, which is what a rare response weighs in every learner that ships; a fit that declares none fits unweighted.
`predict`	A function of `(model, x)` returning a `[unit, variable]` matrix of predictions for the units of `x`, in that order.
`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`reads`	`"tabular"` or `"sequence"`.
`multi`	`"separate"` where the learner fits one model per response, `"joint"` where one model covers them all. It is handed the whole response matrix either way; this is what the learner says it does with it, and what a report says of the candidate.
`control`	A `train_control()` for this learner alone. The settings it names override the control the run was given; everything else is taken from that one.
`needs`	Packages the learner requires. A learner that cannot run says so at once rather than falling back to something else.
`params`	Settings carried with the learner and passed to `fit`.

Details

A learner declares what it can be handed and how it covers several responses. `reads` is `"tabular"` where the bins reach it as a block of predictors and `"sequence"` where their order in time is what it reads, and `multi` is `"joint"` where one fitted model covers every response and `"separate"` where the learner fits one per response. Either way it is handed the whole response matrix and returns one column per response, so nothing above the learner layer has to know which it was, and the block of predictors is built once for the fit rather than once for every response of it.

`data` pins a learner to one representation. Left `NULL` the learner runs across every representation of the run.

Value

A `timesift_learner`.

Examples

```
# Every bin of every channel as a predictor, one weighted logistic regression per variable.
flat_glm <- learner(
  "flat_glm",
  fit = function(x, y, weights, ...) {
    f <- as.data.frame(as.matrix(x))
    lapply(seq_len(ncol(y)), function(j) {
      stats::glm(y[, j] ~ ., data = f, weights = weights[, j],
                 family = stats::quasibinomial())
    })
  },
  predict = function(model, x) {
    f <- as.data.frame(as.matrix(x))
    vapply(model, function(m) stats::predict(m, f, type = "response"), numeric(nrow(f)))
```

```
  }
)
flat_glm
```

`linear` *A generalised linear model on the flattened representation*

Description

One generalised linear model per variable over every bin-by-channel column, its terms chosen by Akaike's criterion. The family is the response head's: logistic under a binary cross-entropy loss, Gaussian under a squared-error one and Poisson under a Poisson-deviance one. So are the case weights, so a rare response weighs here what it weighs in every other learner.

Usage

```
linear(
  data = NULL,
  select = c("both", "forward", "backward", "none"),
  terms = c("power", "column"),
  max_terms = Inf,
  degree = 2L,
  threads = 1L
)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`select`	`"both"`, `"forward"`, `"backward"` or `"none"`.
`terms`	`"power"` for each power of a column its own term, `"column"` for a column's polynomial as one term.
`max_terms`	Terms a forward or two-way search holds at most; `Inf` for no bound.
`degree`	Polynomial degree each column enters at.
`threads`	How many responses are searched at once, or, with one response to fit, how many of one step's candidate fits run at once. What comes back does not depend on it.

Details

The defaults are biomod2's `GLM`: every column enters as `x + I(x^2)`, and the terms are searched in both directions by AIC as `MASS::stepAIC()` searches them, with no bound on how many are kept. Under the shipped presence-absence head those weights are on, so a default `linear()` is biomod2's `GLM` specification fitted under them; a head registered without `weights` fits it unweighted.

`terms` says what one term is. Under `"power"` each power of a column, `x`, `x^2` and so on, is a term of its own, which is how biomod2 writes a quadratic formula and how `stepAIC()` walks it. Under `"column"` a term is a column's orthogonal polynomial of degree `degree`, so a column enters with its curvature at once and can be non-monotone in the reading the way a niche optimum is. A column holding one value over the fitting units is not a term.

`select` is the search. `"both"` starts from the intercept and at every step takes the move that lowers the criterion most, adding a term or dropping one it holds, while one does and the model holds fewer than `max_terms`. `"forward"` only adds, and `"backward"` starts from every term and only drops. `"none"` fits every term and selects nothing. The two-way and backward searches are `stepAIC()`, step for step: the model as it stands wins a tie, a term whose removal leaves the rank unchanged is dropped first, and an addition that does not raise the rank is not offered. `max_terms` bounds what a forward or two-way search adds, and the backward and unselected fits start from every term whatever it is.

Each fit is R's `glm.fit`: iteratively reweighted least squares, the rank read off the same pivoted decomposition, and the same stopping rule. A move whose fit does not settle within its 25 iterations is refused rather than taken, and the fit names every response whose final model did not settle in `stopped`.

A model with nothing but the intercept predicts the response's share among the fitting units. Selection happens inside whichever units the learner is handed, so under `grain_ladder()` it is redone in every fold. Reported beside a penalised fit it also prices discrete selection: choosing a handful of columns out of hundreds is high variance, and that variance is a cost of the selector rather than of the features. An unselected fit over hundreds of columns separates any response it is given and is meant for a coarse grain, `data = grain("season")`.

The search and the fits run on the core the Python package calls, so the two select the same terms and return the same coefficients.

Value

A `learner()`.

Examples

```
linear()
linear(select = "forward", terms = "column", max_terms = 3)
linear(data = grain("season"), select = "none")
```

`lookback_matrix` *Reduce sensor series to a lookback anchored on each target*

Description

Reads, for every target, a fixed length of record ending a fixed lag before that target's own instant, and summarises it by one or more statistics. It is the reduction a calendar cannot express: two targets on the same unit a fortnight apart read two different stretches of the same series, so the bins are relative to the target rather than to a month or a week.

Usage

```
lookback_matrix(
  data,
  id,
  time,
  value,
  at,
  span,
  lag = "0 days",
  bins = 1L,
  stats = "mean"
)
```

Arguments

`data`	A data frame of readings in long form, one row per reading.
`id`	Column identifying the unit carrying the sensor. A bare column name or a string.
`time`	Column of reading instants, `POSIXct`. A bare column name or a string.
`value`	Column of readings, numeric. A bare column name or a string.
`at`	A data frame of targets with an `id` column naming the unit and an `at` column of anchor instants, `POSIXct`. One row per target; a unit may carry any number of them. The columns are read by name, as every alignment in the package is.
`span`	The lookback's length, as a duration. See Durations.
`lag`	The gap between the anchor and the end of the lookback, as a duration. Defaults to `"0 days"`, which ends the lookback at the anchor itself.
`bins`	How many sub-bins the lookback is cut into, oldest first. `span` must divide by it exactly. One bin gives a block of features; several give a sequence a convolution can read.
`stats`	Statistics to compute per bin, one channel each, in the order given. The same seven `grain_matrix()` carries.

Details

Bin `b` of a target anchored at `a` covers `[a - lag - span + b * step, a - lag - span + (b + 1) * step)`, with `step` the lookback's length divided by `bins` and `b` counted from zero. The interval is closed at the left and open at the right, so a reading on a boundary belongs to the later bin, and only the readings of the target's own unit are read.

Every `(target, bin)` cell must hold at least one reading. A bin lying wholly past either end of the record is an error naming the target and the interval, never a padded row: an invented value in front of a model is worse than a target the record cannot answer for. A bin the record reaches only in part is summarised over the readings it holds, and `bin_n` records how many.

The four day-level statistics reduce each calendar day first, so they are defined only where every day lies whole inside one bin. For a lookback that is two conditions rather than one: `step` must be a whole number of days, and `a - lag - span` must fall on a day boundary. Either failing is an error naming the target.

Value

A numeric array of shape `[target, bin, channel]`, of class `timesift_matrix`. Its rows are the rows of `at`, in `at`'s own order, named by `at`'s row names. Its bins are named by where each one opens relative to the anchor, oldest first, and its channels by the statistic. Attributes:

- `grain`: `"lookback"`.
- `span`, `lag`: the durations, resolved to seconds.
- `bins`: how many bins the lookback was cut into.
- `stats`: the statistic names in channel order.
- `bin_n`: a `[target, bin]` matrix of how many readings fell in each cell.

Durations

`span` and `lag` are read from a count and a unit – `"30 days"`, `"12 hours"`, `"1 year"` – or from a bare number of seconds. A **year is 365 days and a month is 30 days** here. A lookback of a fixed length is a fixed length, not a calendar step: the point of anchoring on the target is that every target reads the same amount of record, which a February and a leap year would take away. Where the calendar is what matters, `grain_matrix()` is the call that follows it.

The units are `seconds`, `minutes`, `hours`, `days`, `weeks`, `months` and `years`, singular or plural.

Time zone

The calendar is the series', taken from the `tzone` attribute of `time` as `grain_matrix()` takes it; a column with none is read as UTC. The anchors are instants and are read as a clock in that same calendar, whatever zone `at` carries, so one record is binned by one calendar. The span is measured on that clock: a lookback of one day ending at a local midnight holds the whole local day before it, which is 25 hours of record on the night a zone sets its clock back and 23 on the night it sets it forward. That is what keeps a calendar day whole inside a bin for the four day-level statistics; a length fixed in instants could not.

See Also

`grain_matrix()`, the reduction that follows the calendar instead.

Examples

```
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 60)
d <- data.frame(plot = rep(c("a", "b"), each = length(t)),
                t = rep(t, 2),
                temp = c(sin(seq_along(t) / 24), cos(seq_along(t) / 24)))
at <- data.frame(id = c("a", "b"),
                 at = as.POSIXct(c("2021-10-20", "2021-10-25"), tz = "UTC"))
x <- lookback_matrix(d, plot, t, temp, at = at, span = "30 days", bins = 3L,
                   stats = c("cold_day", "mean", "warm_day"))
dim(x)
dimnames(x)[[2]]
```

`mars` *Multivariate adaptive regression splines on the flattened representation*

Description

One MARS model per response, over every bin-by-channel column of the representation, fitted as the earth package fits it and as biomod2 fits `MARS`. The forward pass starts from the intercept and at each step multiplies a term already in the model by a pair of hinges on one column, `max(0, x - t)` and `max(0, t - x)`, taking the parent, the column and the knot `t` that most reduce the residual sum of squares of a least-squares fit to the response. A knot at a column's least value enters the column linearly. `degree` bounds how many hinges a term multiplies, so `degree = 1` is an additive model and `2` admits pairwise interactions. The pass stops at `max_terms` terms, when a step raises the R-squared by less than `min_gain`, or when no term reduces the residuals.

Usage

```
mars(
  data = NULL,
  degree = 1L,
  penalty = NULL,
  max_terms = NULL,
  min_gain = 0.001,
  minspan = 0L,
  endspan = 0L,
  fast_k = 20L,
  fast_beta = 1,
  prune = TRUE,
  nprune = NULL,
  threads = 1L
)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`degree`	The most hinges a term multiplies.
`penalty`	The generalised cross-validation's charge per knot. `NULL` is earth's, 2 at degree one and 3 above; -1 charges nothing.
`max_terms`	The most terms the forward pass reaches, the intercept included. `NULL` is `min(200, max(20, 2 p)) + 1` for p columns.
`min_gain`	The least rise in R-squared a forward step is kept for.
`minspan, endspan`	Units between knots, and units at either end of a column no knot is placed among. 0 is Friedman's rule; a negative `minspan` asks for that many knots per column.

`fast_k`, `fast_beta`
: Fast MARS: parents tried at each step, and how fast an untried parent ages up the queue. `fast_k = 0` tries every term.

`prune`
: Whether the pruning pass runs. Without it every term of the forward pass is kept.

`nprune`
: The most terms kept, the intercept included; `NULL` for no bound.

`threads`
: Responses fitted at once, or, with one response to fit, columns searched at once. The model is the same on any number.

Details

The pruning pass then removes terms one at a time, each time the one whose loss raises the residuals least, and keeps the subset of least generalised cross-validation, which charges `penalty` for each knot. Under a presence-absence head the kept terms are refitted as a logistic model, as earth's `glm = list(family = binomial)` refits them and biomod2 asks it to, and the prediction is that model's probability; under a count head they are refitted as a Poisson model with a log link, as earth's `glm = list(family = poisson)` refits them, and the prediction is its mean count; under a squared-error head they are refitted by least squares.

The defaults are earth's own, which biomod2 uses under its default option set and under `"bigboss"` alike: degree one, `penalty = 2`, `min_gain = 0.001`, `max_terms = min(200, max(20, 2 p)) + 1` for `p` columns, Friedman's rules for the spans between knots, and Fast MARS over the 20 best parents. The forward pass, the pruning pass and the refit live in the core the Python package calls, pinned against earth in the fixtures, so the two languages keep the same terms and return the same coefficients.

The case weights are the response head's, `positive_weights()` under presence-absence, and weigh both passes and the refit as earth weighs them. earth refits the whole basis by QR at every candidate knot of a weighted fit; the core reaches the same residual sums with Friedman's running updates, which is what makes a weighted fit over hundreds of columns affordable. Under the shipped presence-absence head those weights are on, so a default `mars()` is earth's specification fitted under them; a head registered without `weights` fits it unweighted.

A response holding one value is predicted its mean.

Value

A `learner()`.

Examples

```
mars()
mars(degree = 2L)
mars(data = grain("season"), nprune = 10L)
```

maxent *Maxent on the flattened representation*

Description

One maximum-entropy model per response, over every bin-by-channel column of the representation, in the formulation of the maxnet package (Phillips et al. 2017) and biomod2's MAXNET: maxnet's feature classes, its regularisation of each feature, and a lasso over them, fitted by the penalised core elasticnet() runs on, which the Python package calls too. With the maxnet package's own settings the features and the penalty factors are maxnet's to rounding, and the fit settles at the objective glmnet reaches for maxnet.

Usage

```
maxent(
  data = NULL,
  classes = NULL,
  regmult = 1,
  formulation = c("background", "absence"),
  type = NULL,
  knots = 50L,
  add_samples = TRUE,
  clamp = TRUE,
  n_inner = 5L,
  s = c("lambda.min", "lambda.1se"),
  tol = 1e-08,
  max_design = 2,
  threads = 1L,
  seed = 1L
)
```

Arguments

data	A representation the learner is pinned to, or NULL to run across every representation of the run.
classes	Feature classes, letters of "lqpht", or NULL for maxnet's choice at each response's presence count.
regmult	Multiplier on every feature's regularisation.
formulation	"background" for maxnet's presence-background model, "absence" for a logistic lasso reading the absences as absences.
type	The background formulation's output: "cloglog" or "logistic". The absence formulation predicts a probability, which is its "logistic".
knots	Points over each column's range the hinges and thresholds are placed at.
add_samples	Add each presence to the background, as maxnet's addsamplestobackground.
clamp	Hold each column and each feature inside the range it was fitted on.

n_inner	Folds of the absence formulation's inner cross-validation.
s	Where the absence formulation reads its path: "lambda.min" or "lambda.1se".
tol	Where the coordinate descent stops, as the elastic net's tol.
max_design	Gigabytes the expanded designs fitted at once may take together.
threads	Responses fitted at once, or, with one response to fit, the fits of the absence formulation's inner cross-validation run at once. What comes back does not depend on it.
seed	Seed for the inner cross-validation's fold draw, which is random and would otherwise make the fit irreproducible.

Details

The feature classes are the letters of classes: l the column itself, q its square, p the product of each pair of columns, h forward and reverse hinges at the interior of knots equally spaced points of each column's range, and t thresholds at 49 interior points of it. Left NULL, they follow the response's presence count as maxnet.formula() has them: "l" under 10 presences, "lq" under 15, "lqh" under 80, and "lqph" from 80 on. A column holding one value over the units fitted takes no feature.

formulation says what the absences are. "background" is maxnet's own and what biomod2 fits as MAXNET: every unit is background, each presence joins the background again unless an absence carries the same readings (add_samples), the background is weighted 100 against a presence's 1, and the model is read at the last of maxnet's 200 penalties, which scale with regmult. Its output is maxnet's type, "cloglog" by default as biomod2 predicts it. "absence" reads the absences as absences: a logistic lasso over the same features and penalty factors under the response head's case weights, positive_weights() under presence-absence, with the penalty chosen by an inner cross-validation dealt as the elastic net's is, and a probability as output.

The background formulation takes no case weights, as maxnet takes none and biomod2 passes none: the background weight is what sets a presence's weight there. Either formulation holds each column inside the range it was fitted on, and each feature inside its own, before predicting, as maxnet's predict(clamp = TRUE) does; clamp = FALSE reads them as they are.

A hinge per column per knot makes the design large: a weekly three-channel representation, 471 columns, is 47,100 features under "lqh", and its products under "lqph" 110,685 more. The design is held in memory with a centred copy beside it, and a fit whose design would take more than max_design gigabytes is refused with the size it would have taken. threads fit that many responses at once, each the fit it gets alone, and the designs held at once share max_design. A coarser representation (data = grain("month")), fewer classes or fewer threads is then the way to fit it.

A response with fewer than two presences, or one whose inner training sets cannot each hold two of each outcome under the absence formulation, is predicted its share among the fitting units, and the fit names it in unfitted. A path that does not settle at a penalty ends there, as the elastic net's does, and is read at its last settled point; the fit names every such response in stopped.

The learner needs a presence-absence response: maxnet has no model for a continuous one, and a head whose loss is not the binary cross-entropy is refused.

Value

A learner().

Examples

```
maxent()
maxent(classes = "lqh", regmult = 2)
maxent(formulation = "absence")
```

---

native *How a series becomes an array a learner reads*

Description

A representation is the way a record is reduced before a learner reads it. It is one of four things: the record as recorded, the record at a calendar grain, several grains side by side in one block of features, or a lookback of fixed length that ends at each target's own instant. A representation holds only its settings. It can therefore be described before any record is seen, name the arm it produced in a fitted object, and be rebuilt for new targets in `predict.timesift()`.

Usage

```
native(stats = "mean", year_start = "01-01")

grain(grain, stats = "mean", year_start = "01-01")

multigrain(grains = NULL, stats = "mean", year_start = "01-01")

lookback(span, lag = "0 days", bins = 1L, stats = "mean")
```

Arguments

`stats`	Statistics computed per bin, one channel each, in the order given. See `grain_matrix()` for the seven and for what separates an extreme reading from an extreme day.
`year_start`	`"MM-DD"` boundary of the year, used by `"season"` and `"year"`.
`grain`	One of `"native"`, `"halfday"`, `"day"`, `"week"`, `"month"`, `"season"`, `"year"`, or a function of the reading instants returning each reading's bin start. See `grain_matrix()`.
`grains`	Grains bound side by side into one block, or `NULL` for the automatic set `grains()` describes.
`span`	The lookback's length, as a duration such as `"30 days"` or a number of seconds. See `lookback_matrix()` for how a duration is read.
`lag`	The gap between a target's instant and the end of its lookback.
`bins`	Sub-bins the lookback is cut into, oldest first. One gives a block of features, several give a sequence.

Details

Every representation carries `label`, the name it is reported under; `kind`, one of `"grain"`, `"multigrain"` and `"lookback"`; the settings its kind uses; and `sequence`, which says whether its bins are ordered in time and so mean something to a convolution. `native()`, `grain()` and a `lookback()` of more than one bin are sequences; `multigrain()` and a one-bin `lookback()` are blocks of features.

`multigrain()` flattens each of its grains to one row per target and puts them side by side, so a column of the block names the grain, the statistic and the bin it came from. It is the tabular representation a penalised regression or a random forest reads.

Value

A `timesift_representation`.

Examples

```
native()
grain("week", stats = c("cold_day", "mean", "warm_day"))
multigrain(c("month", "season"))
lookback("30 days", bins = 3L)
```

---

occlusion *What part of the record a fitted model reads*

Description

Holds one bin of the record back at a time, rescores the held-out units, and records the fall in score as that bin's weight. Nothing is refitted: the models kept by `grain_ladder(keep_fits = TRUE)` are the ones read, so the profile describes the models that produced the reported scores rather than a fresh set of them.

Usage

```
occlusion(x, ...)

## Default S3 method:
occlusion(x, ...)

## S3 method for class 'timesift_ladder'
occlusion(
  x,
  data,
  y,
  arm,
  over = c("bin", "channel"),
  substitute = c("permute", "fold_mean", "unit_mean"),
  metric = NULL,
```

```
  permutations = 20L,
  seed = 1L,
  ...
)

## S3 method for class 'timesift'
occlusion(x, candidate, over = c("bin", "channel"), ...)
```

Arguments

x	A `timesift()` fit, or a `grain_ladder()` result, in either case fitted with `keep_fits = TRUE`.
...	Passed to the method.
data	The representation set the ladder was fitted on.
y	The response it was fitted to.
arm	The arm to read, as `"grain|learner"` or `"learner"`.
over	`"bin"` to hold each bin back in turn, `"channel"` for each channel.
substitute	What a held-back part is replaced by: `"permute"`, `"fold_mean"` or `"unit_mean"`.
metric	Name of the registered metric the rescoring is read by, or a function of `(y, p)`. Left unset it is the one the fit was scored under, so a weight is a fall in the number summary() reports rather than in a second one. `"roc_auc"` is usually the steadier reading over many rescorings: it responds to every reordering of the units, where a maximum over thresholds frequently does not move at all.
permutations	Draws averaged over, for `substitute = "permute"`.
seed	Random seed.
candidate	Name of the candidate to read, for a run, or `"ensemble"` for the combination the run fitted. The members of an ensemble read different bins, so the ensemble is held back by channel: a channel is withheld from every member that carries it, all of them shifted the same way, and the members' predictions are combined by the stack the run fitted.

Details

A model has to be shown something in place of a held-back bin, and what it is shown decides what the weight means. Permuting the bin's values across units keeps the observed readings exactly and cuts only the link between a reading and its unit. Replacing every unit by the fitting-fold mean removes all between-unit variation while keeping the shape of the year. Replacing the bin by each unit's own mean over the record keeps how warm a unit is and removes only that bin's departure from it.

A bin is held back whole: a permutation moves every channel of the bin together, so a unit is shown another unit's week rather than a coldest day from one unit beside a warmest day from another. A channel that is the same for every unit, as `calendar_channels()` are, says where the bin sits rather than what a unit read there, and is left in place.

Read with `over = "channel"` the same machinery asks what each statistic of a grain carries, holding one channel back across the whole record instead of one bin across all channels.

A column named in `static` is a channel of a representation built from a record, constant across its bins, so `over = "channel"` holds it back; where the run has no record the columns are the bins of one block and `over = "bin"` does.

Reached through a `timesift()` run rather than through a ladder, the profile reads the per-fold models the run was told to keep, so every bin is held back from a model that never saw the units it is rescored on. The candidate is named as `summary()` reports it.

Value

A data frame of one row per held-back part and variable, carrying the mean weight over folds, the score with and without the part, and `importance`: one minus the correlation between the predictions with and without the part, which is how biomod2 scores a predictor and needs no response. Both are read on the cells the fit was scored on, and `importance` is `NA` where either set of predictions is constant.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 120)
units <- sprintf("p%02d", 1:40)
warmth <- rnorm(40)
d <- data.frame(
  plot = rep(units, each = length(t)), t = rep(t, length(units)),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300) + rnorm(length(t)),
                          numeric(length(t)))))
y <- matrix(rbinom(80, 1, plogis(c(warmth, -warmth))), nrow = 40,
            dimnames = list(units, c("sp1", "sp2")))
x <- grain_matrix(d, plot, t, temp, grain = "month")
lad <- grain_ladder(x, y, elasticnet(), folds = fold_map(y, v = 3),
                    keep_fits = TRUE, verbose = FALSE)
head(occlusion(lad, x, y, "month|elasticnet", permutations = 3))
```

`ordinal_metric` *A metric of ordinal classes*

Description

The statistics biomod2 reads a model of an ordinal response by. The response is a column of whole-number classes, the model predicts a number on the same scale, and each prediction is read as the observed class nearest to it, the lower class on a tie. With `m[i, j]` the units of observed class `j` predicted as class `i`, over the `k` classes observed in the cell:

Usage

```
ordinal_metric(y, p, metric = names(.ordinal_metrics))
```

Arguments

y	Observed values.
p	Predictions for the same units, in the same order.
metric	One of the names in the table.

Details

metric	reads
"accuracy"	sum(diag(m)) / sum(m)
"recall"	the mean over the k classes of m[j, j] over the units of class j
"precision"	the mean over the k classes of m[i, i] over the units predicted as i
"f1"	2 P R / (P + R) of the two means, NA where both are zero

A class with no unit, or in which nothing is predicted, adds zero to its mean. The four are registered as ordinal_accuracy, ordinal_recall, ordinal_precision and ordinal_f1.

Value

One number, or NA where the cell defines none.

Examples

```
ordinal_metric(c(1, 1, 2, 3, 3), c(1.2, 2.4, 2, 2.8, 3.4), "accuracy")
```

---

paired_contrast *Compare two arms cell by cell*

Description

Two arms scored on the same held-out units do not necessarily have the same set of defined cells, so a difference of two marginal means is not a difference between the arms. This takes the difference inside each (variable, fold) cell both arms scored, averages it within a variable over its folds, and summarises those per-variable means, the variables being the independent replicates.

Usage

```
paired_contrast(ladder, a, b, interval = c("variables", "nested_cv"))
```

Arguments

ladder	A grain_ladder() result.
a, b	The two arms, each named "grain|learner".
interval	Which interval the row carries. "variables" is the spread across the response variables of this dataset. "nested_cv" is an interval for the difference in the two arms' risk on a new sample, by the nested cross-validation of Bates, Hastie and Tibshirani (2024) read on the difference of the two arms' cell scores, which needs a ladder fitted with grain_ladder(interval = "nested_cv"). The paper gives the estimator for one procedure's error and names the difference between two as an extension; the identity its estimator rests on is on the difference of two losses as much as on one.

Details

Pairing removes the variation between variables and the part of a threshold-selected metric's bias that the design sets, but not the part that belongs to each arm. TSS read at the threshold that maximises it is biased upward where presences are thin, and how far depends on how an arm's predictions are distributed as well as on how many presences the cell holds, so two arms of equal skill on the same cell can carry different biases. A difference in TSS can therefore favour one arm with no difference in skill behind it. A threshold-free metric such as roc_auc() has no cut to choose, and a TSS contrast is best read beside the same contrast under it.

An arm is named whole, by its grain and its learner. A learner named alone would have to take its best grain, and that grain is chosen on the held-out scores the contrast is then read off: the difference becomes one between two maxima, favouring whichever learner ran across more grains, and neither the interval nor the p-value accounts for the choice. It is the mechanism tss_inflation() measures one level down, and here pairing does not cancel it. select_grain() chooses a grain on inner folds instead, and its compare argument contrasts the selection with the arms of a ladder on matched cells.

Value

A one-row data frame: the mean per-variable difference; the centre of the interval and its bounds, on Student's t across variables or on nested cross-validation as interval names, which the row carries; the number of variables the difference favours; the paired cells and variables it rests on; a Wilcoxon signed-rank p-value; and p_method, "exact" where the p-value is read off the exact distribution and "normal" where it is the normal approximation with continuity and tie corrections, which it is when the per-variable differences hold a zero or a tie or number fifty or more.

References

Bates, S., Hastie, T. and Tibshirani, R. (2024). Cross-validation: what does it estimate and how well does it do it? *Journal of the American Statistical Association* **119**(546), 1434-1445. doi:10.1080/01621459.2023.2197686

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 200)
```

```
units <- sprintf("p%02d", 1:60)
warmth <- rnorm(60)
d <- data.frame(
  plot = rep(units, each = length(t)), t = rep(t, length(units)),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300) + rnorm(length(t)),
                          numeric(length(t)))))
y <- matrix(rbinom(120, 1, plogis(c(warmth, -warmth))), nrow = 60,
            dimnames = list(units, c("sp1", "sp2")))
x <- grain_matrix(d, plot, t, temp, grain = c("week", "month"))
lad <- grain_ladder(x, y, elasticnet(), folds = fold_map(y, v = 3), verbose = FALSE)
paired_contrast(lad, "week|elasticnet", "month|elasticnet")
```

`perceptron` *Single-hidden-layer network on the flattened representation*

Description

One network per response, over every bin-by-channel column of the representation, fitted as the nnet package fits it and as biomod2 fits `ANN`. Each of `hidden` logistic units takes a bias and every column, and the output takes a bias, every hidden unit and, with `skip`, every column again. The weights start uniform on `[-range, range]` and are fitted by the variable metric (BFGS) method of Nash (1990), the minimiser nnet uses, on the response head's loss plus `decay` times the sum of the squared weights, biases included. The fit stops after `max_iter` iterations, when the objective falls below `abs_tol`, or when an iteration lowers it by no more than `rel_tol` of itself.

Usage

```
perceptron(
  data = NULL,
  hidden = NULL,
  decay = NULL,
  range = NULL,
  max_iter = NULL,
  skip = FALSE,
  standardise = FALSE,
  abs_tol = 1e-04,
  rel_tol = 1e-08,
  preset = c("default", "bigboss"),
  max_hessian = 2,
  threads = 1L,
  seed = 1L
)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.

`hidden`	Hidden units.
`decay`	The weight of the squared weights' sum in the objective.
`range`	The starting weights are uniform on `[-range, range]`.
`max_iter`	The most iterations of the minimiser.
`skip`	Whether the output also takes every column directly.
`standardise`	Whether each column is centred and scaled before the fit.
`abs_tol`, `rel_tol`	The fit stops when the objective falls below `abs_tol`, or when an iteration lowers it by no more than `rel_tol` of itself.
`preset`	Whose defaults the settings left `NULL` take: `"default"` or `"bigboss"`.
`max_hessian`	Gigabytes the minimisers' approximate inverse Hessians may take together.
`threads`	Responses fitted at once.
`seed`	Seed for the starting weights.

Details

Under a presence-absence head the output is the logistic function of its sum and the loss the cross-entropy, nnet's `entropy = TRUE`; under a head with a squared-error loss the output is the sum itself and the loss the sum of squares, nnet's `linout = TRUE`; under a count head the output is the exponential of the sum and the loss the Poisson deviance, which nnet does not offer. biomod2 leaves nnet's own `entropy = FALSE`, fitting a presence-absence response by least squares on the logistic output; the learner fits the head's loss, as every learner does.

`preset` says whose defaults the settings left `NULL` take. `"default"` is what biomod2's default option set fits: two hidden units, as biomod2 sets them, and nnet's own `decay = 0`, `range = 0.7` and `max_iter = 100`. `"bigboss"` is biomod2's tuned option set: five hidden units, `decay = 0.1`, `range = 0.1` and `max_iter = 200`. A setting given explicitly beats either.

nnet reads the columns as given, and so does the default: a record in its own units saturates the hidden units sooner the wider its range. With `standardise = TRUE` each column is centred on its mean and divided by its sample standard deviation over the fitting units, and a prediction centres and scales by the fit's own.

The network, its objective and the minimiser live in the core the Python package calls, pinned against nnet in the fixtures from the same starting weights, so the two languages fit the same network. The starting weights are drawn from the core's own generator, so a fit does not repeat nnet's from the same R seed. `threads` fit that many responses at once, each network the same as when fitted alone. The minimiser holds an approximate inverse Hessian of one number per pair of weights for every network fitted at once; a fit that would need more than `max_hessian` gigabytes for them is refused with the size, and a coarser grain, fewer hidden units or fewer threads shrinks it.

The case weights are the response head's, `positive_weights()` under presence-absence, and weigh each unit's term of the loss as nnet's `weights` do. Under the shipped presence-absence head those weights are on, so a default `perceptron()` is biomod2's `ANN` specification fitted under them; a head registered without `weights` fits it unweighted.

A response holding one value is predicted its mean.

Value

A `learner()`.

Examples

```
perceptron()
perceptron(preset = "bigboss")
perceptron(hidden = 4L, decay = 0.01)
```

`plot.timesift` *Draw a run*

Description

One line per learner across the representations it ran on, read the way a ladder is read, and the stack's held-out score drawn across them, its weights fitted inside each outer training fold. The curves are scored on the folds a choice among them would be judged on, so the best of them sits a little high; the ensemble line does not. Where the ensemble line sits above every curve the candidates are carrying different parts of the signal, and where it sits on or below the best curve they are not.

Usage

```
## S3 method for class 'timesift'
plot(x, col = NULL, interval = TRUE, ...)
```

Arguments

`x`	A `timesift` result.
`col`	One colour per learner, recycled.
`interval`	Draw the interval across responses.
`...`	Passed to `graphics::plot()`.

Value

The table the plot is drawn from, invisibly.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 90)
units <- sprintf("p%02d", 1:30)
warmth <- rnorm(30)
logger <- data.frame(
  plot = rep(units, each = length(t)), datetime = rep(t, 30),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300), numeric(length(t)))))
plots <- data.frame(plot = units,
                    sp_a = rbinom(30, 1, plogis(2 * warmth)),
                    sp_b = rbinom(30, 1, plogis(-2 * warmth)))
fit <- timesift(plots, logger, y = starts_with("sp_"), id = plot, time = datetime,
                sift = grains("week", "month"), resampling = cv(v = 3L), verbose = FALSE)
plot(fit)
```

`plot.timesift_ladder` *Draw a ladder*

Description

One line per learner across the grains, at the across-variable mean of the per-variable score, with a 95 percent interval from its standard error across variables, on Student's t with one degree of freedom fewer than there are variables. An open circle marks each learner's best grain, which is where the curve says the record stops paying for being read more finely.

Usage

```
## S3 method for class 'timesift_ladder'
plot(x, col = NULL, interval = TRUE, ...)
```

Arguments

`x`	A `grain_ladder()` result.
`col`	One colour per learner, recycled.
`interval`	Draw the interval across variables.
`...`	Passed to `graphics::plot()`.

Value

The summary table the plot is drawn from, invisibly.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 200)
units <- sprintf("p%02d", 1:60)
warmth <- rnorm(60)
d <- data.frame(
  plot = rep(units, each = length(t)), t = rep(t, length(units)),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300) + rnorm(length(t)),
                          numeric(length(t)))))
y <- matrix(rbinom(120, 1, plogis(c(warmth, -warmth))), nrow = 60,
            dimnames = list(units, c("sp1", "sp2")))
x <- grain_matrix(d, plot, t, temp, grain = c("day", "week", "month"))
lad <- grain_ladder(x, y, elasticnet(), folds = fold_map(y, v = 3), verbose = FALSE)
plot(lad)
```

plot.timesift_range_change

Draw the change in range

Description

The map of cells lost, kept, absent and gained, for one response.

Usage

```
## S3 method for class 'timesift_range_change'
plot(x, variable = NULL, ...)
```

Arguments

x	A range_change() result made from rasters.
variable	The response to draw; the first by default.
...	Passed to terra::plot().

Value

x, invisibly.

Examples

```
now <- terra::rast(nrows = 10, ncols = 10, vals = rep(c(1, 0), each = 50))
names(now) <- "sp1"
later <- terra::rast(now)
terra::values(later) <- rep(c(0, 1, 1, 0), each = 25)
plot(range_change(now, later))
```

plot.timesift_response_curve

Draw a response curve

Description

One line per response against the value of the predictor. For two predictors, the prediction of one response as a filled image over the grid of the two.

Usage

```
## S3 method for class 'timesift_response_curve'
plot(x, variable = NULL, col = NULL, legend = "topright", ...)
```

Arguments

x	A response_curve() result.
variable	For two predictors, the response to draw; the first by default.
col	One colour per response, recycled.
legend	Where the legend of the responses goes, a position graphics::legend() takes such as "bottomright", or NULL for none. Four responses go in each column of it.
...	Passed to graphics::plot() or graphics::image().

Value

The data frame drawn, invisibly.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 90)
units <- sprintf("p%02d", 1:30)
warmth <- rnorm(30)
logger <- data.frame(
  plot = rep(units, each = length(t)), datetime = rep(t, 30),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300), numeric(length(t)))))
plots <- data.frame(plot = units,
                    sp_a = rbinom(30, 1, plogis(2 * warmth)),
                    sp_b = rbinom(30, 1, plogis(-2 * warmth)))
fit <- timesift(plots, logger, y = starts_with("sp_"), id = plot, time = datetime,
                sift = grains("month"), resampling = cv(v = 3L), ensemble = FALSE,
                n_inner = NULL, verbose = FALSE)
plot(response_curve(fit, "elasticnet / month", "mean", n = 20))
```

---

plot.timesift_selection

Draw how stable the choice of grain was

Description

Every candidate's inner score in every outer fold, one line per outer fold, with an open circle on the candidate that fold selected. A selection that lands on the same candidate each time draws its circles in one column; one that wanders says the grid is flat enough that the choice is arbitrary, which is worth seeing beside the estimate rather than after it.

Usage

```
## S3 method for class 'timesift_selection'
plot(x, col = NULL, ...)
```

Arguments

x	A select_grain() result.
col	One colour per outer fold, recycled.
...	Passed to graphics::plot().

Value

The table of inner scores the plot is drawn from, invisibly, with at, the position of each candidate on the axis, and selected, whether its fold chose it.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 120)
units <- sprintf("p%02d", 1:40)
warmth <- rnorm(40)
d <- data.frame(
  plot = rep(units, each = length(t)), t = rep(t, length(units)),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300) + rnorm(length(t)),
                          numeric(length(t)))))
y <- matrix(rbinom(80, 1, plogis(c(warmth, -warmth))), nrow = 40,
            dimnames = list(units, c("sp1", "sp2")))
x <- grain_matrix(d, plot, t, temp, grain = c("week", "month"))
sel <- select_grain(x, y, elasticnet(), folds = fold_map(y, v = 3), n_inner = 3,
                    verbose = FALSE)
plot(sel)
```

---

positive_weights *Case weights that balance a rare response*

Description

The weight every learner that ships fits a presence-absence response under: each presence of a response weighs the ratio of absences to presences among the fitting units, capped, and each absence weighs one. A response with a presence in one target of a hundred is otherwise fitted away by any learner that minimises a mean loss, and the encoders, the penalised fit, the forest and the forward search would each have to decide that for themselves.

Usage

```
positive_weights(y, cap = 50, fitting = NULL)
```

Arguments

y	The response matrix, [unit, variable], 0/1.
cap	Ceiling on the weight a presence is given, at least one.
fitting	A logical vector over the rows of y, TRUE for the units the model is fitted on, or NULL for all of them. The ratio is read off those rows alone.

Details

The ratio is read off the units the model is fitted on and the weight applies to every unit handed in. An encoder holds part of its units back as an inner validation set and reads the loss it stops on under the same weights, so the loss that stops the fit is the loss the fit minimises; a unit held back that way is not in the count the ratio is made from.

The weights are the response head's: the shipped presence-absence head carries this function as its `weights`, and a head registered with `weights = function(y, fitting) positive_weights(y, cap = 20, fitting = fitting)` weights every learner by that cap instead. A head without `weights` is fitted unweighted.

Value

A numeric `[unit, variable]` matrix of case weights, one per cell of `y`.

Examples

```
y <- cbind(rare = c(1, 0, 0, 0, 0, 0), common = c(1, 1, 1, 0, 0, 0))
positive_weights(y)
positive_weights(y, fitting = c(TRUE, TRUE, TRUE, TRUE, FALSE, FALSE))
```

`predict.timesift` *Predict from a fitted timesift*

Description

Rebuilds every member's representation for the new targets from the settings its own arm was built with, predicts with the model refitted on all targets, and combines them where the ensemble is asked for.

Usage

```
## S3 method for class 'timesift'
predict(
  object,
  targets,
  series = NULL,
  candidate = "ensemble",
  type = c("response", "binary", "spread"),
  rule = c("youden", "kappa", "prevalence", "mpa"),
  alpha = 0.05,
  ...
)
```

Arguments

object	A timesift() fit.
targets	A data frame of targets, carrying the identifier, the anchor and the static columns the fit was given.
series	The long table of readings for those targets, or NULL for a targets-only fit.
candidate	"ensemble", the stack refitted on every target; "selected", the candidate the rule chose on every target (object$choice); or the name of one candidate.
type	"response" for the prediction on the scale of the response head, a probability of presence under the shipped one, "binary" for presence and absence, or "spread" for how far the ensemble's members disagree.
rule	With type = "binary", the rule of decision_threshold() the cut is learned by.
alpha	With type = "spread", one minus the coverage of the interval.
...	Ignored.

Details

With type = "binary" each response is cut into presence and absence at the threshold decision_threshold() learns from the same candidate's out-of-fold predictions of the fit's own targets: never from predictions of units the model was fitted on, and never from the new targets, which carry no response. rule is the rule that picks it. A binary map of a species is this prediction on one target per map cell.

With type = "spread" the ensemble's members are read side by side rather than combined: ensemble_spread() gives their weighted mean, standard deviation, coefficient of variation and interval at alpha, which is biomod2's EMcv and EMci and, on one target per map cell, an uncertainty map.

Value

A [target, response] matrix of predictions, named by target and in the order the fit carries its own targets: sorted by identifier, or the targets' own order where target_time anchors them. Under type = "binary" an integer matrix of 0 and 1, NA for a response whose held-out predictions give no cut. Under type = "spread" the [target, response, statistic] array of ensemble_spread().

See Also

decision_threshold() for the cuts themselves.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 90)
units <- sprintf("p%02d", 1:30)
warmth <- rnorm(30)
logger <- data.frame(
  plot = rep(units, each = length(t)), datetime = rep(t, 30),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300), numeric(length(t)))))
plots <- data.frame(plot = units,
                    sp_a = rbinom(30, 1, plogis(2 * warmth)),
```

```
                    sp_b = rbinom(30, 1, plogis(-2 * warmth)))
fit <- timesift(plots, logger, y = starts_with("sp_"), id = plot, time = datetime,
                sift = grains("week", "month"), resampling = cv(v = 3L), verbose = FALSE)
head(predict(fit, plots, logger), 3)
predict(fit, plots, logger, type = "binary", rule = "prevalence")[1:3, ]
```

`project` *Project a fit onto rasters*

Description

Predicts one target per cell of a raster, each carrying the record of its own cell, and returns the predictions as a raster with one layer per response. This is `BIOMOD_Projection()` and `BIOMOD_EnsembleForecasting()`: a map is the fit applied to one target per cell, at the grain the fit reads.

Usage

```
project(
  fit,
  series = NULL,
  static = NULL,
  candidate = "ensemble",
  type = c("response", "binary", "spread"),
  chunk = 5000L,
  ...
)
```

Arguments

`fit`	A `timesift()` result.
`series`	A terra::SpatRaster whose layers are instants, for a fit made on a record; for a record of several variables, a list of them named as the fit's `x` columns. Every raster shares one geometry and one set of instants.
`static`	A `SpatRaster` whose layers are named as the fit's `static` columns.
`candidate, type, ...`	As for `predict.timesift()`: the candidate (the ensemble by default), `"response"`, `"binary"` or `"spread"`, and the arguments they take.
`chunk`	Number of cells predicted at once.

Details

The record of a cell is its values through the layers of `series`, which are placed in time by `terra::time()`, and the static predictors of a cell are its values in the layers of `static`, named as the columns of `targets` were when the fit was made. A fit made with `coords` reads the centre of each cell. Cells are predicted in chunks, so a raster larger than memory is read a chunk of cells at a

time. A cell is predicted where every input holds a value at that cell; a cell with a missing reading anywhere is NA in every layer.

A map for a later period is the same call with the later record, and the binary maps of the two are what range_change() compares.

Value

A SpatRaster of the geometry of the inputs, one layer per response. Under type = "spread" one layer per response and statistic, named response.statistic.

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  set.seed(1)
  r <- terra::rast(nrows = 6, ncols = 5, xmin = 0, xmax = 5, ymin = 0, ymax = 6)
  elev <- terra::setValues(r, rnorm(30))
  slope <- terra::setValues(r, rnorm(30))
  names(elev) <- "elev"
  names(slope) <- "slope"
  static <- c(elev, slope)
  cells <- data.frame(cell = seq_len(30), elev = elev[][, 1], slope = slope[][, 1])
  cells$sp1 <- rbinom(30, 1, plogis(cells$elev))
  cells$sp2 <- rbinom(30, 1, plogis(-cells$slope))
  fit <- timesift(cells, y = c(sp1, sp2), id = cell, static = c(elev, slope),
                  learners = elasticnet(), ensemble = FALSE, n_inner = NULL,
                  resampling = cv(v = 3), verbose = FALSE)
  project(fit, static = static, candidate = "elasticnet / static")
}
```

---

pseudo_absences *Draw pseudo-absences from a pool of background units*

Description

Presence-only records give a response of ones. A model needs units where the response is zero, and these are drawn from pool, a table of background units that carry the same columns as the targets (an identifier, the static predictors, the coords, and a record in series for each). The strategies are bm_PseudoAbsences()'s:

Usage

```
pseudo_absences(
  pool,
  presences,
  n,
  strategy = c("random", "sre", "disk"),
  id = NULL,
```

```
  env = NULL,
  quantile = 0.025,
  coords = NULL,
  dist_min = 0,
  dist_max = Inf,
  lonlat = FALSE,
  repeats = 1L,
  seed = 1L
)
```

Arguments

`pool`	A data frame of background units.
`presences`	A data frame of the presences, with the columns the strategy reads.
`n`	Number of pseudo-absences in each draw.
`strategy`	`"random"`, `"sre"` or `"disk"`.
`id`	The column naming a unit in both tables, or `NULL` where no unit of the pool is a presence.
`env`	For `"sre"`, the numeric columns the envelope is drawn on.
`quantile`	For `"sre"`, the share of the presences left outside at each end of a column, in `[0, 0.5]`.
`coords`	For `"disk"`, the two columns holding the coordinates.
`dist_min`, `dist_max`	For `"disk"`, the least and greatest distance from the nearest presence.
`lonlat`	For `"disk"`, whether `coords` are longitude and latitude in degrees.
`repeats`	Number of independent draws, each with its own seed.
`seed`	Random seed of the first draw; draw `r` uses `seed + r - 1`.

Details

- `"random"`: any unit of the pool that is not a presence.
- `"sre"`: a unit outside the envelope of the presences, the band between the `quantile` and `1 - quantile` quantiles of each of the `env` columns over the presences, as `envelope()` draws it. A unit is outside where it leaves the band in at least one column.
- `"disk"`: a unit whose distance to the nearest presence lies between `dist_min` and `dist_max`, both included, on `coords`. The distance is planar in the units of the coordinates, or in metres from longitude and latitude in degrees where `lonlat` is `TRUE`, on a sphere of radius 6371008.8 m.

A presence is never its own absence: a unit of the pool whose `id` is one of the presences' is not a candidate. The units a strategy admits are the same in both languages; which `n` of them are drawn depends on the language's generator, as the folds of `fold_map()` do. To fix the units, draw once and keep the table. A set of one's own is the rows of `pool` it names, and needs no strategy.

The drawn units are not flagged in a fit: bind them to the presences with a response of zero and fit as one table. Their `pseudo` column says which they are, and `set` which draw.

Value

The drawn rows of `pool`, with the columns `set` (the draw) and `pseudo` (`TRUE`), of class `timesift_pseudo_absences`.

Examples

```
pool <- data.frame(cell = 1:200, x = rep(1:20, 10), y = rep(1:10, each = 20),
                   temp = rnorm(200))
presences <- pool[c(5, 25, 45), ]
pseudo_absences(pool, presences, n = 10, strategy = "disk", id = "cell",
                coords = c("x", "y"), dist_min = 3, dist_max = 8)
```

---

range_change *How the range of each response changes between two projections*

Description

Counts the cells a response is lost from, kept in and gained, between a map for the present and a map for another period or scenario, as `BIOMOD_RangeSize()` does. A cell is in the range where the response is predicted present.

Usage

```
range_change(now, later, threshold = NULL)
```

Arguments

`now`, `later`	The two maps: a terra::SpatRaster with a layer per response, such as `project()` with `type = "binary"` returns, or a matrix of cells by responses. Layers are matched by position and must be as many.
`threshold`	`NULL` where the maps are already 0 and 1. Otherwise one cut per response, or one for all, at or above which a cell is present; the cuts `decision_threshold()` learns for a fit are what to give.

Details

With `L` the cells lost, `K` the cells kept, `G` the cells gained and `A` the cells absent in both: the current range is `L + K`, the later one `K + G`, `percent_loss` is `100 L / (L + K)`, `percent_gain` is `100 G / (L + K)` and `change` is `percent_gain - percent_loss`, the change in range size as a share of the current one. A cell that is `NA` in either map is left out of every count.

The `map` codes a cell as biomod2 does: `-2` lost, `-1` kept, `0` absent in both and `1` gained.

Value

A `timesift_range_change`: a list with the `table`, one row per response, and the `map`, a raster or a matrix of the codes above, of the shape of the inputs.

Examples

```
now <- cbind(sp1 = c(1, 1, 1, 0, 0, NA), sp2 = c(0, 0, 1, 1, 1, 1))
later <- cbind(sp1 = c(1, 0, 0, 1, 1, 1), sp2 = c(0, 0, 1, 1, 0, 0))
range_change(now, later)$table
```

`register_learner` *Register a learner*

Description

Makes a learner available by name to `grain_ladder()` and to `learners()`. The learners that ship are registered the same way, so there is no list of names inside the fitting code.

Usage

```
register_learner(name, constructor, overwrite = FALSE)

learners()
```

Arguments

`name`	Name the learner is asked for by.
`constructor`	A function returning a `learner()`.
`overwrite`	Replace an existing registration.

Value

The constructor, invisibly.

Examples

```
learners()
```

register_metric *Register a metric*

Description

A metric scores one held-out cell: the observed response of the units in a fold and a model's predictions for them. Registering one makes it available to `grain_ladder()` by name, with no change to the fitting code.

Usage

```
register_metric(name, fn, overwrite = FALSE)

metrics()
```

Arguments

`name`	Name the metric is asked for by.
`fn`	A function of `(y, p)`, the observed values and the predictions for one cell, both vectors of the same length, returning one number or `NA` where the cell defines none.
`overwrite`	Replace an existing registration.

Value

The registered function, invisibly.

Examples

```
register_metric("hit_rate", function(y, p) mean((p >= 0.5) == (y == 1)), overwrite = TRUE)
"hit_rate" %in% metrics()
```

register_response *Register a response head*

Description

A response head says what the values being predicted are, how they reach a learner, and which cells of the (variable, fold) grid a score is defined on. Presence-absence with a joint multi-label head is what ships; an abundance or phenology response is a registration rather than a second fitting path.

Usage

```
register_response(name, spec, overwrite = FALSE)

responses()
```

Arguments

name	Name the response is asked for by.
spec	A list with elements prepare(y), returning the numeric matrix a learner is fitted on; activation, the name of the output transform ("sigmoid", "identity" or "exp"); loss, the name of the training objective ("binary_cross_entropy", "squared_error" or "poisson_deviance"); metric, the default metric name; and cells(y, folds), returning the mask of scorable cells. Every learner that ships reads loss and activation from here: the encoders train under the loss and predict through the activation, and the learners fitting one model per response take the family the loss names, logistic, Gaussian or Poisson. The combiner minimises the same loss. An optional weights(y, fitting) returns a [unit, variable] matrix of case weights every learner fits under, with fitting a logical vector marking the rows the model is fitted on: whatever the head reads off the response, it reads off those rows, and it weights every row, so an encoder's inner validation loss is weighted as its fit is. The shipped head's is positive_weights(), and a head without one fits unweighted.
overwrite	Replace an existing registration.

Value

The registered specification, invisibly.

Examples

```
responses()
```

---

register_tuning *Register the grid a learner is tuned over*

Description

Makes tune(learner) search grid when it is given no grid of its own. The grids of the learners that ship are registered the same way.

Usage

```
register_tuning(name, grid, overwrite = FALSE)

tunings()
```

Arguments

name	Name of the learner the grid belongs to, as it reports under.

grid	A named list of values to try, or a function of (learner, x) returning one, where x is the representation the learner is fitted on. The second form is for a grid that depends on the data, as the number of columns does, or on a setting the learner carries.
overwrite	Replace an existing registration.

Value

The grid, invisibly.

Examples

```
register_tuning("flat_glm", list(thresh = c(1e-4, 1e-6)), overwrite = TRUE)
tunings()
```

---

regression_metric *A metric of a numeric response and its predictions*

Description

The statistics biomod2 reads an abundance model by. With e = y - p:

Usage

```
regression_metric(y, p, metric = names(.regression_metrics))
```

Arguments

y	Observed values.
p	Predictions for the same units, in the same order.
metric	One of the names in the table.

Details

metric	reads
"r_squared"	1 - sum(e^2) / sum((y - mean(y))^2), NA where y is constant
"pearson"	the correlation of y and p, NA where either is constant
"rmse"	sqrt(mean(e^2))
"mse"	mean(e^2)
"mae"	mean(abs(e))
"max_error"	max(abs(e))
"poisson_deviance"	mean(2 (y log(y / p) - (y - p))), the logarithm taken as zero at y = 0; NA where a prediction i

A comparison across candidates reads the highest score as the best, so the five errors are registered under the names neg_rmse, neg_mse, neg_mae, neg_max_error and neg_poisson_deviance with their sign reversed, and r_squared and pearson under their own names. A cell holding a prediction that is not a number scores NA.

Value

One number, or NA where the cell defines none.

Examples

```
y <- c(1, 2, 3, 4)
p <- c(1.5, 2, 2.5, 5)
regression_metric(y, p, "rmse")
regression_metric(y, p, "r_squared")
```

`response_curve` *How a prediction responds to one predictor*

Description

Varies one predictor of a fitted candidate across the range it takes while every other predictor is held at a reference value, and records the prediction, as biomod2's response curves do. A predictor is a cell of the representation the candidate reads: one statistic in one bin. Given as the name of a channel, `"warm_day"`, it is that statistic moved together in every bin, which is the way a column named in `static` is moved and the way a statistic of the whole record is asked about. Given as `list(bin = "2021-11", channel = "mean")` it is one cell, and a bare bin name is enough where the representation has a single channel.

Usage

```
response_curve(x, ...)

## Default S3 method:
response_curve(x, ...)

## S3 method for class 'timesift'
response_curve(
  x,
  candidate = "ensemble",
  predictor,
  with = NULL,
  fixed = c("mean", "median", "min", "max"),
  n = 50L,
  spread = FALSE,
  ...
)
```

Arguments

`x`	A `timesift()` result.
`...`	Passed to the method.

candidate	The candidate to read, as `summary()` names it, or `"ensemble"`.
predictor	The predictor to vary: a channel, a bin of a one-channel representation, or `list(bin = , channel = )`.
with	A second predictor, given the same way. The prediction is then read over the grid of the two.
fixed	What the other predictors are held at: `"mean"`, `"median"`, `"min"` or `"max"` of the cell over the targets.
n	Number of values of a predictor, equally spaced over the range it takes.
spread	For the ensemble, also return the members' standard deviation and interval at every value, as `predict(type = "spread")` reads them.

Details

The reference is one made-up unit whose every cell holds the `fixed` summary of that cell over the targets. A cell that is the same for every target, as a calendar channel is, keeps the value it has. The curve is read off the model fitted on all targets, not off the per-fold models.

For the ensemble, every member is moved the same way: a member that does not carry the predictor is held at the reference, and the members' predictions are combined by the stack. A member reading another grain carries a channel by its name, so an ensemble is asked about by channel.

Value

A data frame of one row per value and response, of class `timesift_response_curve`: the value of the predictor (`value`, and `value_with` for a second one), the response (`variable`) and the `prediction`, with `sd`, `lower` and `upper` under `spread`.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 120)
units <- sprintf("p%02d", 1:40)
warmth <- rnorm(40)
d <- data.frame(
  plot = rep(units, each = length(t)), t = rep(t, length(units)),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300) + rnorm(length(t)),
                          numeric(length(t)))))
targets <- data.frame(plot = units, sp1 = rbinom(40, 1, plogis(warmth)),
                      sp2 = rbinom(40, 1, plogis(-warmth)))
fit <- timesift(targets, d, y = starts_with("sp"), id = plot, time = t, x = temp,
                learners = elasticnet(),
                sift = grains("month", stats = c("cold_day", "warm_day")),
               ensemble = FALSE, n_inner = NULL, resampling = cv(v = 3), verbose = FALSE)
head(response_curve(fit, "elasticnet / month", "warm_day", n = 5))
```

roc_auc *The area under the ROC curve*

Description

The whole curve rather than its best point. TSS is a maximum over cuts, so a small change in a prediction often moves it not at all and then moves it a long way; the area responds to every reordering, which is what makes it the steadier reading when many rescorings are compared, as in `occlusion()`. Tied predictions take the average rank.

Usage

```
roc_auc(y, p)
```

Arguments

`y`	Observed presence-absence, `0`/`1` or logical.
`p`	Predicted scores for the same units, in the same order. Higher means presence.

Value

One number, or `NA` where the cell defines none.

Examples

```
roc_auc(c(0, 0, 1, 1), c(0.1, 0.2, 0.8, 0.9))
```

scorable_cells *Which cells a score is defined on*

Description

A per-variable score needs both classes among the held-out units, and a per-variable model needs both classes among the units it was fitted on, so a `(variable, fold)` cell where either side of the split is one-class carries no score. The mask says which cells those are.

Usage

```
scorable_cells(y, folds)
```

Arguments

`y`	The response: a matrix or data frame of units by variables, with unit identifiers in the row names or in a leading character or factor column.
`folds`	A fold map from `fold_map()`, or any named integer vector of the same shape.

Details

It is computed from the response and the fold map alone, with no model involved. Every learner in a ladder is then restricted to the same cells, so their means share one denominator and every paired difference runs on matched cells. Computing it from a model instead would let a joint multi-label learner, which emits a number for every cell whether or not it could be fitted per variable, be scored on cells its opponents were never fitted on.

Value

A data frame of one row per `(variable, fold)` cell, of class `timesift_cells`, with the counts on each side of the split and a `scorable` flag.

Examples

```
set.seed(1)
y <- matrix(rbinom(600, 1, 0.2), nrow = 100,
            dimnames = list(sprintf("p%03d", 1:100), paste0("sp", 1:6)))
cells <- scorable_cells(y, fold_map(y, v = 5))
cells
```

---

score_predictions *Score held-out predictions on the cells the mask allows*

Description

The scoring every arm of a ladder and every candidate of a run goes through, reachable on its own for a prediction matrix that came from somewhere else: a combination of arms, a model fitted outside the package, predictions read back from a file.

Usage

```
score_predictions(y, p, folds, cells = NULL, metric = "roc_auc")
```

Arguments

`y`	The response, as a matrix of units by variables.
`p`	Held-out predictions for the same units and variables.
`folds`	A `fold_map()` result, or one fold per unit.
`cells`	A `scorable_cells()` mask. Computed from `y` and `folds` when left unset.
`metric`	Name of a registered metric to read the cells by, or a function of `(y, p)`.

Details

A cell is one response in one fold. Scoring only the cells the mask allows is what keeps two arms comparable, so the mask is computed from the response and the fold map alone and never from a model.

Value

A data frame of one row per variable and fold, carrying the score and whether the cell was scorable.

Examples

```
set.seed(1)
y <- matrix(rbinom(120, 1, 0.4), nrow = 30,
            dimnames = list(sprintf("p%02d", 1:30), paste0("sp", 1:4)))
p <- matrix(runif(120), nrow = 30, dimnames = dimnames(y))
head(score_predictions(y, p, fold_map(y, v = 3)))
```

`select_grain` *Choose the grain inside the training data, and score the whole procedure*

Description

`grain_ladder()` fits every candidate against one fold map and reports the grid, so reading the best grain off it and quoting that grain's score quotes a number the held-out units helped choose. This does the choosing inside the training data instead. Within each outer fold the training units are split again, every candidate is fitted on part of them and scored on the rest, the best is refitted on the whole outer training set, and the outer test fold is predicted once. The estimate that comes back is therefore of the procedure including its choice of grain, which is what an ecologist applying it to a new site would run.

Usage

```
select_grain(
  x,
  y,
  learners,
  folds = NULL,
  n_inner = 5L,
  choose = c("argmax", "coarsest_adequate"),
  cut_rule = NULL,
  interval = c("variables", "nested_cv"),
  repeats = 1L,
  response = "presence_absence",
  metric = NULL,
  compare = NULL,
  control = train_control(),
  seed = 1L,
  verbose = TRUE
)

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

Arguments

x	A grain_matrix() result, a timesift_set(), or a named list of representations. Its names are the grains being chosen between.
y	The response for the same units.
learners	A learner, a list of them, or names of registered ones, as grain_ladder() takes. Named alongside the grains they form the candidate set.
folds	The outer fold map, from fold_map() or any named integer vector. Built with the defaults of fold_map() when not given.
n_inner	Number of inner folds the selection is made on, or a function of the outer training response returning a fold map for those units. A count deals the inner folds by the grouping the outer fold map carries, so what grouped_cv() kept whole outside stays whole inside.
choose	How a candidate is chosen from its inner scores. "argmax" takes the highest. "coarsest_adequate" takes the coarsest candidate whose inner score lies within one standard error of the highest, the one-standard-error rule of Breiman, Friedman, Olshen and Stone (1984) and of Hastie, Tibshirani and Friedman (2009, section 7.10) with coarseness in place of model complexity. See Choosing a candidate.
cut_rule	NULL, or the rule of decision_threshold() a presence-absence cut is learned by: "youden", the cut that maximises TSS, "kappa" or "prevalence". See A cut learned inside the training data.
interval	Which interval to report beside the across-variable one, which is always reported: "variables" for that one alone, or "nested_cv" for an interval for the procedure's risk. See What the interval is for.
repeats	Repetitions of the nested cross-validation, each on its own fold map. The first is the map the estimate was computed on.
response	Name of the registered response head.
metric	Name of a registered metric the selection is made on, or NULL for the response's own. The estimate is reported under every registered metric whichever this is.
compare	A grain_ladder() result on the same units, response and outer fold map, whose arms the selected procedure is contrasted against cell by cell. NULL for no contrast.
control	train_control(), the training settings every neural learner reads, in the inner search and in the refit alike. A learner carrying a control of its own overrides it on the settings that control names.
seed	Seed for the inner splits. Each outer fold splits under seed plus its own number, so no two outer folds inherit the same inner partition.
verbose	Report each outer fold and what it selected as it runs.
object	A selection.
...	Ignored.

Details

A candidate is a `(grain, learner)` pair: the grains are the elements of the representation set, which is where a grain and the statistic its grains are summarised by are both named, and the learners are the ones passed. Both are registry entries or objects built by `learner()`, so a new grain, a new grain summary or a new candidate model widens the search with no change here.

What the estimate is of: the expected held-out score of the whole pipeline, selection included, on units drawn as these were. What it is not: the score of the winning grain. That is higher, by the amount selection buys itself, and the difference between the two is the quantity this function exists to keep out of a reported number. It also does not say the selected grain is the one a mechanism acts at; it says that grain predicted best on the units the selector saw.

The cost is the ladder's, multiplied by the number of inner folds: `v_outer * (v_inner * candidates + 1)` fits. With a neural learner that is where an overnight run goes.

Value

A `timesift_selection`: a list carrying `selected`, one row per outer fold with the candidate it chose, the inner score it chose on, the highest inner score in that fold (`inner_best`) and that score's standard error (`inner_se`); `estimate`, the nested score under every registered metric with its standard error across variables; `contrast`, one `paired_contrast()` row against each arm of `compare`, or `NULL`; `candidates`, the set that was searched; and `scores`, the per-cell rows of the selected procedure under the selection metric, in the layout `grain_ladder()` returns. The held-out prediction of every unit is in the `predictions` attribute and the scorable-cell mask in `cells`. `inner` holds every candidate's inner score and standard error in every outer fold. With `cut_rule` set, the estimate carries the score, its interval and the interval's name in `interval`, one row per metric and interval. With `interval = "nested_cv"` it also carries `nested_cv`, the same rows with the estimator's own quantities beside them, and `final`, the procedure fitted on every unit, whose risk that interval is for. With `cut_rule` set, the estimate carries one further row, `tss_inner_cut`, the procedure's TSS at the learned cuts; `thresholds` holds the cut of every outer fold and variable; and `cut_scores` the per-cell rows it is averaged from, in the layout of `scores`. Both are `NULL` otherwise.

Choosing a candidate

Inside each outer fold every candidate carries an inner score, the mean over variables of its per-variable mean over the inner folds, and a standard error, the standard deviation over the inner folds of the fold's own score (the mean over the variables scored in that fold) divided by the square root of the number of inner folds. `choose = "argmax"` takes the highest inner score, and on an exact tie the candidate declared first.

`choose = "coarsest_adequate"` first finds that highest score and its standard error, calls every candidate scoring at least the highest minus one standard error adequate, and takes the coarsest adequate one. Coarseness is read off the representation as the package holds it: fewer bins is coarser, and between two candidates with the same number of bins, fewer channels is coarser. A tie on both goes to the higher inner score, then to the candidate declared first. Where one candidate scores more than a standard error above every other, the two rules agree; where the inner profile is flat, this one returns the least storage the record can be kept at without a measured loss inside the training data. A standard error that cannot be computed, because a candidate was scored in fewer than two inner folds, is taken as zero, so the rule falls back to the candidates tied with the highest score.

What the interval is for

The across-variable interval, the one every level of the package reports, is the estimate plus or minus a Student's t quantile times the standard error across the response variables. Its spread is the spread of true skill between variables, and it cannot see the error every variable shares, since all of them are fitted and scored on the same units and the same folds. It is an interval over the variables of this dataset, and not an interval for what the procedure would score on a new sample.

`interval = "nested_cv"` adds one that is meant to be, by the nested cross-validation of Bates, Hastie and Tibshirani (2024). Inside every repetition, each outer training set is cross-validated again over the remaining folds of the same map, which gives the mean squared error of a cross-validation estimate as the difference of two terms it can estimate: the squared gap between the inner estimate and the held-out fold's score, less the variance of that fold's score. The paper's error is a mean of per-unit losses; here a fold's score is the mean over the variables scorable in it, the inner estimate is averaged as the reported estimate is, and the variance of a fold's score is its delete-one jackknife variance over the units of the fold, which for a mean of per-unit losses is exactly the paper's `var(e) / |I_k|`. The centre is the nested estimate less the paper's bias correction, its Appendix C, so the interval is for the risk of the procedure fitted on a sample of this size, which is the fit `final` holds. The square root of the estimated mean squared error is held between the jackknife standard error of the estimate and the square root of the fold count times it, as the paper's section 4.3.2 has it.

The width departs from the paper in one respect. The paper's width is the mean squared error of the plain cross-validation estimate, and its bias correction is added to the centre as a shift with no spread of its own. The correction is `1 + (K - 2) / K` times the gap between two cross-validation estimates on the same units, and in the package's benchmark (`inst/benchmark/`) that gap moved from sample to sample by more than the estimate it corrects where the sample was small or the signal absent, so the paper's interval covered a nominal 95% at 0.86 to 0.93 there at ten repetitions. The width here is therefore the same identity applied to the corrected estimator: inside every outer training set the nested cross-validation is run once more, over the unordered triples of folds, which gives the bias-corrected estimate that training set alone would report, and the squared gap between that and the held-out fold's score replaces the plain estimate's; the bounds are the corrected centre's own jackknife standard error, with every prediction held fixed, and the square root of the fold count times it. The paper's width is reported beside it as `se_bates` in `nested_cv`. On the package's cheap recovery design (`tests/testthat/test-interval.R`: 150 units, five outer folds, one repetition, 200 replicates) the interval covers a nominal 95% at 0.96 with a signal and 0.94 without, where the paper's covers 0.955 and 0.895.

The cost is the selection's, multiplied: one repetition fits the procedure once for every unordered pair and every unordered triple of outer folds, `choose(v_outer, 2) + choose(v_outer, 3)` fits, and every repetition after the first refits the outer folds as well. More repetitions steady the estimate of the mean squared error; the paper uses two hundred random splits, which is affordable where a fit is cheap and is not where a fit is a neural network.

A cut learned inside the training data

TSS read at the cut that maximises it on the scored units is biased upward, most where presences are few (`tss_inflation()`). With `cut_rule` set, each outer fold learns one cut per variable on the inner out-of-fold predictions of the candidate it selected, which the inner search has already made for every outer training unit, by `decision_threshold()` under the rule named. The cut is then frozen and the outer test fold's predictions are read at it by `tss()`. No unit of an outer test fold

enters the cut its own fold is read at. The inner out-of-fold predictions come from models fitted on part of the outer training set and the held-out predictions from the refit on all of it, so the cut is learned on predictions of the same candidate from slightly smaller training sets.

References

Bates, S., Hastie, T. and Tibshirani, R. (2024). Cross-validation: what does it estimate and how well does it do it? *Journal of the American Statistical Association* **119**(546), 1434-1445. doi:10.1080/01621459.2023.2197686

See Also

`grain_ladder()` for the grid this selects from, and `paired_contrast()` for the comparison the `contrast` element holds.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 120)
units <- sprintf("p%02d", 1:40)
warmth <- rnorm(40)
d <- data.frame(
  plot = rep(units, each = length(t)), t = rep(t, length(units)),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300) + rnorm(length(t)),
                          numeric(length(t)))))
y <- matrix(rbinom(80, 1, plogis(c(warmth, -warmth))), nrow = 40,
            dimnames = list(units, c("sp1", "sp2")))
x <- grain_matrix(d, plot, t, temp, grain = c("week", "month"))
sel <- select_grain(x, y, elasticnet(), folds = fold_map(y, v = 3), n_inner = 3,
                    verbose = FALSE)
sel
sel$estimate
```

`simulate_records` *Simulate sensor records whose response acts at a known temporal grain*

Description

Draws units carrying a year of sensor readings and a multi-variable presence-absence response whose dependence on the record is a fixed linear functional of the record at one named grain. The grain is therefore known before anything is fitted, which is what makes the output usable for asking whether `select_grain()` finds it.

Usage

```
simulate_records(
  n = 300L,
  mechanism = c("none", "event", "season", "lag"),
  variables = 10L,
  prevalence = 0.1,
  auc = 0.75,
  from = "2021-09-01",
  days = 365L,
  step_hours = 3,
  seasonal = 8,
  offset_sd = 1,
  anomaly_sd = 1,
  anomaly_days = 2,
  offset_effect = 0,
  sensor_sd = 0.3,
  year_start = "09-01",
  seed = 1L,
  draw = 1L
)
```

Arguments

`n`	Number of units to draw.
`mechanism`	Which generating mechanism, see The mechanisms.
`variables`	Number of response variables. Each gets its own weights within the mechanism.
`prevalence`	Marginal probability of presence, shared by every variable.
`auc`	Population area under the ROC curve of the driver against the response.
`from`	First reading instant, `"YYYY-MM-DD"`, read as UTC.
`days`	Length of the record in days.
`step_hours`	Sampling step in hours. Must divide 24.
`seasonal`	Amplitude of the seasonal cycle shared by every unit, in reading units.
`offset_sd`	Standard deviation of the unit-level thermal offset.
`anomaly_sd`	Marginal standard deviation of each unit's AR(1) anomaly.
`anomaly_days`	Correlation time of that anomaly, in days.
`offset_effect`	Weight the unit-level offset enters the driver with. At the default `0` the driver reads the unit's *anomaly* alone, so a grain coarse enough to average the anomaly away loses the signal. At `1` the offset carries the response as well, and since every grain however coarse reports the offset, every grain is then equally good: that is the grain-invariant control, not a temporal mechanism.
`sensor_sd`	Standard deviation of the measurement noise added to the latent record. The response is generated from the latent record; the readings returned carry this noise.

year_start	"MM-DD" boundary of the hydrological year, passed to grain_matrix() when the mechanism's bins are located. Use the same value when representing the readings.
seed	Seed of the design: the weights and the link coefficients. Two calls with the same seed and different draw share a design and draw independent units, which is what lets a held-out deployment sample be drawn from the same population as a training sample.
draw	Seed of the unit draw. Also names the units, so two draws never collide.

Value

A timesift_simulation: a list with readings, the long table grain_matrix() takes; y, the [unit, variable] 0/1 response; driver, the standardised driver z behind it; grain, the true grain or NA; weights, the [reading, variable] weights defining the driver; link, the solved b0 and b1; and design, the settings the draw is reproducible from.

What the true grain is

The response is driven by g_ij = sum_t w_j(t) * a_i(t), a weighted mean of unit i's latent *anomaly*: the record with the seasonal cycle every unit shares and the unit's own constant offset taken out, since neither of those is temporally located and a grain of any width reports both. The weights w_j are constant within the bins of one grain and zero outside a short stretch of them, so g is exactly a linear combination of that grain's bin means. The true grain of a mechanism is the **coarsest grain of grain_matrix() at which g is still an exact linear functional of the representation**: at that grain and at every grain whose bins nest inside it, no information about g has been averaged away, and at any coarser grain some has. Finer grains keep the information but spread it over more coefficients, so they lose to the true grain by variance rather than by bias, which is the tension the selection has to resolve.

The mechanisms

"none" No temporal signal. The driver is a standard normal drawn independently of the record, so nothing in the readings carries information about the response and no grain is correct. grain is NA.

"event" An isolated event. The weights are uniform over three consecutive **day** bins at a fixed calendar position, one position per variable. True grain "day": a week bin mixes the three days with four others and cannot be unmixed.

"season" A smooth seasonal response. The weights are uniform over one whole **season** bin, one season per variable, cycled. True grain "season": month and day bins nest inside a season so they are exact too, a year bin mixes all four seasons.

"lag" A lagged, cumulative response. The weights decay geometrically over four consecutive **week** bins from a fixed anchor, one anchor per variable. True grain "week": day bins nest inside weeks so they are exact too, month bins straddle week boundaries.

How the skill is set rather than emergent

The driver is standardised to a standard normal by its population mean and standard deviation, both computed in closed form from the generating parameters rather than from the drawn units, so every

draw and every chunk of a draw is on the same scale. The response is `y_ij ~ Bernoulli(plogis(b0 + b1 z_ij))` with `b0` and `b1` solved by numerical integration so that the marginal prevalence is `prevalence` and the population area under the ROC curve of `z` is `auc`. `auc` is therefore a ceiling no fitted model reaches: the response is generated from the latent record and the readings carry `sensor_sd` of measurement noise on top of it, and the weights have to be estimated.

See Also

`select_grain()`, which this exists to test, and `grain_matrix()`, whose calendar the weights are defined on.

Examples

```
sim <- simulate_records(n = 40L, mechanism = "event", variables = 2L, days = 60L)
sim
sim$grain
x <- grain_matrix(sim$readings, unit, time, reading, grain = c("day", "month"))
dim(x$day)
```

---

table_metric *A metric of the two-by-two table of decisions*

Description

The predictions are cut into presence and absence, and the table of decisions against observations is summarised. With `H` the hits, `F` the false alarms, `M` the misses and `C` the correct negatives:

Usage

```
table_metric(
  y,
  p,
  metric = names(.table_metrics),
  rule = c("youden", "kappa", "prevalence", "mpa"),
  threshold = NULL,
  perc = 0.9
)
```

Arguments

`y`	Observed presence-absence, `0`/`1` or logical.
`p`	Predicted scores for the same units, in the same order. Higher means presence.
`metric`	One of the names in the table.
`rule`	Threshold rule: `"youden"`, `"kappa"`, `"prevalence"` or `"mpa"`. `kappa_score()` cuts at `"prevalence"` by default, the rule the registered `kappa` metric reads, and the other functions at `"youden"`.

threshold	NULL for the maximum over every cut, or one cut, presence being predicted at p >= threshold.
perc	For rule = "mpa", the share of presences the cut must keep, 0.9 by default. Where a rule is named without a call to decision_threshold() it takes that default.

Details

metric	reads
"pod"	probability of detection, H / (H + M)
"pofd"	probability of false detection, F / (F + C)
"far"	false alarm ratio, F / (H + F)
"sr"	success ratio, H / (H + F)
"accuracy"	(H + C) / n
"bias"	(H + F) / (H + M), the presences called against the presences there are
"or"	odds ratio, H C / (M F)
"orss"	odds ratio skill score, (H C - M F) / (H C + M F)
"csi"	critical success index, H / (H + M + F)
"ets"	equitable threat score, (H - h) / (H + M + F - h) with h = (H + M)(H + F) / n

These are biomod2's evaluation statistics. biomod2 reads each at the cut that brings that statistic closest to its own optimum on a grid of 100 cuts; here the cut is the one rule of decision_threshold() selects, "youden" by default, or a threshold learned elsewhere. Read at the cut that optimises it, some of these are trivial, so the cut is a separate choice and stays fixed while the statistic changes. Each is registered under its name at the default rule, so grain_ladder(metric = "csi") reads it. A registration at another rule is one line: register_metric("csi_kappa", function(y, p) table_metric(y, p, "csi", "kappa")).

A value the table does not define, a zero denominator, is NA, as is a cell of one class.

Value

One number, or NA where the cell defines none.

Examples

```
y <- c(0, 0, 0, 1, 1, 1, 0, 1)
p <- c(0.10, 0.20, 0.35, 0.40, 0.60, 0.90, 0.55, 0.70)
table_metric(y, p, "pod")
table_metric(y, p, "csi", threshold = 0.5)
```

`timesift` *Fit and compare representations of time-varying data*

Description

One call from two tables to a scored comparison and a held-out estimate of choosing among it. `targets` is one row per thing to predict and `series` is the long, time-stamped record belonging to those rows. Every representation in `sift` is built and every learner in `learners` is paired with the ones it can read; each pair is a candidate.

Usage

```
timesift(
  targets,
  series = NULL,
  y,
  x = NULL,
  id = NULL,
  time = NULL,
  target_time = NULL,
  static = NULL,
  coords = NULL,
  learners = NULL,
  sift = NULL,
  ensemble = TRUE,
  resampling = cv(),
  n_inner = 5L,
  choose = c("argmax", "coarsest_adequate"),
  response = "presence_absence",
  metric = NULL,
  control = train_control(),
  keep_fits = FALSE,
  seed = 1L,
  verbose = TRUE,
  .refit = TRUE
)
```

Arguments

`targets`	A data frame, one row per prediction target.
`series`	A long data frame of readings, or `NULL` to fit on `static` alone.
`y`	Columns of `targets` holding the response, as a tidyselect expression such as `starts_with("sp_")`.
`x`	Columns of `series` holding the readings, as a tidyselect expression. Defaults to every numeric column but `id` and `time`.
`id`	Column naming the unit, present in both tables. A bare column name or a string.

time	Column of reading instants in series, POSIXct.
target_time	Column of targets anchoring each row in time, POSIXct. Optional, and what a unit carrying several targets through time needs.
static	Columns of targets carried alongside the representation, as a tidyselect expression. None by default.
coords	The two columns of targets holding each target's coordinates, as a tidyselect expression, for a learner that places a unit in space, such as hierarchical(). They travel with every representation and are not predictors. None by default.
learners	A learner, a set of them from c(), or a list. Defaults to elasticnet().
sift	The representations a learner without a data = of its own is run across. A grains() or lookbacks() set, a set from c(), a bare vector of grain names, a single representation, or a list of them. Defaults to grains("auto").
ensemble	TRUE for the default stack, FALSE for none, or an ensemble() spec.
resampling	The outer split: cv(), grouped_cv(), a fold vector, or a fold_map() result. With repeats above one the run is made once per repeat: scores and cells carry a repeat column and every repeat's folds under numbers of their own, estimate is read off all the repeats' held-out predictions, oof is a target's mean prediction over the repeats, and the stack is fitted on the repeats' out-of-fold predictions together. The models, the per-fold fits and folds are the first repeat's.
n_inner	Number of inner folds the choice and the stack's weights are made on inside each outer training set, a function of the outer training response returning a fold map for those targets, or NULL to compare the candidates on the outer folds without an estimate. A count deals the inner folds by the grouping the outer split carries, as select_grain() does.
choose	How a candidate is chosen from its inner scores, "argmax" or "coarsest_adequate", as in select_grain().
response	Name of the registered response head.
metric	Name of a registered metric, or a function of (y, p), or NULL for the response head's own. It is what the candidates are chosen on and what the report reads. Whichever it is, it travels with the fit and is what every later rescoring reads; a function is reported as <function>. The estimate is also reported under every registered metric.
control	train_control(), the training settings every neural learner reads.
keep_fits	Keep every per-fold fitted candidate beside the refits.
seed	Seed for the inner splits. Each outer fold splits under seed plus its position.
verbose	Report each outer fold as it runs.
.refit	Whether the candidates are refitted on all targets at the end, which is what predict() reads. A repeated resampling asks it of its first run alone.

Details

Within each outer fold of resampling the training targets are split again into n_inner folds. Every candidate is cross-validated on that inner split, choose picks one on its inner score, and the stack's weights are fitted on the inner out-of-fold predictions. Every candidate is then refitted on the whole

outer training set and predicts the outer test fold, and the selected candidate's prediction and the prediction combined under that fold's weights are kept. Nothing the outer test fold holds enters the choice or the weights it is scored under, so `estimate` is of the procedure, selection and stacking included, which is what an ecologist applying it to a new site would run.

The same refits give every candidate an out-of-fold prediction on the outer folds, and `scores` holds those. They say where predictive skill saturates as the record is read more coarsely, which is the measurement the package exists for, but the highest of them is a number the held-out targets helped choose: read the candidates for the shape of the comparison and `estimate` for the level. With `n_inner = NULL` no inner search is run, the candidates are compared on the outer folds alone and no estimate is made.

The cost is `v_outer * (v_inner + 1) * candidates` fits for the evaluation and one refit per candidate on every target, against `v_outer * candidates` for the comparison alone.

Value

A `timesift` object, a list carrying:

- `estimate`: the held-out score of the selected candidate (`arm = "selected"`) and of the stack (`arm = "ensemble"`), one row per metric, with the standard error and the 95% interval across response variables. An interval across the variables of this dataset, not one for a new sample: every variable is fitted and scored on the same targets and folds, so the error they share is not in it. `NULL` with `n_inner = NULL`.
- `selected`: one row per outer fold, the candidate it chose, the inner score it chose on, the highest inner score and that score's standard error; `inner`: every candidate's inner score in every outer fold; `fold_weights`: the stack's weights in every outer fold, one row per fold and a column per candidate, 0 where that fold's stack left the candidate out. All `NULL` with `n_inner = NULL`.
- `predictions`: the held-out prediction of every target under the selected candidate and under the stack.
- `candidates`, `scores` and `oof`: every candidate, its per-cell scores on the outer folds and its outer out-of-fold predictions.
- `choice`, `models`, `stack` and `weights`: the procedure applied to every target, which is what predict() uses. Every candidate is refitted on all of them; `choice` is the candidate `choose` takes on the outer scores, with the outer folds as the split it chooses on, and `stack` holds weights fitted on the outer out-of-fold predictions.
- `representations`, `fits`, `folds`, `cells`, `y`, and the `metric`, `response`, `spec` and `call` it was asked for.

Rules the entry point enforces

`static` is never implicit: a column of `targets` that is neither the response, the identifier nor the anchor is ignored unless `static` names it, because a predictor nobody asked for is worse than one that is missing.

One target row per `id`, unless `target_time` says where in time each row sits. Repeated identifiers without an anchor are an error naming them.

With `target_time`, every representation has to be anchored on the target, so `native()`, `grain()` and `multigrain()` are refused and `sift` must be given as `lookbacks()`. There is no default set of spans, because there is no defensible one.

Without `series`, `static` is the whole predictor block and `sift` is ignored.

What a learner may be handed

A learner declares whether it reads a tabular block or a sequence. A tabular learner given `native()` is refused before anything is built, and a sequence learner given a representation of one bin is refused once the array says how many bins it has. Inside a `sift` expansion such a pair is skipped and reported once by name; named explicitly through a learner's `data =` it is an error.

See Also

`build_representation()` for the array a candidate reads, `fold_map()` for the splits.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 90)
units <- sprintf("p%02d", 1:30)
warmth <- rnorm(30)
logger <- data.frame(
  plot = rep(units, each = length(t)), datetime = rep(t, 30),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300), numeric(length(t)))))
plots <- data.frame(plot = units,
                    sp_a = rbinom(30, 1, plogis(2 * warmth)),
                    sp_b = rbinom(30, 1, plogis(-2 * warmth)))
fit <- timesift(plots, logger, y = starts_with("sp_"), id = plot, time = datetime,
                sift = grains("week", "month"), resampling = cv(v = 3L),
                ensemble = FALSE, verbose = FALSE)
fit
```

`timesift_report` *What a run found*

Description

Two kinds of row. A candidate row is the candidate's mean score on the outer folds, how many responses it scored highest on, and whether one fitted model covered those responses or one was fitted per response. These rows are the comparison: they share folds and cells, so their shape is read across grains and learners, but the highest of them was picked out on the folds it is scored on. The `selected` and `ensemble` rows are the procedure's held-out score, the choice and the weights made inside every outer training fold, and they are the level to quote.

Usage

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

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

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

Arguments

object	A timesift result.
...	Ignored, so that the methods take the arguments their generics declare.
x	A timesift result, or the table this returns.

Details

Both columns beside a candidate's mean are worth reading. A candidate can carry the ensemble without winning a single response, which is what won shows and a mean alone hides; and a joint model and a per-response one reach the same [target, response] matrix by different routes, which is what responses records.

A candidate the run built no representation for, because its learner cannot read the representation it was paired with, is listed with no mean rather than dropped, so the report says what was asked for as well as what ran.

Value

A data frame of class timesift_summary, one row per candidate and, where the run made an estimate, one for the selected candidate and one for the stack. It carries the mean score, its standard error across responses for the two procedure rows, the responses won, how the responses were covered, and scored: "outer folds" for a candidate, "nested" for the procedure. The weights fitted on every target are in the weights attribute and the candidate chosen on every target in choice.

Examples

```
set.seed(1)
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 90)
units <- sprintf("p%02d", 1:30)
warmth <- rnorm(30)
logger <- data.frame(
  plot = rep(units, each = length(t)), datetime = rep(t, 30),
 temp = as.numeric(vapply(warmth, function(w) w + sin(seq_along(t) / 300), numeric(length(t)))))
plots <- data.frame(plot = units,
                    sp_a = rbinom(30, 1, plogis(2 * warmth)),
                    sp_b = rbinom(30, 1, plogis(-2 * warmth)))
fit <- timesift(plots, logger, y = starts_with("sp_"), id = plot, time = datetime,
               sift = grains("week", "month"), resampling = cv(v = 3L), verbose = FALSE)
report <- summary(fit)
```

```
report
attr(report, "weights")
```

`timesift_set` *Several built representations of the same targets*

Description

A named list of arrays covering the same targets and differing only in how the record was reduced: several calendar grains, several lookback spans, a block of features beside a sequence. It is keyed by the label each representation is reported under, which is what `grain_ladder()` fits across and what a candidate in a `timesift()` fit is named by.

Usage

```
timesift_set(x)
```

Arguments

`x`	A named list of built representations – `grain_matrix()`, `lookback_matrix()` or `feature_matrix()` results – or a single one.

Value

A `timesift_set`: the list, keyed by representation label.

Examples

```
t <- seq(as.POSIXct("2021-09-01", tz = "UTC"), by = "hour", length.out = 24 * 60)
d <- data.frame(plot = rep(c("a", "b"), each = length(t)), t = rep(t, 2),
                temp = rnorm(2 * length(t)))
s <- grain_matrix(d, plot, t, temp, grain = c("day", "week", "month"))
s
names(s)
```

`torch_learners` *Sequence encoders with a joint multi-label head*

Description

Three encoders, all fitted the same way and all ending the same way: one encoder maps a unit's representation to an embedding and a single linear layer maps that embedding to one output per variable, so every variable is predicted together from a shared representation. Pooling strength across variables is what makes the rarer ones learnable at all at these sample sizes.

Usage

```
mlp(
  data = NULL,
  hidden = c(512L, 256L),
  dropout = 0.3,
  activation = c("relu", "gelu", "selu"),
  ...
)

cnn(
  data = NULL,
  channels = c(16L, 32L, 64L, 128L),
  kernel = 7L,
  dropout = 0.3,
  ...
)

rescnn(
  data = NULL,
  channels = c(32L, 64L, 128L, 256L),
  blocks_per_stage = 2L,
  kernel = 7L,
  dilations = c(1L, 2L, 4L, 8L),
  dropout = 0.3,
  ...
)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`hidden`	Hidden layer widths, for the fully connected encoder.
`dropout`	Dropout rate.
`activation`	Nonlinearity between the hidden layers of the fully connected encoder: `"relu"`, `"gelu"` or `"selu"`.
`...`	Training settings for this learner, named as in `train_control()`.
`channels`	Channel width of each stage.
`kernel`	Convolution kernel width.
`blocks_per_stage`	Residual blocks in each stage.
`dilations`	Dilation of each stage, cycled if shorter than `channels`.

Details

`mlp()` flattens the channels and builds in no temporal geometry. It is what separates the effect of the model class from the effect of the representation: where it matches a penalised regression, the difference a convolutional encoder makes is the convolution rather than the network.

`cnn()` is blocks of a one-dimensional convolution, batch normalisation, a rectified linear activation and max pooling, then global average pooling. Pooling becomes an identity once a sequence is shorter than its kernel, so the same stack runs at every grain of a ladder, including one bin per year.

`rescnn()` adds dilated residual blocks with squeeze-excitation channel gates, whose stage dilations widen the receptive field toward the seasonal scale without widening the kernel, and concatenates global average with global maximum pooling so extremes reach the head beside the level.

The three constructors carry architecture. How that architecture is trained is `train_control()`, which the run supplies; a setting named in `...` here overrides the run's control for this learner alone. What the head is trained toward is the response head's: its `loss` is the training objective and its `activation` the output transform, so a head registered with a squared-error loss and an identity activation trains the same encoders on a continuous response, and one with a Poisson-deviance loss and an exponential activation trains them on a count.

Every channel is standardised by its own centre and scale, computed over every unit and bin of the fitting units, so a static predictor appended as a channel sits on the same footing as a reading whatever its units are. The channels of `calendar_channels()` are the exception and are read at their own amplitude: they lie on the unit circle already, and being identical across units they reach a network as a bias, whose step under an adaptive optimiser grows with the scale of the input carrying it. Standardised, the four calendar channels of an hourly record move the fully connected encoder's first layer 1.4 times as far per step, and on the Schrankogel record that costs it 0.044 TSS (`inst/reproduce/README.md`).

A fitted encoder holds its weights as plain arrays and rebuilds the network when it predicts, so a fit saved with `base::saveRDS()` predicts after `base::readRDS()` in a fresh session.

Value

A `learner()`.

Examples

```
cnn(epochs = 5)
```

`train_control` *Training settings every neural learner reads*

Description

The one place a training setting is defaulted. An architecture constructor carries its architecture and nothing else, the control carries how that architecture is trained, and both a whole run and a single learner take one, so there is never a second table of defaults to keep in step with this one.

Usage

```
train_control(
  epochs = 60L,
  batch_size = 64L,
```

```
  learning_rate = 0.001,
  weight_decay = 1e-04,
  optimizer = c("adamw", "adam", "sgd"),
  penalty = 0,
  alpha = 0.5,
  schedule = c("cosine", "constant", "plateau"),
  plateau_factor = 0.1,
  plateau_patience = 10L,
  early_stopping = Inf,
  val_frac = 0,
  device = "auto",
  seed = 1L,
  swa = FALSE,
  swa_start = 0.7
)
```

Arguments

epochs	Epoch budget the cosine schedule anneals over.
batch_size	Most targets per optimiser step. The fitting targets are cut into as few batches of at most this many as they divide into, of as equal a length as they can be, so no batch is a remainder of one.
learning_rate	Learning rate, the one the schedule starts from.
weight_decay	The optimiser's weight decay: decoupled from the gradient under `"adamw"`, and added to it as `weight_decay` times each parameter under `"adam"` and `"sgd"`, as torch's optimisers take it.
optimizer	`"adamw"`, `"adam"` or `"sgd"`, torch's optimisers at their own defaults besides the learning rate and the weight decay; `"sgd"` takes no momentum.
penalty	The weight of a penalty added to the loss of every batch: `penalty * (alpha * sum(abs(W)) + (1 - alpha) * sqrt(sum(W^2)))` summed over every weight matrix and kernel of the network, its biases and normalisation scales left out. This is cito's `lambda`, the penalty biomod2's `DNN` fits under. 0 adds nothing.
alpha	The share of the penalty on the absolute weights, as `elasticnet()` reads its `alpha`: 1 penalises the absolute weights alone and 0 the norm alone. cito's `alpha` is `1 - alpha`.
schedule	How the learning rate moves over the epochs: `"cosine"` anneals it to zero over the budget, `"constant"` holds it, and `"plateau"` multiplies it by `plateau_factor` once the loss has not improved for `plateau_patience` epochs, reading the validation loss where `val_frac` holds a set back and the epoch's mean training loss where it does not, as cito's `reduce_on_plateau` reads them. torch's own relative threshold of `1e-4` decides an improvement.
plateau_factor, plateau_patience	The factor a plateau multiplies the learning rate by, and the epochs without improvement that make one.
early_stopping	Epochs without an inner-validation improvement before training stops. Read only where `val_frac` holds a validation set back. `Inf`, the default, never stops:

the whole budget is trained and the epoch with the lowest validation loss is restored.

`val_frac` Share of the fitting targets held back as an inner validation set, used for early stopping and for nothing else. It is never scored as a result. The set is drawn from every fit alike by a plain random permutation, so the fit on all targets that a run ends with also trains on the rest. At the default of 0 nothing is held back: every fitting target is trained on, the whole budget runs, and the fit keeps the last epoch, where the cosine schedule has annealed the learning rate to zero. On the Schrankogel weekly arm that epoch scores 0.007 AUC above the epoch early stopping keeps on a 15 percent split, and within 0.001 of the best epoch read on the test folds.

`device` `"auto"` to take a graphics processor where there is one, NVIDIA's or Apple's, or a device name such as `"cuda"`, `"mps"` or `"cpu"`. A fitted encoder carries the setting rather than the device it resolved to, so a fit made on one machine predicts on another.

`seed` Seed for initialisation, batching and the inner validation split.

`swa` Average the weights of the tail epochs instead of keeping a single epoch. The schedule anneals to `swa_start` of the epoch budget and is then held flat while the remaining epochs' weights are averaged, and the batch-normalisation statistics are recomputed for the average. Early stopping is off while an average is being accumulated, so the averaging grain always runs.

`swa_start` Share of the epoch budget after which averaging begins.

Details

A control records which of its settings were named in the call. Merging two controls therefore moves only the settings that were asked for: a learner given `train_control(epochs = 200)` reads 200 epochs and takes every other setting from the control the run was given.

Value

A `timesift_control`.

Examples

```
train_control()
train_control(epochs = 200L, device = "cpu")
# biomod2's tuned DNN, cito's adam with a penalty and a plateau schedule
train_control(optimizer = "adam", learning_rate = 0.05, weight_decay = 0, penalty = 0.001,
              alpha = 0, schedule = "plateau", plateau_patience = 7L, epochs = 150L,
              batch_size = 100L, val_frac = 0.2, early_stopping = 14L)
```

`tree` *Classification and regression tree on the flattened representation*

Description

One tree per response, over every bin-by-channel column of the representation, grown under rpart's rules: the Gini index under a presence-absence head, the sum of squares under a head with a squared-error loss and the Poisson deviance under a count head, a split only between two distinct values of a column, and the cost-complexity bookkeeping that keeps a split only where it lowers the risk by at least `cp` of the root's. On the same columns, weights and folds the tree is the one rpart grows, split for split, and its complexity table the one rpart reports; the tree is grown by the core the Python package calls, so the two languages grow it identically.

Usage

```
tree(
  data = NULL,
  min_split = NULL,
  min_leaf = NULL,
  cp = NULL,
  max_depth = NULL,
  prune = c("se_sum", "one_se", "min", "none"),
  n_inner = NULL,
  preset = c("default", "bigboss"),
  shrink = 1,
  seed = 1L,
  threads = 1L
)
```

Arguments

`data`	A representation the learner is pinned to, or `NULL` to run across every representation of the run.
`min_split`	Observations a node needs before a split of it is tried.
`min_leaf`	Observations each child of a split keeps.
`cp`	The share of the root's risk a split has to remove to be kept.
`max_depth`	Depth of the deepest node, the root at depth 0; 30 at most.
`prune`	How the grown tree is pruned: `"se_sum"`, `"one_se"`, `"min"` or `"none"`.
`n_inner`	Folds of the inner cross-validation the pruning reads.
`preset`	Whose defaults the settings left `NULL` take: `"default"` or `"bigboss"`.
`shrink`	Under a count head, the coefficient of variation of the gamma prior a leaf's rate is shrunk by; `0` for no shrinkage. rpart's default is `1`.
`seed`	Seed for the inner cross-validation's fold draw.
`threads`	Responses grown at once. What comes back does not depend on it.

Details

The grown tree is pruned back by an inner cross-validation. Its folds are dealt for each response and stratified on it, as the elastic net's are, and `prune` names the rule that reads the complexity table: `"se_sum"` takes the row of least cross-validated error plus its standard error among the rows that keep a split, the last of them where several tie, which is how biomod2 prunes its classification tree; `"one_se"` takes the smallest tree within one standard error of the least cross-validated error; `"min"` the first row reaching the least error; and `"none"` keeps the tree as grown.

`preset` says whose defaults the settings left `NULL` take. `"default"` is rpart's own, which is what biomod2's default option set fits: `min_split = 20`, `min_leaf = round(min_split / 3)` (or `min_split = 3 * min_leaf` where only `min_leaf` is given), `cp = 0.01`, `max_depth = 30` and ten inner folds. `"bigboss"` is biomod2's tuned option set: `min_split = 5`, `min_leaf = 5`, `cp = 0.001`, `max_depth = 10` and five inner folds. A setting given explicitly beats either.

The case weights are the response head's, `positive_weights()` under presence-absence. They weigh every class count, sum of squares and event count the tree is grown on; `min_split` and `min_leaf` count observations, as rpart's do. Under the shipped presence-absence head those weights are on, so a default `tree()` is rpart's specification fitted under them; a head registered without `weights` fits it unweighted.

Under a count head a leaf predicts a rate, and the rate is shrunk towards the rate of the units the tree is grown on, as rpart's `method = "poisson"` shrinks it: the posterior mean of a gamma prior whose coefficient of variation is `shrink`. A split is chosen on the deviance of the unshrunk rates, and a subtree's risk, its complexity and the pruning's cross-validated error are read on the shrunk ones.

Value

A `learner()`.

Examples

```
tree()
tree(preset = "bigboss", prune = "one_se")
```

---

tss *The true skill statistic*

Description

Sensitivity plus specificity minus one, at the threshold that maximises it. This is the metric species distribution modelling reports. The shipped presence-absence response is scored by `roc_auc()` instead, which chooses no threshold, and TSS is reported beside it.

Usage

```
tss(y, p, threshold = NULL)
```

Arguments

y	Observed presence-absence, 0/1 or logical.
p	Predicted scores for the same units, in the same order. Higher means presence.
threshold	NULL for the maximum over every cut, or one cut, presence being predicted at p >= threshold.

Details

A cut may only fall between distinct predictions: units sharing a prediction are decided together, so the same score comes back whatever order they arrived in. A cell holding only presences or only absences has no skill to measure and returns NA rather than a number.

The threshold is chosen on the same units the score is then read on, which is how the metric is defined in the literature and how it is defined here, and it inflates the level where presences are thin. tss_inflation() measures that inflation for a given design. How large it is depends on how a model's predictions are distributed as well as on the design, so two models of equal skill scored on the same cells can carry different inflations, and a paired difference in TSS is not free of it. Given a threshold learned elsewhere, the score is read at that cut instead, which is what select_grain() reports under threshold = with a cut learned on the inner folds.

Value

One number, or NA where the cell defines none.

Examples

```
tss(c(0, 0, 1, 1), c(0.1, 0.2, 0.8, 0.9))
tss(c(0, 0, 1, 1), c(0.9, 0.8, 0.2, 0.1))
tss(c(0, 0, 0, 0), c(0.1, 0.2, 0.8, 0.9))
tss(c(0, 0, 1, 1), c(0.1, 0.6, 0.8, 0.9), threshold = 0.5)
```

---

tss_inflation *How much a self-selected threshold inflates the reported level*

Description

The true skill statistic (TSS) is overestimated when the threshold is chosen to maximise TSS on the same held-out observations used to report the score. The fewer presences there are in the held-out data, the more TSS tends to overestimate population skill. tss_inflation() estimates the TSS that a given sampling design would report for a known population skill. implied_skill() reverses this calculation: given a reported TSS, it estimates the population skill that would yield that score on average under the same sampling design and assumed prediction distribution. This simulation-based matching follows the principle of indirect inference (Gouriéroux et al., 1993).

Usage

```
tss_inflation(y, folds, skill = c(0.6, 0.7, 0.9), replicates = 200L, seed = 1L)
```

Arguments

y	The response: a matrix or data frame of units by variables, with unit identifiers in the row names or in a leading character or factor column.
folds	A fold map from fold_map(), or any named integer vector of the same shape.
skill	Population skill values to simulate.
replicates	Replicates per value.
seed	Random seed.

Details

Predictions are simulated for each value of skill as normal scores of equal variance, at the cell sizes and presence counts of the response and fold map supplied. The TSS is read at its best threshold exactly as grain_ladder() reports it, and the mean reading minus skill is the inflation.

The inflation is an expectation. A level read on one design is optimistic on average, not a bound every reading sits above: a single level can fall below the population skill. Predictions distributed otherwise inflate by a different amount at the same skill, which is also why a paired difference paired_contrast() takes is not free of it.

Value

A data frame with one row per skill value: the skill, the mean level read back, the inflation, and its interval across replicates.

Examples

```
set.seed(1)
y <- matrix(rbinom(1200, 1, 0.15), nrow = 200,
            dimnames = list(sprintf("p%03d", 1:200), paste0("sp", 1:6)))
tss_inflation(y, fold_map(y, v = 5), skill = c(0.6, 0.9), replicates = 40)
```

---

tune *Tune a learner's settings on the training units*

Description

Returns a learner that, whenever it is fitted, searches grid on the units it is handed and fits the setting that scored best. The search is a cross-validation inside those units, so in a run the outer folds never see it: each fold chooses from its own training units, and the score it is then read at is not selected on. biomod2's BIOMOD_Tuning() searches a grid per algorithm by the same device.

Usage

```
tune(learner, grid = NULL, metric = NULL, n_inner = 5L, seed = 1L)
```

Arguments

learner	A learner(), or the name of a registered one.
grid	A named list of values to try, one element per setting, or NULL for the grid registered for the learner.
metric	The registered metric, or a function of (y, p), a setting is scored by. Left unset it is the response head's own.
n_inner	Number of inner folds.
seed	Random seed of the inner folds.

Details

A learner's settings are the ones it carries as params: the arguments of its constructor, as mtry and trees are for forest(). grid names some of them and gives the values to try; the grid is every combination. A value that is itself a vector, as the layer widths of mlp() are, is given as an element of a list. The inner folds are dealt by fold_map() and keep the grouping the outer fold map keeps whole. A setting is scored by the mean over the responses of the mean over inner folds of metric on the cells a score is defined on, and ties go to the first combination in the grid.

What was chosen is recorded: on the fitted model as $model$chosen and $model$table, and in the settings column of the candidate table of a run, which reads the model fitted on all targets.

With grid left unset the learner is searched over the grid registered under its name by register_tuning(), which for the learners that ship is the one BIOMOD_Tuning() searches: mtry of a forest() from 1 to the smaller of 10 and the number of columns; trees, depth and shrinkage of a gbm-style boosting(), shrinkage and colsample of the second-order one; degree and nprune of mars(); degree of discriminant(); regmult of maxent(); quantile of envelope(); hidden of a perceptron() at 2, 4, 6 and 8 with decay at 0.01, 0.05 and 0.1; and the layer width of mlp() at 2, 4, 6 and 8.

Value

A learner() reporting under the name of the one it wraps.

Examples

```
tuned <- tune(forest(), list(mtry = c(2, 4), min_node = c(1, 5)), n_inner = 3L)
tuned
```

Index

additive, 4
artifacts, 5
as.matrix.timesift_matrix, 6
as_sift *(*grains*)*, 31
average_precision, 7

base::readRDS(), *96*
base::saveRDS(), *96*
bind_channels, 8
bind_channels(), *13*
block_cv *(*cv*)*, 15
boosting, 8
boosting(), *103*
boyce_index, 10
build_representation, 11
build_representation(), *6*, *92*

c(), *34*, *90*
c.timesift_learner *(*combine*)*, 13
c.timesift_models *(*combine*)*, 13
c.timesift_representation *(*combine*)*, 13
c.timesift_sift *(*combine*)*, 13
calendar_channels, 12
calendar_channels(), *55*, *96*
cnn *(*torch_learners*)*, 94
combine, 13
coverage, 14
coverage(), *37*
cv, 15
cv(), *90*

decision_threshold *(*kappa_score*)*, 41
decision_threshold(), *21, 22*, *67*, *71*, *81*, *83*, *88*
digest_array, 16
discriminant, 17
discriminant(), *103*

elasticnet, 19
elasticnet(), *51*, *90*
ensemble, 21
ensemble(), *23*, *90*
ensemble_combine, 22
ensemble_fit, 23
ensemble_fit(), *21*, *23, 24*
ensemble_spread, 24
ensemble_spread(), *67*
ensemble_weights, 25
env_cv *(*cv*)*, 15
envelope, 26
envelope(), *70*, *103*

feature_matrix, 27
feature_matrix(), *6*, *94*
fit_learner, 27
fold_map, 29
fold_map(), *5*, *16*, *34*, *41*, *70*, *78, 79*, *81*, *90*, *92*, *102, 103*
forest, 30
forest(), *103*

grain *(*native*)*, 53
grain(), *11*, *92*
grain_contrasts, 32
grain_ladder, 33
grain_ladder(), *27*, *33*, *46*, *55*, *58*, *62*, *72, 73*, *80–82*, *84*, *94*, *102*
grain_matrix, 35
grain_matrix(), *6*, *12*, *14*, *17*, *28*, *32*, *34*, *47, 48*, *53*, *81*, *86, 87*, *94*
grains, 31
grains(), *53*, *90*
graphics::image(), *64*
graphics::legend(), *64*
graphics::plot(), *61, 62*, *64, 65*
grouped_cv *(*cv*)*, 15
grouped_cv(), *28*, *81*, *90*

hierarchical, 38
hierarchical(), *90*

implied_skill, 40
implied_skill(), *101*

kappa_score, 41

learner, 43
learner(), *5*, *10*, *19*, *21*, *26*, *28*, *31*, *40*, *46*, *50*, *52*, *60*, *72*, *82*, *96*, *100*, *103*
learners *(*register_learner*)*, 72
learners(), *72*
linear, 45
lookback *(*native*)*, 53
lookback(), *11*
lookback_matrix, 46
lookback_matrix(), *6*, *12*, *53*, *94*
lookbacks *(*grains*)*, 31
lookbacks(), *90*, *92*

mars, 49
mars(), *18*, *103*
maxent, 51
maxent(), *103*
metrics *(*register_metric*)*, 73
mlp *(*torch_learners*)*, 94
mlp(), *103*
model_agreement *(*kappa_score*)*, 41
multigrain *(*native*)*, 53
multigrain(), *11*, *92*

native, 53
native(), *11*, *92*

occlusion, 54
occlusion(), *34*, *78*
ordinal_metric, 56

paired_contrast, 57
paired_contrast(), *33–35*, *82*, *84*, *102*
perceptron, 59
perceptron(), *103*
plot.timesift, 61
plot.timesift_ladder, 62
plot.timesift_range_change, 63
plot.timesift_response_curve, 63
plot.timesift_selection, 64
positive_weights, 65
positive_weights(), *4*, *10*, *18*, *20*, *31*, *50*, *52*, *60*, *74*, *100*
predict(), *42*, *91*
predict.timesift, 66
predict.timesift(), *11*, *40*, *53*, *68*
predict.timesift_fit *(*fit_learner*)*, 27
print.timesift *(*timesift_report*)*, 92
print.timesift_summary *(*timesift_report*)*, 92
project, 68
project(), *71*
pseudo_absences, 69

range_change, 71
range_change(), *63*, *69*
read_cells *(*artifacts*)*, 5
read_folds *(*artifacts*)*, 5
read_response *(*artifacts*)*, 5
register_learner, 72
register_metric, 73
register_response, 73
register_tuning, 74
register_tuning(), *103*
regression_metric, 75
rescnn *(*torch_learners*)*, 94
response_curve, 76
response_curve(), *64*
responses *(*register_response*)*, 73
roc_auc, 78
roc_auc(), *7*, *58*, *100*

scorable_cells, 78
scorable_cells(), *5*, *23*, *79*
score_predictions, 79
select_grain, 80
select_grain(), *34*, *58*, *65*, *84*, *87*, *90*, *101*
simulate_records, 84
stats::predict(), *28*
summary(), *55*
summary.timesift *(*timesift_report*)*, 92
summary.timesift_ladder *(*grain_ladder*)*, 33
summary.timesift_selection *(*select_grain*)*, 80

table_metric, 87
terra::plot(), *63*
terra::SpatRaster, *68*, *71*
terra::time(), *68*
timesift, 89
timesift(), *11*, *15*, *16*, *22*, *24*, *39*, *40*, *42*, *55*, *56*, *67*, *68*, *76*, *94*
timesift_report, 92

timesift_set, 94
timesift_set(), *14*, *34*, *36*, *37*, *81*
torch_learners, *13*, 94
train_control, 96
train_control(), *28*, *34*, *43*, *44*, *81*, *90*, *95*, *96*
tree, 99
tree(), *31*
tss, 100
tss(), *41*, *42*, *83*
tss_inflation, 101
tss_inflation(), *40*, *41*, *58*, *83*, *101*
tune, 102
tunings *(*register_tuning*)*, 74

write_cells *(*artifacts*)*, 5
write_folds *(*artifacts*)*, 5
write_response *(*artifacts*)*, 5