Package: baggingReserve (via r-universe)

November 12, 2026

Title Bagging and Regression Trees for Individual Claims Reserving

Version 0.2.0

Description Claim-level (individual) loss reserving in non-life insurance, and the means to check claim-level reserves against the payments that followed. A data contract turns a table of claims and a table of payment transactions into a claim-by-development-year panel at a valuation date; a recursion engine completes the payment history of every reported claim one development year ahead, with plugged-in means or simulated payments, for regression trees, their bootstrap aggregation (bagging) and user-supplied learners; payments for incurred-but-not-reported claims are added from chain-ladder claim counts and the average predicted payment per development year; and back-tests score a prediction and a reference method such as the chain ladder against realised payments by calendar year and component. The default estimator is the method of Janoušek and Pešta (2025) <doi:10.1007/s00362-025-01715-9>, one regression tree per development year with bagging for replicate predictions, whose spread describes the bootstrap resampling variation of the predictions of (by default unpruned) trees, and an out-of-bag diagnostic of the one-year regressions; for totals beyond next year simulated instead of plugged-in completion is recommended. Ships a reproducible synthetic individual-claims dataset generated with the individual claims simulator of Wang and Wüthrich, built on 'SynthETIC'.

License GPL (>= 3)

URL https://github.com/HeirOfIsildur/baggingReserve

BugReports https://github.com/HeirOfIsildur/baggingReserve/issues

Depends R (>= 4.1)

Imports codetools, graphics, parallel, rpart, rngtools, stats, utils, withr

Suggests ChainLadder, knitr, R.rsp, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr, R.rsp

Encoding UTF-8

LazyData true

LazyDataCompression xz

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Jan Janoušek [aut, cre] (ORCID:
<https://orcid.org/0009-0007-3731-4416>), Michal Pešta [aut]
(ORCID: <https://orcid.org/0000-0001-5311-0797>)

Maintainer Jan Janoušek <janousek@karlin.mff.cuni.cz>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-12 15:10:08 UTC

RemoteUrl https://github.com/cran/baggingReserve

RemoteRef HEAD

RemoteSha 760786b8fac3673a6e462086b14b6d650c200f20

Contents

as.data.frame.reserve_prediction . . . 3
as_triangle.chain_ladder . . . 4
backtest . . . 5
bag_reserve . . . 9
calendar_year_payments . . . 12
calendar_year_payments.chain_ladder . . . 13
chain_ladder . . . 14
claim_reserves . . . 15
claims_panel . . . 16
ibnr_reserve . . . 19
oob_score . . . 21
plot.bag_reserve . . . 22
plot.reserve_prediction . . . 23
plot.reserve_tree . . . 25
predict.bag_reserve . . . 25
predict.reserve_tree . . . 27
reserve_tree . . . 30
summary.reserve_prediction . . . 32
synthetic_claims . . . 33
synthetic_claims_full . . . 34
synthetic_payments . . . 35
synthetic_payments_full . . . 36
two_part_learner . . . 37

Index **39**

`as.data.frame.reserve_prediction` *Data frames of predictions and back-tests*

Description

`as.data.frame()` on a reserve_prediction gives one row per future cell (accident year, development year, calendar year) with the mean over the replicates of the reported-claim payments and, when attached, of the IBNR component and the total. On a `reserve_backtest` it stacks the per-component tables of `backtest()` with a `component` column.

Usage

```
## S3 method for class 'reserve_prediction'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'reserve_backtest'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A reserve_prediction or a `reserve_backtest`.
`row.names, optional`	Unused, for compatibility with the generic.
`...`	Unused.

Value

A data frame.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
res <- ibnr_reserve(predict(reserve_tree(panel, seed = 1)))
head(as.data.frame(res))
```

`as_triangle.chain_ladder` *Run-off triangles of a claims panel or a reserve prediction*

Description

`as_triangle()` aggregates a claims_panel into an observed payment or reporting-count triangle, or extracts a completed payment triangle from a reserve_prediction. For a panel, `what = "paid"` gives the accident-year by development-year triangle of incremental (or cumulative) payments and `what = "counts"` the accident-year by reporting-delay triangle of numbers of reported claims; cells that are not observable at the valuation date (`acc_year + l > t`) are `NA` and observed cells without claims or payments are `0`. For a prediction, the future cells are filled with the predicted payments of one replicate or of the mean over replicates: the observed cells are kept for `"reported"` and `"total"` and are zero for `"ibnr"`, and no cell is `NA`. `claim_counts()` returns the incremental count triangle of a panel.

Usage

```
## S3 method for class 'chain_ladder'
as_triangle(x, type = c("incremental", "cumulative"), ...)

## S3 method for class 'reserve_prediction'
as_triangle(
  x,
  component = c("reported", "ibnr", "total"),
  replicate = "mean",
  ...
)

as_triangle(x, ...)

## S3 method for class 'claims_panel'
as_triangle(x, what = c("paid", "counts"), cumulative = FALSE, ...)

## S3 method for class 'reserve_triangle'
print(x, digits = NULL, ...)

claim_counts(x)
```

Arguments

`x`	A claims_panel (for `claim_counts()` and the panel method), a reserve_prediction or a chain_ladder object; for `print`, a triangle.
`type`	For a chain_ladder object: the completed `"incremental"` (default) or `"cumulative"` triangle.

...	Passed on by the generic to its methods and by print to the matrix printing method; in the methods, an argument that is not one of theirs (a misspelt one, say) is an error.
component	"reported", "ibnr" or "total"; without an IBNR component "total" falls back to "reported", with a warning.
replicate	Replicate to display, or "mean" for the average over replicates.
what	"paid" (default) or "counts"; panels only.
cumulative	Logical (default FALSE): cumulate along development (or reporting delay) within each accident year; panels only.
digits	Round the cells before printing; NULL prints them as they are.

Value

A numeric matrix of class "reserve_triangle" with the accident years as row names and the development years (or reporting delays) 0, ..., L as column names, carrying the attributes "t", "what" and "cumulative". It is an ordinary matrix in every computation; only its print method hides the attributes behind a heading.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
panel <- claims_panel(synthetic_claims, synthetic_payments, t = 12, L = 11,
                      features = c("Type", "Age", "AccWeekday"))
round(as_triangle(panel) / 1e6, 2)
claim_counts(panel)
# a completed triangle from a prediction on a small subset: all claims of one type
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
small <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
res <- ibnr_reserve(predict(reserve_tree(small, seed = 1)), claim_counts(small))
round(as_triangle(res, "total") / 1e6, 2)
```

---

backtest *Back-test a reserve prediction against realised payments*

Description

Compares the predicted future payments with the payments that actually followed, cell by cell, from a table of claims and a table of payments that extend beyond the valuation date (for the shipped data: synthetic_claims_full and synthetic_payments_full). The truth is split into the component of the claims that were reported by the valuation date (those in the prediction) and the claims reported later (IBNR), and restricted to development years 0, ..., L; payments beyond L are reported separately as the excluded tail. The prediction's replicate range is shown next to the truth; it describes the variability recorded in the prediction (resampling variation of bagged trees, simulated payments of one fitted model, or both; see summary.reserve_prediction()), so "truth inside the range" is a descriptive statement, not a coverage probability.

Usage

```
backtest(
  pred,
  claims,
  payments,
  id = c(claims = "Id", payments = "Id"),
  acc_year = "AccYear",
  event_year = "EventYear",
  paid = "Paid",
  probs = c(0.025, 0.975),
  rep_year = "RepYear",
  type = 7L,
  reference = NULL,
  reference_component = c("total", "reported"),
  observed_to = NULL
)
```

Arguments

`pred`	A reserve_prediction (with or without an IBNR component).
`claims`, `payments`	Claims and payments tables with the columns named below.
`id`, `acc_year`, `event_year`, `paid`	Column names, as in `claims_panel()`; by default those the prediction's panel was built with (see the section on column names).
`probs`	Two increasing probabilities in `[0, 1]` defining the replicate range shown.
`rep_year`	Name of the reporting-year column of `claims` (by default the one the panel was built with), used for the population and chronology checks above and the IBNR claim counts. When `claims` has no such column these are skipped with a warning; `rep_year = NULL` skips them silently.
`type`	Quantile algorithm for the replicate range, as in `stats::quantile()` (an integer from 1 to 9; default 7, as in `summary.reserve_prediction()`).
`reference`	Optional prediction of another method, scored against the same truth (see the section on the reference): a chain_ladder object fitted on the paid triangle of the prediction's panel, or a named numeric vector of payments by future calendar year, with an optional `total` element (as returned by `calendar_year_payments()`). It adds the columns `reference`, `reference_error` and `reference_rel_error` to the table of the component given by `reference_component`.
`reference_component`	The population the `reference` predicts: `"total"` (default; all claims) or `"reported"` (the claims reported by `t`; named vectors only).
`observed_to`	Last fully observed calendar year of `payments`, after the valuation date `t`; `NULL` (default) takes every calendar year of the prediction as observed. See the section on the follow-up.

Details

The truth tables must describe the same portfolio as the prediction, *including* the claims reported after the valuation date: the IBNR component of the truth consists of the claims in `claims` that are absent from the prediction, so selecting the truth by the identifiers of the reported claims removes the IBNR claims from it, and an unrelated claim in `claims` would be counted as IBNR. Select the population by the same criterion as the panel (all claims of a type, say), not by identifiers. When `claims` carries the reporting year (`rep_year`), the function rejects claims absent from the prediction that were reported by the valuation date; it cannot detect IBNR claims that are missing from the truth tables altogether, so the completeness of the truth population remains the caller's responsibility. The claims of the prediction must carry in `claims` the accident years and, when the reporting year is known, the reporting years they were predicted with (the reporting delay is a covariate of the fit); a contradiction is an error. (A prediction saved before the reporting delays were stored in it, `claim_rep_delays`, is only checked for a reporting year by `t`.) Claims whose accident year is not among the prediction's accident years (for example accident years after the valuation date), and their payments, are not scored; their numbers and amount are reported in `not_scored`. Claims that the panel set aside with `claims_panel(beyond_L = "drop")` (reported by `t` with a reporting delay above `L`, recorded in the prediction as `beyond_L_ids`) belong to the portfolio but not to the prediction: they are not counted as missing from it, and their payments after `t`, which all fall beyond development year `L`, are part of the excluded tail, also when their accident year lies outside the prediction's accident years (the panel's default range is determined after setting them aside); `not_scored` records their number and the number and amount of those payments, and does not count them among the claims outside the accident years.

Chronology. The truth is held to the data contract of `claims_panel()`: when `claims` carries the reporting year, a claim reported before its accident year, and a payment dated before the reporting year of its claim (which the model defines as zero, and which the constructor refuses), are errors naming an example; move such a payment to the reporting year, or correct the reporting year, as for the constructor. A claim of the prediction's accident years without a reporting year is treated, as by the constructor, as not reported by `t`: its payments after `t` count as IBNR truth, with a warning. Without a reporting-year column (see `rep_year`) none of these checks, nor the population checks above, can be made.

Column names. The defaults of `id`, `acc_year`, `event_year`, `paid` and `rep_year` are the column names of the input tables of the panel the prediction was made from (stored in the prediction as `mapping`), so that a portfolio with another layout is back-tested with the names given once to `claims_panel()`; a name passed here overrides it. The reporting year follows the rule of `claims_panel()`: when the panel was built from a reporting-delay column (`rep_delay`) and `claims` has that column, the reporting year is the accident year plus the delay, whether or not `claims` also has a reporting-year column, which must then agree with it wherever both are known; otherwise it is the reporting-year column. A `rep_year` passed explicitly is used instead. For a prediction without a mapping (made by an earlier version) the defaults are the names shown in the usage.

Follow-up. Every truth cell up to `observed_to` starts at zero and receives the payments dated in it, so a calendar year without any payment counts as a realised zero. By default (`observed_to = NULL`) the payments are taken to cover every calendar year of the prediction; if the extract ends earlier, pass the last fully observed calendar year as `observed_to`: the cells after it are unobserved (`NA` in `truth_cells`), their rows have `NA` truth, error and range check, and the `total` rows (truth, prediction, range and reference alike) sum the calendar years up to `observed_to` only (`total_years` of the result). The function does not infer the end of the follow-up from the data, because a fully

observed year can have no payment; it warns when `observed_to` is `NULL` and the payments hold no transaction in the last calendar years of the prediction. Payments dated after `observed_to` are not scored.

Reference. A `reference` is scored against the truth of the population it predicts: `reference_component = "total"` (the default) for a prediction of all claims, such as a chain ladder on the panel's paid triangle, which needs a prediction with an IBNR component (attach it with `ibnr_reserve()`); `"reported"` for a named vector predicting the payments of the reported claims only. A chain_ladder reference always predicts all claims, and it must have been fitted on the paid triangle of the prediction's panel: the same accident years, development years `0, ..., L`, valuation date and observed payments (a chain ladder of another triangle can be passed as the named vector `calendar_year_payments()` of it). A named vector must hold every future calendar year of the prediction and no other year, plus optionally a `total`, which must equal the sum over those years; the `total` row of the reference is that sum (over the observed years only when `observed_to` cuts the follow-up).

IBNR claim counts. When `pred` has an IBNR component and `claims` carries the reporting year, the result holds `counts`, the projected and the realised numbers of IBNR claims by accident year: projected, the chain-ladder count projection stored by `ibnr_reserve()` (`ibnr_details$counts_projected`) summed over the cells after the valuation date; realised, the claims of `claims` absent from the prediction and reported after `t` with a reporting delay of at most `L` (and by `observed_to`, when given). The count error is a part of the IBNR error that the pooled mean payment does not explain.

Value

An object of class `"reserve_backtest"`: a list with `tables`, per component a data frame with one row per future calendar year plus `total` (truth, predicted mean, `error` = `predicted` − `truth`, `rel_error` = `error` / `truth`, `NA` where the truth is zero; where the realised amount is negative, a calendar year of net recoveries, `rel_error` has the sign opposite to that of `error`, so an overprediction shows a negative relative error and `error` gives the direction; `reference_rel_error` is defined alike; then the replicate range `lower`, `upper` and whether the truth lies inside it, `inside_range`; the range columns are `NA` for a prediction with a single column, such as a single-tree fit, whose range is degenerate); the cell-level truth matrices (`truth_cells`); the excluded tail (`tail`); `counts` (the IBNR claim counts described above, or `NULL`); `total_years`, the calendar years summed in the `total` rows; `observed_to`; `reference_component` (`NULL` without a reference); `not_scored`, the claims and the payments after `t` outside the prediction's accident years, the payments after `observed_to` and the claims set aside by `beyond_L = "drop"` with their payments after `t` (`n_claims_beyond_L`, `n_payments_beyond_L`, `amount_beyond_L`); the column names used (`columns`); and the settings.

Examples

```
data("synthetic_claims", "synthetic_payments", "synthetic_claims_full",
     "synthetic_payments_full", package = "baggingReserve")
# a small subset for a quick example: the claims of one type
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
res <- ibnr_reserve(predict(reserve_tree(panel, seed = 1)), claim_counts(panel))
# the truth of the same portfolio: all claims of that type, including those reported
# after the valuation date (selecting by the reported claims' identifiers would drop them)
cl_full <- synthetic_claims_full[synthetic_claims_full$Type == "3", ]
pay_full <- synthetic_payments_full[synthetic_payments_full$Id %in% cl_full$Id, ]
```

```
backtest(res, cl_full, pay_full, reference = chain_ladder(as_triangle(panel)))
# an extract that ends three years after the valuation date
short <- pay_full[pay_full$EventYear <= 15, ]
backtest(res, cl_full, short, observed_to = 15)$tables$total
```

`bag_reserve` *Fit bootstrap regression trees and complete reserve replicates*

Description

Bootstrap aggregating of the trees of `reserve_tree()`. For every replicate `b = 1, ..., B` and every development year `l` (its `period`), a bootstrap sample of the training claims of that development year is drawn (with replacement, duplicates kept) and a regression tree is grown on it (by default unpruned, `cp = 0`, no cross-validation). Replicate `b` then completes the payment histories of all reported claims with its `L` trees, one per development year, recursively one development year ahead. The `B` completed replicates are the output: the point estimate is the bagged mean of the completed replicate predictions, and their spread describes the variation under this resampling of the predictions of one set of bootstrap trees (unpruned by default, unlike the pruned trees of `reserve_tree()`). It is not a predictive distribution of the future payments: the numbers of unreported claims are projected once and held fixed, and with the default `completion = "mean"` no payment noise is simulated. With `completion = "draw"` each replicate is completed once by payments simulated from its own trees, so its spread adds process variability to this resampling variation. It is still not a predictive distribution: the estimation part is the variation of one set of (unpruned) bootstrap trees, a noisier estimator than the bagged mean that is reported; the bootstrap samples are drawn independently for each development year, not once per claim, so they do not reproduce how the errors of successive trees are linked through the same claims; and the IBNR counts stay fixed. The prediction records which case applies (`variability`), and its `print()`, `summary()`, `plot()` and `backtest()` output says so.

Usage

```
bag_reserve(
  panel,
  B = 100L,
  seed = NULL,
  ncores = 1L,
  control = rpart::rpart.control(cp = 0, xval = 0),
  prune = c("none", "min_xerror"),
  oob = TRUE,
  keep_trees = FALSE,
  batch_size = NULL,
  backend = c("auto", "fork", "psock", "serial"),
  claim_level = TRUE,
  learner = NULL,
  completion = c("mean", "draw")
)
```

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

Arguments

panel	A claims_panel with L >= 1.
B	Number of bootstrap replicates (a positive integer; default 100).
seed	Integer seed of the stream allocation. NULL draws one from the current generator (advancing it, so that repeated calls differ) and records it in the result.
ncores	Number of worker processes (forked on Unix, a socket cluster elsewhere), a positive integer; default 1, which runs serially. Values above the number of cores detected are reduced to it, with a message; when parallel::detectCores() cannot determine that number (it returns NA in some containers), the value is used as given. The value used is recorded (ncores of the result) and printed.
control	rpart::rpart.control() for the bootstrap trees (default cp = 0, xval = 0).
prune	"none" (default) or "min_xerror" (needs cross-validation, xval >= 2 in control, and is an error otherwise). Ignored, and not checked, with a user-supplied learner.
oob	Compute the out-of-bag diagnostics (default TRUE; with FALSE the oob element of the result is NULL and oob_score() fails).
keep_trees	Keep the B * L fitted trees (and the bootstrap indices) in the result (default FALSE), which allows predict(..., newdata =) (for completion = "mean" only) and inspection at a large memory cost.
batch_size	Replicates processed per worker task before OOB votes are returned; NULL (default) uses max(1, ceiling(B / ncores / 2)). The batching does not change the completed replicates.
backend	"auto" forks workers on Unix and uses a socket cluster elsewhere; "fork", "psock" and "serial" force one backend. Results are the same for all of them (up to floating-point summation order in the OOB sums). Socket workers load the installed package; a version different from the one in use is an error. They are fresh R sessions, so for a user-supplied learner they are given the variables of the global environment that its functions refer to (found with codetools::findGlobals(), also inside the functions among them) and the attached packages those functions use; an object a learner reaches otherwise (through get(), a character string, a formula's environment) must be created inside its functions or carried in their environment, as two_part_learner() does, and a failure on the workers says so.
claim_level	Return the claim-level reserve: the mean over the replicates of the completed claim-by-development-year payment matrices (default TRUE; component claim_level of the prediction). It costs one matrix of the panel's size per worker.
learner	NULL (default) grows **rpart** trees as described above. Otherwise a list with functions fit(formula, data), returning a model for one development year from its bootstrap sample, and predict(model, newdata), returning numeric predictions; the bootstrap, the recursion, the out-of-bag diagnostic and the random-number streams are those of the package. control and prune apply only to

the default learner (given with a `learner`, they are ignored with a warning and not validated; the fit records `prune = "none"`), as in `reserve_tree()`. Every prediction of a learner (out-of-bag votes, plugged-in and simulated completion) is checked: a numeric vector with one finite value per row of `newdata`; see `reserve_tree()` and `two_part_learner()`.

`completion` `"mean"` (default, the source article's method) completes each replicate by plugging in its predicted means; `"draw"` simulates each replicate's completion once, drawing every future payment from the training responses of the claim's leaf in that replicate's tree (for a user-supplied learner, from its `simulate()`), on a separate random-number substream, so the trees are those of `"mean"`. The replicate spread then adds the process variability of one simulated completion to the resampling variation of one set of bootstrap trees (see Details for what it still leaves out). For bagged fits the completion is chosen here, when fitting; `predict.bag_reserve()` has no `completion` argument (unlike `predict.reserve_tree()`) and refuses one that differs from the fit's; a fit with `"draw"` cannot predict `newdata` (the leaf responses of its trees are not kept). Plugging in means is biased for payments beyond next year when the occurrence of a payment depends on the payment history: in the correctly specified simulation of the JSS manuscript (Section 4.6) it overstated the total reserve of the reported claims by up to 54% for the bagged mean, a bias that does not vanish as the portfolio grows. Simulated completion removes this mechanism and most of the bias; what remained there was at most 1.7% for the bagged mean, whose unpruned bootstrap trees also overpredict next year's payments by about 1%. Next year's payments are the same in expectation under both completions. **For totals beyond next year we recommend `"draw"`.**

`object` A `bag_reserve` fit (for `summary`).

`...` For `summary`, passed on to `summary.reserve_prediction()` (e.g. `probs`).

Details

Completion. The default `completion = "mean"` is the source article's method, kept so that the package reproduces it. For totals beyond next year we recommend `completion = "draw"`: plugging in means is biased for the later years when the occurrence of a payment depends on the payment history (see the argument `completion`).

The default `B = 100` suffices for the bagged mean: its Monte Carlo standard error is the replicate standard deviation divided by `sqrt(B)`, small against the errors back-tests measure. Extreme replicate quantiles and complete out-of-bag coverage need more; the JSS manuscript uses `B = 1000`.

A factor level absent from the training claims of a development year gives a warning (see `predict.reserve_tree()`). A rare level can also be missing from the bootstrap sample of one replicate while present in the training claims; that replicate's tree then treats it as missing without a warning, which is part of the resampling variation.

The out-of-bag (OOB) score of development year `l` uses, for each training claim, the trees of the replicates in which it was not sampled (its out-of-bag predictions, or "votes"), predicts its observed `X_{l+1}` from its observed history, and compares: `OOB_error = mean((y - OOB)^2)` over the covered claims and `OOB_score = 1 - sum_covered (y - OOB)^2 / sum_all (y - mean(y))^2`, the denominator running over all training claims of the development year (Janoušek and Pešta 2025). It

is a diagnostic of the one-year regressions given observed histories, not a validation of the recursive forecast.

Random numbers come from `"L'Ecuyer-CMRG"` streams allocated to (replicate, development year) tasks, so the completed replicates do not depend on `ncores`, on the batching or on the scheduling of the workers under the same software; the out-of-bag sums are accumulated batch by batch and can differ in their last floating-point digits. The caller's generator kind and state are restored on exit (after the one draw that makes the seed when `seed = NULL`). Replicates are streamed: each worker fits the trees of a replicate, records the OOB votes, aggregates the predicted future cells and discards the trees unless `keep_trees = TRUE`.

Value

An object of class `c("bag_reserve", "reserve_fit")`: a list with the cached reserve_prediction of the fitted portfolio (`prediction`), the OOB table (`oob`, `NULL` when `oob = FALSE`), the trees and bootstrap indices (`trees`, `inbag`; `NULL` unless `keep_trees = TRUE`; `trees[[b]][[l + 1]]` is the tree of replicate `b` for development year `l`), `B`, `L`, `t`, `n_train`, `features`, `feature_levels`, the `control`, `prune`, `keep_trees`, `ncores`, `backend`, `elapsed_secs`, the RNG information (`rng`), the package versions (`versions`) and the call. Five replicates, as in the example, illustrate the interface; they are far too few for stable quantiles.

See Also

`predict.bag_reserve()`, `oob_score()`, `ibnr_reserve()`

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
# a small subset for a quick example: the claims of one type
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
bag <- bag_reserve(panel, B = 5, seed = 2001)
bag
oob_score(bag)
calendar_year_payments(predict(bag), "reported")
# simulated completion, recommended for totals beyond next year
bag_draw <- bag_reserve(panel, B = 5, seed = 2001, completion = "draw")
calendar_year_payments(predict(bag_draw), "reported")[c(1, 11), ]
# another learner for the one-year regressions, here a linear model
lm_learner <- list(fit = function(formula, data) lm(formula, data = data),
                   predict = function(model, newdata) predict(model, newdata = newdata))
bag_lm <- bag_reserve(panel, B = 5, seed = 2001, learner = lm_learner)
calendar_year_payments(predict(bag_lm), "reported")[c(1, 11), ]
```

`calendar_year_payments`

Payments by future calendar year

Description

Sums the predicted future cells by calendar year (`acc_year + dev`), i.e. along the diagonals of the run-off triangle: the first row is next year's payments, the last row the total of all future payments through development year `L`.

Usage

```
calendar_year_payments(x, ...)

## S3 method for class 'reserve_prediction'
calendar_year_payments(x, component = c("total", "reported", "ibnr"), ...)

## Default S3 method:
calendar_year_payments(x, ...)
```

Arguments

`x`	A reserve_prediction (or a chain_ladder object, see `calendar_year_payments.chain_ladder()`).
`...`	Not used; an argument given here is an error (for a chain_ladder object, which has a single component, any argument is).
`component`	Component to aggregate; the default `"total"` sums reported-claim and IBNR payments when an IBNR component is attached and otherwise uses the reported-claim payments; `"ibnr"` requires `ibnr_reserve()` to have been applied.

Value

A matrix with one row per future calendar year (row names are the years) plus a `total` row, one column per replicate (the column names of the prediction), and the attribute `"component"` holding the component actually aggregated.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
res <- ibnr_reserve(predict(reserve_tree(panel, seed = 1)), claim_counts(panel))
cbind(reported = calendar_year_payments(res, "reported")[, 1],
      ibnr = calendar_year_payments(res, "ibnr")[, 1],
      total = calendar_year_payments(res, "total")[, 1])
```

`calendar_year_payments.chain_ladder`
Future payments of a chain-ladder projection by calendar year

---

Description

`calendar_year_payments()` on a chain_ladder object sums the projected incremental cells after the valuation date by calendar year (accident year plus development year).

Usage

```
## S3 method for class 'chain_ladder'
calendar_year_payments(x, ...)
```

Arguments

`x`	A chain_ladder object whose columns are development years starting at 0 and whose row names are the accident years.
`...`	Not used; any argument given here is an error.

Value

Named vector of projected payments per future calendar year and a `total`.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11)
calendar_year_payments(chain_ladder(as_triangle(panel)))
```

`chain_ladder` *Chain-ladder projection of a run-off triangle*

Description

Completes a run-off triangle (accident years in rows, development years in columns, `NA` for the cells after the valuation date) with development factors estimated from the observed cells. Two projections are available:

`"chain_ladder"` the standard chain ladder: factors are ratios of successive column sums of the *cumulative* triangle over the accident years observed in both columns, and each row is projected from its latest observed cumulative value. Point predictions only, no tail, no uncertainty measure.

`"incremental_ratio"` the projection used for the baselines of Janoušek and Pešta (2025), the source article: ratios of successive column sums of the *incremental* triangle, applied multiplicatively to the incremental cells. Provided to apply the source article's convention, not as a reproduction of its results; a factor with a zero denominator is `NaN` (zero numerator) or `Inf` (positive numerator), as in the source article's code, and the projected cells that are not finite are listed in `undefined`.

Usage

```
chain_ladder(
  triangle,
  input = NULL,
  projection = c("chain_ladder", "incremental_ratio")
)
```

Arguments

`triangle`	Numeric matrix with `NA` in the unobserved cells; each row must be observed in a leading block of columns. Typically `as_triangle()` output.
`input`	Whether `triangle` holds `"incremental"` or `"cumulative"` amounts. `NULL` (default) takes it from the triangle's `"cumulative"` attribute, which `as_triangle()` sets, and assumes incremental amounts when the attribute is absent. An explicit value that contradicts the attribute is an error.
`projection`	`"chain_ladder"` (default) or `"incremental_ratio"`, as described above.

Value

An object of class `"chain_ladder"`: a list with `factors` (one per development year `1, ..., L`, with the number of rows used), the completed `incremental` and `cumulative` triangles, `latest`, `ultimate` and `outstanding` per accident year, `outstanding_total`, the observation mask and the method.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
panel <- claims_panel(synthetic_claims, synthetic_payments, t = 12, L = 11,
                      features = c("Type", "Age", "AccWeekday"))
cl <- chain_ladder(as_triangle(panel))
cl
round(cl$outstanding)
```

`claim_reserves` *Reserves of the individual reported claims*

Description

Sums the completed payments of each reported claim over its development years after the valuation date `t` (cells with `acc_year + dev > t`), up to development year `L`. For a bagged prediction these are the bagged means; they add up to the mean reported-claim total of `calendar_year_payments()`.

Usage

```
claim_reserves(x)
```

Arguments

x	A reserve_prediction that holds a claim-level matrix (a single-tree prediction, or a bagged one with `claim_level = TRUE`).

Details

The per-claim values are consistent with the aggregate prediction, but they are not a reliable allocation of the reserve between the claims, in particular between open and closed claims. Following the source article, every reported claim is completed, including claims already closed, and the trees see static features and the payment history only, not the settlement status: on the shipped portfolio a third of the predicted reported-claim payments falls on claims closed at the valuation date, and the open claims are underpredicted by about 30% (supplementary material of the JSS manuscript, Section S5). Use the values to aggregate over groups of claims with care, and not as case reserves; allocating a reserve to the open claims needs a model with a settlement layer, such as **hirem**.

Value

A data frame with one row per reported claim: `id`, `acc_year` and `reserve`.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
cr <- claim_reserves(predict(reserve_tree(panel, seed = 1)))
summary(cr$reserve)
```

`claims_panel` *Build a claim-by-development panel from individual claims and payments*

Description

Turns two long tables, one row per claim and one row per payment, into the observation-masked panel on which the reserving method works: one row per claim reported by the valuation date `t`, static features, the reporting delay, and the incremental payments in development years `0, ..., L`. Cells that lie after the valuation date are `NA`; observed cells without payments are `0`.

Usage

```
claims_panel(
  claims,
  payments,
  t,
  L,
  id = c(claims = "Id", payments = "Id"),
  acc_year = "AccYear",
```

```
  rep_year = "RepYear",
  rep_delay = NULL,
  event_year = "EventYear",
  paid = "Paid",
  features = character(),
  accident_years = NULL,
  na_features = c("fail", "surrogate"),
  future_payments = c("error", "drop"),
  beyond_L = c("error", "drop")
)

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

## S3 method for class 'claims_panel'
as.data.frame(x, ...)
```

Arguments

`claims`	Data frame with one row per claim.
`payments`	Data frame with one row per payment (several payments of one claim in the same year are summed).
`t`	Valuation (reserving) date, an integer year on the same grid as the accident and event years. Claims reported after `t` and payments after `t` are not part of the panel. The grid need not be yearly: any integer time unit works (quarters or months numbered consecutively, say), and "development year" then means development period.
`L`	Maximal development year; payments have columns `X0, ..., XL`. It is not inferred from the data. The fitting functions need training claims for the model of every development year up to `L - 1`, i.e. claims with `acc_year + L <= t`; when the latest accident year is `t`, `L` is therefore at most the number of accident years less one (`reserve_tree()` and `bag_reserve()` stop otherwise, naming the development years without training claims).
`id`	Name of the claim identifier column, either one name used in both tables or `c(claims = , payments = )`.
`acc_year`	Name of the accident-year column in `claims`.
`rep_year`	Name of the reporting-year column in `claims`. Ignored if `rep_delay` is given and absent from `claims`; if both are present they must agree.
`rep_delay`	Optional name of a reporting-delay column (`rep_year - acc_year`).
`event_year`	Name of the payment-year column in `payments`.
`paid`	Name of the amount column in `payments`.
`features`	Character vector of static feature columns of `claims` to carry into the panel as predictors. Character columns become factors with levels in sorted order; factors keep their levels. The reporting delay is always a predictor and need not be listed. No feature is ever picked up automatically.

`accident_years` Optional integer vector of accident years defining the rows of the aggregate triangles. The default runs from the earliest to the latest accident year with a reported claim, gaps included. Accident years after the latest one are not added, because a portfolio in run-off has none; when that latest year is before `t`, a warning says so, since otherwise the most recent accident years, whose claims may all be reported later (a long-tail line, a new line of business), would silently have neither a row nor an IBNR component. To include them, pass them, e.g. `accident_years = first:t`: their rows then exist (empty), but the chain ladder projects no IBNR claims from an empty row, so `ibnr_reserve()` warns again and their IBNR payments must be estimated outside the package (see `ibnr_reserve()`). For a portfolio that wrote no business in those years, pass `accident_years = first:last` to state this and silence the warning.

`na_features` `"fail"` (default) refuses missing feature values; `"surrogate"` keeps them for **rpart**'s handling of missing values: a claim whose value of the split feature is missing is sent down by the first surrogate split it has a value for, and by the majority direction of the split when no surrogate resolves it (with rpart's defaults `usesurrogate = 2`, `maxsurrogate = 5`; a user-supplied `control` changes this). Claims with a missing value do not count when a split on that feature is chosen.

`future_payments` `"error"` (default) refuses payments dated after `t`; `"drop"` discards them and records how many and how much.

`beyond_L` `"error"` (default) refuses reported claims with a reporting delay above `L` and observed payments in development years after `L`, which are outside the model; `"drop"` discards them (such claims together with all their payments) and records how many and how much (`n_claims_beyond_L_dropped`, `n_payments_beyond_L_dropped`, `amount_beyond_L_dropped`), for a long history whose oldest accident years have developed beyond `L`. The identifiers of the claims set aside are kept (`beyond_L_ids`) and passed on to the predictions made from the panel, so that `backtest()` accepts the complete claims and payments tables: it does not count these claims as missing from the prediction, and their payments after `t`, all beyond development year `L`, are part of the excluded tail it reports.

`object` A `claims_panel` (for `summary`).

`...` For `summary`, not used: an argument given there is an error. Ignored by `as.data.frame`.

`x` A `claims_panel` (for `as.data.frame`).

Details

The constructor enforces the valuation date on both tables: claims are kept only if reported by `t` (reporting year `<= t`), payments only if made by `t`. Claims reported after `t` are dropped, and so are claims whose accident year is after `t` (they cannot be reported by `t`); both are counted in `n_unreported_dropped`. A claim without a reporting year (or delay) is treated as not reported by `t` and dropped, with a warning, since a missing date is more often a data defect than a late report. It preserves recoveries (negative payments), keeps reported claims without any payment, and refuses input it cannot interpret: duplicated or missing identifiers, payments on claims absent from `claims`, payments before the accident or reporting year, reporting delays or observed payments beyond `L` unless `beyond_L = "drop"` is given, and payments after `t` unless `future_payments = "drop"` is given explicitly (the dropped numbers and amounts are recorded in the diagnostics). Payments

dated before the recorded reporting year (advance payments, or reporting dates corrected after a first payment) are refused: move such a payment to the reporting year, or correct the reporting year, before calling the constructor.

Value

An object of class `"claims_panel"`: a list with elements

claims data frame with `id`, `acc_year`, `rep_delay` and the feature columns

payments numeric matrix, one row per claim, columns `X0, ..., XL`; `NA` after `t`

observed logical matrix of the same shape, `TRUE` where `acc_year + l <= t`

t, L, accident_years, I valuation date, horizon, accident years, their number

features, feature_levels the feature names and the factor levels

mapping, diagnostics, call column mapping (carried into the predictions, where `backtest()` uses it as its default column names), construction counts, the call

beyond_L_ids identifiers of the reported claims set aside by `beyond_L = "drop"` (empty otherwise)

See Also

`as_triangle()`, `claim_counts()`, `reserve_tree()`, `bag_reserve()`

Examples

```
claims <- data.frame(Id = c("a", "b", "c"), AccYear = c(1, 1, 2), RepYear = c(1, 2, 2),
                     Type = c("x", "y", "x"))
payments <- data.frame(Id = c("a", "a", "a", "b"), EventYear = c(1, 1, 2, 2),
                       Paid = c(10, 5, -2, 7))
panel <- claims_panel(claims, payments, t = 3, L = 2, features = "Type")
panel
panel$payments
as_triangle(panel)
```

`ibnr_reserve` *Payments of the claims incurred but not reported*

Description

Adds the IBNR component to a reserve_prediction, following Theorem 1 and Section 3.6 of Janoušek and Pešta (2025): the numbers of claims still to be reported are projected from the reporting-count triangle with a chain-ladder-type method and held fixed across replicates; the payment of such a claim in development year `l` is estimated by the pooled average payment of the reported claims in development year `l`,

$$\hat{\mu}_l = \frac{\sum_i \hat{R}_{i,l}}{\sum_i \sum_{d \le l,\ i+d \le t} N_{i,d}},$$

where $\hat{R}_{i,l}$ are the observed plus predicted payments of the reported claims (a replicate of the prediction) and $N_{i,d}$ the observed numbers of claims of accident year i reported with delay d. The IBNR payment in cell (i, l) is the projected number of claims of accident year i reported after the valuation date with delay $\leq l$, times $\hat{\mu}_l$.

Usage

```
ibnr_reserve(
  pred,
  counts = NULL,
  projection = c("chain_ladder", "incremental_ratio"),
  round = FALSE
)
```

Arguments

`pred`	A reserve_prediction (from `predict()` on a fit). An existing IBNR component is replaced.
`counts`	The *incremental* reporting-count triangle of the panel the prediction was made for. `NULL` (default) uses the triangle stored in the prediction (`claim_counts()` of that panel). A supplied triangle must agree with the stored one in its observed cells (it can only differ in how it is passed, e.g. without attributes); for a prediction made by an older version of the package, which stores no counts, it must have the same dimensions, observation mask, accident years and valuation date as the prediction, with finite non-negative integer observed cells.
`projection`	Count projection, `"chain_ladder"` (default) or `"incremental_ratio"`; see `chain_ladder()`.
`round`	Round the projected numbers of claims to integers (default `FALSE`).

Details

The estimator assumes that the pooled payment per development year of the reported claims applies to the claims reported late; a payment pattern that depends on the reporting delay or a business mix that differs between early and late reporters invalidates it (see the JSS manuscript, Section 2.4).

An accident year without any reported claim at the valuation date has an all-zero row in the count triangle, and the chain ladder projects no late-reported claims from it: such years get no IBNR payments, with a warning. The function has no argument for an external count projection; when this matters, estimate the IBNR payments of such years outside the package (for example, as in the formula above, externally projected numbers of late claims with delay `<= 1` times the pooled mean payments `ibnr_details$mu`), or leave those accident years out of the panel (argument `accident_years` of `claims_panel()`).

In the last development years $\hat{\mu}_l$ rests on little information: only the oldest accident years are observed there, and the other accident years contribute the predictions of the late trees, which are trained on those few accident years. A few recoveries can then make $\hat{\mu}_l$ negative in every replicate (on the claims of type 2 of the shipped portfolio, for example, $\hat{\mu}_L$ is negative, and so is the IBNR payment of calendar year 23); inspect `ibnr_details$mu`, and `ibnr_details$divisor`, the reported claims behind it, before relying on the late cells.

The projected numbers of claims are the same in every replicate, so the spread of the IBNR component over the replicates comes from the pooled mean payments $\hat{\mu}_l$ only; it contains no uncertainty of the count projection.

With `projection = "incremental_ratio"` and `round = TRUE` the count projection uses the source article's incremental-ratio factors and rounding convention: undefined (`NA`/`NaN`) projected counts become zero, an infinite projected count is an error, and the completed count matrix is rounded once. This applies the source article's computational conventions; it does not reproduce its numerical results on the shipped portfolio. The default is the standard chain ladder on cumulative counts without rounding.

Value

`pred` with the components `ibnr` (matrix, future cells by replicates) and `ibnr_details` added: the projected count triangle (`counts_projected`), the projected numbers of late-reported claims per cell (`counts_future_cumulative`), the development `factors`, the `divisor` (observed reported-claim counts per development year), `mu` (replicates by development years), and the projection settings.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
# a small subset for a quick example: the claims of one type
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
tree <- reserve_tree(panel, seed = 1)
res <- ibnr_reserve(predict(tree), claim_counts(panel))
calendar_year_payments(res, "total")
```

---

oob_score *Out-of-bag diagnostics of a bagged fit*

Description

Extracts the out-of-bag errors, scores and coverage per development year from a bagged fit. For development year `l`, every training claim that was left out of the bootstrap sample of at least one replicate (a *covered* claim) receives the average prediction of those replicates' trees, computed from its observed history. `oob_error` is the mean squared error over the covered claims; `oob_score` is one minus the sum of squared errors over the covered claims divided by the total sum of squares of the response over *all* training claims of the development year (the definition of Janoušek and Pešta 2025). With `B` replicates a claim is covered with probability `1 - (1 - 1/n)^n)^B`, which is also the expected coverage: about 0.99 at `B = 10`, so that coverage is essentially complete for `B` in the hundreds but not for very small `B`; the observed `coverage = n_scored / n_train` reports it, and the score is `NA` when no claim is covered or the response has zero variance. Because its numerator runs over the covered claims only, the score is biased upwards when the coverage is below one (small `B`); `oob_score_covered` is the ordinary coefficient of determination over the covered claims, which does not have that bias. A negative score means that the out-of-bag predictions are worse than

the mean of the response, which is common for unpruned bootstrap trees (`cp = 0`) on noisy late development years; a larger `minbucket` or a positive `cp` in `control` smooths them. The score is a diagnostic of the one-year regressions, not a validation of the recursive multi-year forecast or of the IBNR component.

Usage

```
oob_score(object)
```

Arguments

`object`	A bag_reserve fit created with `oob = TRUE` (otherwise an error).

Value

A data frame with one row per development year and the columns `period` (the development-year index), `n_train`, `n_scored` (claims with at least one out-of-bag vote), `coverage`, `mean_votes` (mean number of votes among the covered claims), `oob_error`, `oob_score`, `response_var` and `oob_score_covered`.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
bag <- bag_reserve(panel, B = 5, seed = 2026)
oob_score(bag)[, c("period", "n_train", "coverage", "oob_score")]
```

---

plot.bag_reserve *Plot the out-of-bag scores of a bagged fit*

Description

Plot the out-of-bag scores of a bagged fit

Usage

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

Arguments

`x`	A bag_reserve fit.
`...`	Passed to `graphics::plot()`.

Value

`x`, invisibly.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
plot(bag_reserve(panel, B = 5, seed = 2026))
```

`plot.reserve_prediction` *Plot a reserve prediction*

Description

Base-graphics displays of a reserve_prediction:

`"calendar"` predicted payments per future calendar year: the bagged mean (or the single-tree prediction) as a line, the replicate quantile band shaded when `B > 1`, and the realised payments as points when `truth` is given.

`"next_year"`, `"ultimate"` the replicate predictions of next year's payments, or of the total through development year `L`: a histogram with a rug for `B >= 20`, individual points otherwise; vertical lines mark the bagged mean (solid black), the replicate quantiles (dashed grey), the `truth` (dot-dashed blue) and any `reference` values (dotted green, e.g. a chain-ladder estimate).

Bands and quantiles are empirical quantiles over the columns of the prediction and are labelled by what these are: the bootstrap replicates of a bagged fit (resampling variation, with simulated payments as well for `completion = "draw"`) or the simulated completions of a single-tree fit (process variability given the one fitted model); see the `variability` of `predict.reserve_tree()`. They are neither prediction intervals nor confidence intervals for the mean.

Usage

```
## S3 method for class 'reserve_prediction'
plot(
  x,
  what = c("calendar", "next_year", "ultimate"),
  component = c("total", "reported", "ibnr"),
  probs = c(0.025, 0.975),
  truth = NULL,
  reference = NULL,
  main = NULL,
  ...,
  type = 7L
)
```

Arguments

x	A reserve_prediction.
what	Which display: "calendar" (default), "next_year" or "ultimate".
component	"total" (default; falls back to "reported" without an IBNR component), "reported" or "ibnr".
probs	Two increasing probabilities in [0, 1] for the quantile band or lines (default c(0.025, 0.975)).
truth	Optional realised payments (default NULL, none drawn): for "calendar" a vector named by calendar year, e.g. setNames(bt$tables$total$truth, bt$tables$total$calendar) from a backtest(), or an unnamed vector with one value per future calendar year; for the others a single number.
reference	Optional named numeric vector of reference estimates drawn as dotted vertical lines ("next_year", "ultimate"); default NULL.
main	Title; NULL (default) builds one from the display and the estimator.
...	Further graphical arguments passed to the underlying plotting function (graphics::plot() for "calendar", graphics::hist() or graphics::stripchart() otherwise); they override the defaults (e.g. ylab, ylim, xlab, col).
type	Quantile algorithm for the band and the quantile lines, as in stats::quantile() (an integer from 1 to 9; default 7, as in summary.reserve_prediction() and backtest()); it must be named.

Value

x, invisibly.

Examples

```
data("synthetic_claims", "synthetic_payments", "synthetic_claims_full",
     "synthetic_payments_full", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
bag <- bag_reserve(panel, B = 20, seed = 2026)
res <- ibnr_reserve(predict(bag), claim_counts(panel))
cl_full <- synthetic_claims_full[synthetic_claims_full$Type == "3", ]
bt <- backtest(res, cl_full, synthetic_payments_full[synthetic_payments_full$Id %in% cl_full$Id, ])
plot(res, "calendar", truth = setNames(bt$tables$total$truth, bt$tables$total$calendar))
plot(res, "next_year", truth = bt$tables$total$truth[1])
plot(bag)
```

plot.reserve_tree *Plot the tree of one development year of a fit*

Description

Plot the tree of one development year of a fit

Usage

```
## S3 method for class 'reserve_tree'
plot(x, period = 0L, ...)
```

Arguments

x	A reserve_tree fit.
period	Development year l (0, ..., L - 1) whose tree is drawn.
...	Passed to rpart::plot.rpart().

Value

x, invisibly.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
plot(reserve_tree(panel, seed = 1), period = 0)
```

predict.bag_reserve *Predictions of a bagged fit*

Description

Returns the completed replicate predictions, or their bagged mean, for the fitted portfolio or for compatible new data. Without newdata, the reserve_prediction cached at fitting time is returned: one column per bootstrap replicate (type = "replicates") or their mean (type = "mean", which averages the *completed* replicates, never the trees development year by development year). With newdata, the stored trees complete another claims_panel; this needs keep_trees = TRUE and completion = "mean" at fitting time (a fit with completion = "draw" does not keep the leaf responses its simulation draws from, so it predicts only the fitted portfolio).

Usage

```
## S3 method for class 'bag_reserve'
predict(object, newdata = NULL, type = c("replicates", "mean"), ...)
```

Arguments

object	A bag_reserve fit.
newdata	NULL (default) completes the panel the model was fitted on. Otherwise another claims_panel with the same features in the same order, the same L, and factor levels among those seen at fitting time (a compatible subset is re-levelled; levels unknown to the fit are an error). It may have another valuation date t than the fitted panel: the trees then complete its cells after its own t, and the prediction is labelled by that t (its first calendar year is t + 1 of newdata). A panel valued before the fit's valuation date that shares claims with the fitted panel gives a warning: the trees were trained on those claims' payments after its valuation date, so a back-test of the prediction would use information that was not available then; refit at the earlier date instead (Section 4.3 of the JSS manuscript). A factor level that the fit knows but that is absent from the training claims of the period in which a claim is completed gives a warning, here as when the fitted panel is completed: at each split on that feature the trees treat it as missing (surrogate splits, else the majority direction).
type	"replicates" returns every replicate; "mean" averages the completed replicates (for a single-tree fit the two coincide).
...	Only completion, equal to the fit's (object$completion); anything else is an error.

Details

The completion (plugged-in means or simulated payments) of a bagged fit is chosen when fitting, with bag_reserve(completion =), not here: completion is accepted only when it restates the fit's own choice, and any other argument (R, seed, a misspelt type) is an error, so that no call silently returns another estimator than the one asked for.

Value

A reserve_prediction as documented in predict.reserve_tree(), with B columns of predicted reported-claim payments and, unless the fit was made with claim_level = FALSE, the claim-level matrix averaged over the replicates (claim_level; its future cells sum to the bagged mean of the reported component). With type = "mean" the object holds one column named "mean", B = 1 and the estimator label "bagging (mean of replicates)"; the replicate distribution is then no longer available.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
bag <- bag_reserve(panel, B = 5, seed = 2026, keep_trees = TRUE)
pred <- predict(bag)                      # five columns, one per replicate
pred_mean <- predict(bag, type = "mean")  # one column, the bagged mean
calendar_year_payments(pred)[c(1, nrow(calendar_year_payments(pred))), ]
# the same trees applied to another panel of the same layout (the claims of type 4)
```

```
cl4 <- synthetic_claims[synthetic_claims$Type == "4", ]
panel4 <- claims_panel(cl4, synthetic_payments[synthetic_payments$Id %in% cl4$Id, ],
                       t = 12, L = 11, features = c("Age", "AccWeekday"))
predict(bag, newdata = panel4)
```

`predict.reserve_tree` *Predicted future payments of the reported claims*

Description

`predict()` on a reserve_tree completes the payment histories of all claims reported by the valuation date, recursively one development year ahead: the tree for development year `l` predicts `X_{l+1}` from the features, the reporting delay and `X_0, ..., X_l`, where the cells after the valuation date among `X_0, ..., X_l` are the predictions of the earlier trees. Following the source article, every reported claim is completed, including claims recorded as closed; the claim status is not a covariate unless it is named in `features`. Do not name a status recorded at the valuation date as a feature without care: the panel holds it as a static feature, but in the earlier development years from which the trees learn it was not yet known (a claim closed at `t` was open in the years in which it paid), so the trees learn from information that the claims did not have at the time (see the JSS manuscript on open and closed claims).

Usage

```
## S3 method for class 'reserve_tree'
predict(
  object,
  newdata = NULL,
  type = c("replicates", "mean"),
  completion = c("mean", "draw"),
  R = 100L,
  seed = NULL,
  ...
)
```

Arguments

`object`	A fitted reserve_tree.
`newdata`	`NULL` (default) completes the panel the model was fitted on. Otherwise another claims_panel with the same features in the same order, the same `L`, and factor levels among those seen at fitting time (a compatible subset is re-levelled; levels unknown to the fit are an error). It may have another valuation date `t` than the fitted panel: the trees then complete its cells after its own `t`, and the prediction is labelled by that `t` (its first calendar year is `t + 1` of `newdata`). A panel valued before the fit's valuation date that shares claims with the fitted panel gives a warning: the trees were trained on those claims' payments after its valuation date, so a back-test of the prediction would use information that was not available then; refit at the earlier date instead (Section 4.3 of the JSS manuscript).

A factor level that the fit knows but that is absent from the training claims of the period in which a claim is completed gives a warning, here as when the fitted panel is completed: at each split on that feature the trees treat it as missing (surrogate splits, else the majority direction).

`type` `"replicates"` returns every replicate; `"mean"` averages the completed replicates (for a single-tree fit the two coincide).

`completion` `"mean"` (default, the source article's method) plugs the predicted mean of each development year into the payment history used for the next, as described above. `"draw"` simulates instead: each future payment is a training response drawn from the claim's leaf (for a user-supplied learner, from its `simulate(model, newdata)` function). The `R` simulated completions are the columns of the result (process variability of the fitted trees, no estimation variability); `type = "mean"` averages them. Because most later-year payments are exactly zero, plugging in means can move claims across splits that separate paying from non-paying claims; simulation avoids this. (For a bagged fit the completion is chosen in `bag_reserve()`.) Plugging in means is biased for payments beyond next year when the occurrence of a payment depends on the payment history: in the correctly specified simulation of the JSS manuscript (Section 4.6) it overstated the total reserve of the reported claims by up to 62% for the single tree, a bias that does not vanish as the portfolio grows. Simulated completion removes this mechanism and most of the bias (at most 0.8% remained there for the single tree; the fitted model's own errors remain). Next year's payments are the same in expectation. **For totals beyond next year we recommend `"draw"`.**

`R` Number of simulated completions for `completion = "draw"` (default 100).

`seed` Seed for the simulated completions, set with R's default generator kinds as in `reserve_tree()` (the caller's kind and state are restored on exit); `NULL` (default) uses and advances the current random-number state. `R` and `seed` belong to `completion = "draw"`: given with `completion = "mean"`, they are an error.

`...` Not used; any argument given here (a misspelt one, say) is an error.

Details

The result is a `reserve_prediction`: the observed incremental triangle, the coordinates of the cells after the valuation date, and one column per replicate with the predicted reported-claim payments in those cells. A single-tree fit gives one prediction column; `bag_reserve()` gives `B`.

Value

An object of class `"reserve_prediction"`, a list with

observed incremental payments by accident year and development year, `NA` after the valuation date

cells data frame of the future cells: `acc_year`, `dev`, `calendar` (`acc_year + dev`), `index` (position in the triangle, column-major)

reported matrix, one row per future cell and one column per replicate, of predicted payments of the reported claims

ibnr, ibnr_details `NULL` until `ibnr_reserve()` adds the IBNR component and its details (projected counts, factors, pooled mean payments)

counts the incremental reporting-count triangle of the panel (`claim_counts()`), used by `ibnr_reserve()` by default

claim_level the completed claim-level payment matrix, one row per reported claim (columns `X0, ..., XL`): for a single-tree fit its completion, for a bagged fit the mean of the completed replicates; observed cells keep their observed values. Its future cells add up to the reported-claim prediction, but they are not a reliable allocation of that reserve between the claims: every reported claim is completed, open or closed, and the trees do not know the settlement status, so on the shipped portfolio a third of the reserve falls on claims already closed while the open claims are underpredicted (supplementary material of the JSS manuscript, Section S5). See `claim_reserves()`; for case-level allocation a model with a settlement layer, such as **hirem**, is needed.

claim_ids, replicate_ids identifiers of the completed claims and of the columns

claim_acc_years, claim_rep_delays the accident year and reporting delay of every completed claim, which `backtest()` checks against its truth tables

mapping, beyond_L_ids the column names of the panel's input tables, which `backtest()` uses as its defaults, and the claims the panel set aside with `beyond_L = "drop"`

t, L, accident_years, B, estimator, call metadata; `estimator` is `"tree"`, `"tree (R simulated completions)"`, `"bagging"` or `"bagging (simulated completion)"`, with `" (mean of replicates)"` (bagging) or `"tree (mean of R simulated completions)"` after `type = "mean"`

completion, variability the completion mode (`"mean"` or `"draw"`) and the sources of variability among the columns: none for a single-tree fit with `completion = "mean"`; `"simulated payments"` for the `R` simulated completions of one fitted model (process variability given the fitted model, no estimation variability); `"bootstrap resampling"` for the replicates of `bag_reserve()`; both for a bagged fit with `completion = "draw"`, whose replicate `b` is one completion simulated from the `b`-th set of bootstrap trees (its estimation part is the variation of one set of unpruned bootstrap trees, with the samples drawn per development year, not that of the bagged mean). The `print()`, `summary()`, `plot()` and `backtest()` methods describe the columns accordingly. In no case do the columns contain the uncertainty of the IBNR claim counts or of the model itself.

provenance the fitting settings, the fitting random-number information (`rng`), the package versions, the `completion` and, for simulated completion, `R` and the random-number information of the simulation (`completion_rng`: its `seed`, or with `seed = NULL` the generator state the draws started from)

See Also

`as_triangle()`, `calendar_year_payments()`, `summary.reserve_prediction()`, `plot.reserve_prediction()`, `backtest()`

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
# fit on the claims of one type, then complete another portfolio with the same layout:
# the same features in the same order; factor levels must be known to the fit
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
```

```
fit <- reserve_tree(panel, seed = 1)
cl2 <- synthetic_claims[synthetic_claims$Type == "4" & synthetic_claims$AccWeekday != "Sun", ]
pay2 <- synthetic_payments[synthetic_payments$Id %in% cl2$Id, ]
other <- claims_panel(cl2, pay2, t = 12, L = 11, features = c("Age", "AccWeekday"))
levels(other$claims$AccWeekday)          # a subset of the fit's levels is re-levelled
pred2 <- predict(fit, newdata = other)
calendar_year_payments(pred2)[c(1, 10), ]
```

`reserve_tree` *Fit one regression tree per development year*

Description

Fits a single-tree reserving model: for every development year `l = 0, ..., L - 1` (called `period` in the package's arguments and output), a regression tree (**rpart**, ANOVA method) that predicts the payment of the next development year, `X_{l+1}`, of a reported claim from its static features, its reporting delay and its payments `X_0, ..., X_l`. The tree for development year `l` is trained on the claims reported by development year `l` whose payment in development year `l + 1` is already observed at the valuation date (reporting delay `<= l` and `acc_year + l + 1 <= t`); every fitted development year needs at least two such claims. Each tree is grown with the given control (default `cp = 0` with ten-fold cross-validation) and, by default, pruned at the smallest cross-validated error (the simplest tree on ties), the pruning rule of Janoušek and Pešta (2025), the source article (deliberately not **rpart**'s customary one-standard-error rule, which prunes more and would turn more of the late, mostly-zero development years into single-leaf trees).

Usage

```
reserve_tree(
  panel,
  control = rpart::rpart.control(cp = 0, xval = 10),
  prune = c("min_xerror", "none"),
  seed = NULL,
  learner = NULL
)

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

Arguments

`panel`	A claims_panel with `L >= 1`.
`control`	An `rpart::rpart.control()` object; the default is `cp = 0` with `xval = 10`.
`prune`	`"min_xerror"` (default) prunes at the smallest cross-validated error (`xerror` of the complexity table; the simplest tree on ties; requires cross-validation, `xval >= 2` in `control`, and is an error otherwise); `"none"` keeps the full tree.

seed — Optional integer seed for the cross-validation folds, which are random. It is set with `set.seed(seed, kind = "Mersenne-Twister", normal.kind = "Inversion", sample.kind = "Rejection")`, R's default generator, so that the fit does not depend on the session's `RNGkind()`; the caller's generator kind and state are restored on exit. With `NULL` the current generator (of whatever kind) is used, its state recorded in the result and advanced, as by any random draw in R, so that repeated calls give different folds. (The stream allocation of `bag_reserve()` does not apply to single-tree fits.)

learner — `NULL` (default) fits pruned **rpart** trees as described above. Otherwise a list with functions `fit(formula, data)` and `predict(model, newdata)` that replaces them for every development year (see `bag_reserve()`); `control` and `prune` are then ignored (with a warning when they are given) and not validated; the fit records `prune = "none"`. `fit(formula, data)` receives the model frame of one development year (the features, `rep_delay`, `X0, ..., Xl` and the response `X(l+1)`, one row per training claim) and returns a model; `predict(model, newdata)` receives the same columns without the response and must return a numeric vector (or one-column matrix) with one finite value per row, which is checked for every prediction; an optional `simulate(model, newdata)` with the same contract is used by `predict(fit, completion = "draw")`. `two_part_learner()` builds one from two models.

object — A `reserve_tree` fit (for `summary`).

... — Not used; an argument given here is an error.

Details

`predict()` completes the payment histories of all reported claims recursively one development year ahead and returns a reserve_prediction with a single prediction column; see `predict.reserve_tree()`.

Value

An object of class `c("reserve_tree", "reserve_fit")`: a list with the fitted trees (`fits`, one per development year, `fits[[l + 1]]` for development year `l`), the chosen complexity parameters (`cp_chosen`, `NA` where no pruning was applied), training sizes (`n_train`), `prune`, `features`, `feature_levels`, `L`, `t`, the panel, the control, the RNG information (`rng`), the package versions (`versions`) and the call. `summary()` returns the same information with a table of leaves and depths per development year. With a user-supplied `learner`, `fits` holds the learner's models, whatever `fit()` returned; `print()` and `summary()` do not inspect them and report the training sizes per development year (the tree columns and the control apply to the default learner only).

See Also

`bag_reserve()` for the bagged version, `predict.reserve_tree()`.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
# a small subset for a quick example: the claims of one type
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
```

```
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
tree <- reserve_tree(panel, seed = 1)
tree
pred <- predict(tree)
calendar_year_payments(pred)
```

`summary.reserve_prediction` *Summarise a reserve prediction over its replicates*

Description

Summarises the future payments by component and calendar year across the retained replicate predictions: for every available component (reported-claim payments, IBNR component, total) and every future calendar year plus the total through development year `L`, the mean, standard deviation and empirical quantiles across the replicates. What the quantiles describe depends on the prediction, whose `variability` records it (see `predict.reserve_tree()`): for the replicates of `bag_reserve()`, each made by the trees of one bootstrap sample (unpruned by default, unlike the pruned trees of `reserve_tree()`), the resampling variation of the bagged trees; for the simulated completions of a single-tree fit (`predict(fit, completion = "draw")`), the process variability of payments simulated from that one fitted model, without estimation variability; for a bagged fit with `completion = "draw"`, the resampling variation of one set of bootstrap trees (samples drawn per development year) together with one simulated completion from each. They are in no case prediction intervals for the future payments (the IBNR claim counts are held fixed and the model is taken as given) nor confidence intervals for the mean. The printed summary states which case applies.

Usage

```
## S3 method for class 'reserve_prediction'
summary(
  object,
  probs = c(0.025, 0.25, 0.5, 0.75, 0.975, 0.995),
  type = 7L,
  ...
)
```

Arguments

`object`	A reserve_prediction.
`probs`	Probabilities of the empirical quantiles (numeric in `[0, 1]`; default `c(0.025, 0.25, 0.5, 0.75, 0.975, 0.995)`). The `q99.5%` column is the 99.5% quantile of the replicate (or simulated) predictions, as described above, not the 99.5% value at risk of the Solvency II capital requirement nor an upper prediction bound for the payments; `print()` says so for every probability of 0.99 or more.
`type`	Quantile type passed to `stats::quantile()`, an integer from 1 to 9. The source article used order statistics of 10,000 replicates, which corresponds to `type = 1`; the default is R's `type = 7`.
`...`	Not used; an argument given here (e.g. a misspelt `probs`) is an error.

Value

An object of class "summary.reserve_prediction": a list with tables, a named list of one matrix per component (rows: future calendar years and total; columns: mean, sd, min, the quantiles, max; sd is NA and only mean is printed when the prediction has a single column), plus B, estimator, completion, variability, probs, type, t and L.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
res <- ibnr_reserve(predict(bag_reserve(panel, B = 5, seed = 2026)), claim_counts(panel))
summary(res, probs = c(0.1, 0.5, 0.9))$tables$total
```

---

synthetic_claims *Synthetic individual claims observed at the reserving date*

Description

A synthetic individual claims portfolio generated with the Wang and Wüthrich (2022) individual claims generator (built on **SynthETIC**) on a yearly grid with seed 2026: six claim types, twelve accident years, reporting delays of up to eight years. It is a fresh portfolio from the same simulator family as the one analysed by Janoušek and Pešta (2025), not that article's data (which used another seed and ten accident years). synthetic_claims holds what the insurer sees at the reserving date $t = 12$: one row per *reported* claim (closed or RBNS), with the settlement information censored at $t = 12$. The claims that are still IBNR at that date are absent here and present only in synthetic_claims_full. Only quantities observable at $t = 12$ are kept: the generated ultimate claim amount and total number of payments are in synthetic_claims_full alone, so that nothing unobservable can leak into a fit.

Usage

```
synthetic_claims
```

Format

A data frame with one row per reported claim and the columns

Id claim identifier, integer, consistent across the four datasets

Type claim type, factor with levels 1–6

Age age of the insured at the accident, years

AccDate accident time, continuous, in years since the start of year 1

AccYear accident year i, 1–12

AccWeekday weekday of the accident, factor

RepDate reporting time, continuous, in years

RepYear reporting year $i + d$

RepDelDays reporting delay in the generator's day count (see above; not used)

RepDel reporting delay in years, $d =$ `RepYear - AccYear`, `0`–`8`

SetYear settlement year; `NA` if the claim is open (RBNS) at $t = 12$

SetDel settlement delay in years, `SetYear - RepYear`; `NA` if open

Status factor, `Closed` or `RBNS` at $t = 12$

CumPaid cumulative amount paid up to $t = 12$

Details

A word on the time grid. The generator was written for a monthly grid; run with `time_unit = 1` its yearly indices `AccYear`, `RepYear`, `SetYear` and `EventYear` are the development grid the method uses, and the reporting delay `RepDel = RepYear - AccYear` runs from 0 to 8 years. The generator's calendar conversion, however, treats one time unit as a month, so `RepDelDays` is the reporting delay in days divided by twelve and the three-year cap the generator places on it does not bind. Both columns are kept as generated; the method only uses the yearly grid.

Regenerate with `data-raw/make_synthetic_claims.R`; `data-raw/GENERATION.txt` records the generator call and the software versions used.

Source

Wang M, Wüthrich MV (2022). Individual claims generator for claims reserving studies: data simulation.R. SSRN 4127073. doi:10.2139/ssrn.4127073. Code: https://github.com/actuarial-data-science/PackageIndividualClaimsSimulator (GPL-3; formerly JSchelldorfer/IndividualClaimsSimulator, the location the vendored copy was taken from).

See Also

synthetic_payments, synthetic_claims_full, synthetic_payments_full

Other synthetic data: `synthetic_claims_full`, `synthetic_payments`, `synthetic_payments_full`

Examples

```
data("synthetic_claims", package = "baggingReserve")
str(synthetic_claims)
table(synthetic_claims$AccYear)
```

---

synthetic_claims_full *Synthetic individual claims: the generated truth*

Description

Every generated claim, including those that are IBNR at $t = 12$, with the uncensored settlement information. Used to back-test predictions of the outstanding liabilities. IBNR claims at $t = 12$ are those whose `Id` is not in synthetic_claims.

Usage

```
synthetic_claims_full
```

Format

A data frame with one row per generated claim and the columns

Id, Type, Age, AccDate, AccYear, AccWeekday, RepDate, RepYear, RepDelDays, RepDel as in synthetic_claims; here the reporting information is uncensored, so IBNR claims have `RepYear > 12`

SetYear settlement year, uncensored

SetDel settlement delay in years, `SetYear - RepYear`, uncensored

Ultimate ultimate claim amount as generated (not observable in practice)

PayCount total number of payments as generated (not observable in practice)

ReopenInd 1 if the claim was reopened after its first settlement

Source

Wang M, Wüthrich MV (2022). Individual claims generator for claims reserving studies: data simulation.R. SSRN 4127073. doi:10.2139/ssrn.4127073. Code: https://github.com/actuarial-data-science/PackageIndividualClaimsSimulator (GPL-3; formerly JSchelldorfer/IndividualClaimsSimulator, the location the vendored copy was taken from).

See Also

Other synthetic data: synthetic_claims, synthetic_payments, synthetic_payments_full

Examples

```
data("synthetic_claims", "synthetic_claims_full", package = "baggingReserve")
# claims reported after the valuation date t = 12 (the IBNR claims)
sum(!synthetic_claims_full$Id %in% synthetic_claims$Id)
```

---

synthetic_payments *Synthetic claim payments observed at the reserving date*

Description

One row per payment event up to and including year 12 on the claims in synthetic_claims. Events are yearly-gridded; several events of one claim can fall into the same year, negative amounts (recoveries) occur, and a few events carry no payment. Summing `Paid` per claim and year gives the incremental payments the method works with.

Usage

```
synthetic_payments
```

Format

A data frame with the columns

Id claim identifier, matches synthetic_claims

EventId running number of the event within the claim

EventYear calendar year of the event, $i + d + k$

Paid amount paid (negative for recoveries, zero for non-payment events)

PayInd `1` for a payment, `0` for a recovery or a zero-amount event, as generated

OpenInd `1` if the claim is still open after the event, as generated

Source

Wang M, Wüthrich MV (2022). Individual claims generator for claims reserving studies: data simulation.R. SSRN 4127073. doi:10.2139/ssrn.4127073. Code: `https://github.com/actuarial-data-science/PackageIndividualClaimsSimulator` (GPL-3; formerly JSchelldorfer/IndividualClaimsSimulator, the location the vendored copy was taken from).

See Also

Other synthetic data: `synthetic_claims`, `synthetic_claims_full`, `synthetic_payments_full`

Examples

```
data("synthetic_payments", package = "baggingReserve")
head(synthetic_payments)
tapply(synthetic_payments$Paid, synthetic_payments$EventYear, sum)
```

`synthetic_payments_full`

Synthetic claim payments: the generated truth

Description

Every generated payment, past and future, for the claims in synthetic_claims_full. Same columns as synthetic_payments. One generated event far beyond the simulated horizon carried no amount and is dropped (the count is recorded in `data-raw/GENERATION.txt`).

Usage

```
synthetic_payments_full
```

Format

A data frame with the columns of synthetic_payments.

Source

Wang M, Wüthrich MV (2022). Individual claims generator for claims reserving studies: data simulation.R. SSRN 4127073. doi:10.2139/ssrn.4127073. Code: https://github.com/actuarial-data-science/PackageIndividualClaimsSimulator (GPL-3; formerly JSchelldorfer/IndividualClaimsSimulator, the location the vendored copy was taken from).

See Also

Other synthetic data: synthetic_claims, synthetic_claims_full, synthetic_payments

Examples

```
data("synthetic_payments_full", package = "baggingReserve")
# realised payments after the valuation date t = 12, by calendar year
fut <- synthetic_payments_full[synthetic_payments_full$EventYear > 12, ]
tapply(fut$Paid, fut$EventYear, sum)
```

---

two_part_learner *A two-part learner: whether a claim pays, and how much*

Description

Builds a learner for reserve_tree() and bag_reserve() from two models: one for the probability that a claim makes a non-zero payment in the development year, one for the amount of a non-zero payment. Most payments of the later development years are exactly zero; a single smooth model then predicts a small positive amount for every claim, and the recursive completion carries these forward. Separating occurrence from size, and simulating the occurrence (predict(fit, completion = "draw")), avoids this.

Usage

```
two_part_learner(occurrence, size, min_nonzero = 20L)
```

Arguments

occurrence	A list with fit(formula, data) and predict(model, newdata); it is fitted to a 0/1 response (1 for a non-zero payment) and must predict probabilities. The outputs of both models are checked separately, before they are combined, against the learner contract of reserve_tree(): a numeric vector (or one-column matrix) with one finite value per row of newdata (per simulated payment for size$simulate()), and for the occurrence model values in [0, 1]. A violation is an error naming the model, so that R's recycling cannot turn, say, a single number returned for a whole batch into a prediction.
size	A list with fit(formula, data) and predict(model, newdata), fitted to the non-zero payments only, and optionally simulate(model, newdata), returning one simulated non-zero amount per row (used by completion = "draw"; without it the predicted amount is used).

`min_nonzero` When fewer non-zero payments are available in a development year, the size model is replaced by their mean (default 20).

Details

The prediction is `p * s`, the probability of a payment times the predicted amount; the simulation, used by `completion = "draw"`, draws the payment indicator and returns `s` when it is 1, else 0. Only the occurrence is then random: the amount of a simulated payment is the size model's predicted mean, so the simulated completions contain no variation of the payment size given its predictors, unless `size` has a `simulate(model, newdata)` function, which is then used for the amount of every simulated payment.

Value

A list with functions `fit`, `predict` and `simulate`, to pass as `learner`.

Examples

```
data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
cl <- synthetic_claims[synthetic_claims$Type == "3", ]
pay <- synthetic_payments[synthetic_payments$Id %in% cl$Id, ]
panel <- claims_panel(cl, pay, t = 12, L = 11, features = c("Age", "AccWeekday"))
logistic <- list(
 fit = function(formula, data) stats::glm(formula, data = data, family = stats::binomial()),
  predict = function(model, newdata) stats::predict(model, newdata, type = "response"))
linear <- list(
  fit = function(formula, data) stats::lm(formula, data = data),
  predict = function(model, newdata) stats::predict(model, newdata))
fit <- reserve_tree(panel, learner = two_part_learner(logistic, linear))
calendar_year_payments(predict(fit))["total", ]
calendar_year_payments(predict(fit, completion = "draw", R = 20, seed = 1))["total", ]
```

Index

∗ **datasets**
synthetic_claims, 33
synthetic_claims_full, 34
synthetic_payments, 35
synthetic_payments_full, 36
∗ **synthetic data**
synthetic_claims, 33
synthetic_claims_full, 34
synthetic_payments, 35
synthetic_payments_full, 36

as.data.frame.claims_panel *(*claims_panel*)*, 16
as.data.frame.reserve_backtest *(*as.data.frame.reserve_prediction*)*, 3
as.data.frame.reserve_prediction, 3
as_triangle *(*as_triangle.chain_ladder*)*, 4
as_triangle(), *15*, *19*, *29*
as_triangle.chain_ladder, 4

backtest, 5
backtest(), *3*, *9*, *18, 19*, *24*, *29*
bag_reserve, 9, *22*, *26*
bag_reserve(), *17*, *19*, *28, 29*, *31, 32*, *37*

calendar_year_payments, 12
calendar_year_payments(), *6*, *8*, *15*, *29*
calendar_year_payments.chain_ladder, 13
calendar_year_payments.chain_ladder(), *13*
chain_ladder, *4*, *6*, *8*, *13*, 14, *14*
chain_ladder(), *20*
claim_counts *(*as_triangle.chain_ladder*)*, 4
claim_counts(), *19, 20*, *29*
claim_reserves, 15
claim_reserves(), *29*
claims_panel, *4*, *10*, 16, *25–27*, *30*
claims_panel(), *6, 7*, *20*
codetools::findGlobals(), *10*

get(), *10*
graphics::hist(), *24*
graphics::plot(), *22*, *24*
graphics::stripchart(), *24*

ibnr_reserve, 19
ibnr_reserve(), *8*, *12, 13*, *18*, *28, 29*

oob_score, 21
oob_score(), *10*, *12*

parallel::detectCores(), *10*
plot.bag_reserve, 22
plot.reserve_prediction, 23
plot.reserve_prediction(), *29*
plot.reserve_tree, 25
predict.bag_reserve, 25
predict.bag_reserve(), *11, 12*
predict.reserve_tree, 27
predict.reserve_tree(), *11*, *23*, *26*, *31, 32*
print.reserve_triangle *(*as_triangle.chain_ladder*)*, 4

reserve_prediction, *3, 4*, *6*, *12, 13*, *16*, *19, 20*, *23–26*, *31, 32*
reserve_prediction *(*predict.reserve_tree*)*, 27
reserve_tree, *25*, *27*, 30
reserve_tree(), *9*, *11*, *17*, *19*, *28*, *32*, *37*
RNGkind(), *31*
rpart::plot.rpart(), *25*
rpart::rpart.control(), *10*, *30*

stats::quantile(), *6*, *24*, *32*
summary.bag_reserve *(*bag_reserve*)*, 9
summary.claims_panel *(*claims_panel*)*, 16
summary.reserve_prediction, 32

summary.reserve_prediction(), *5, 6*, *11*, *24*, *29*
summary.reserve_tree *(*reserve_tree*)*, 30
synthetic_claims, 33, *34–37*
synthetic_claims_full, *5*, *33*, 34, *34*, *36, 37*
synthetic_payments, *34*, 35, *35–37*
synthetic_payments_full, *5*, *34, 35*, 36, *36*

two_part_learner, 37
two_part_learner(), *10, 11*, *31*