Package: scorecraft (via r-universe)

October 10, 2026

Title Scorecard Development and Internal Ratings-Based Risk Parameters

Version 0.3.2

Description Builds points scorecards for binary targets (credit risk, fraud, propensity) on the optimal binning and weight of evidence engine of 'OptimalBinningWoE', and takes them to the risk parameters of the internal ratings-based (IRB) approach. Variables are selected through optimal binning, eight admission rules, hold-out revalidation with frozen bins and a consensus of 'glmnet', 'xgboost', 'lightgbm' and 'ranger' models weighted by out-of-sample performance; the audit funnel never drops a candidate from the report. The scorecard is fitted with an explicit, auditable scale alignment (a log-odds regression on the raw score composed with the points-to-double-the-odds map); cut-offs are swept with frozen cuts; reject inference is reported as a sensitivity band; the population and characteristic stability indices (PSI and CSI) are monitored with both the fixed and the sample-size-adjusted threshold; and production SQL is generated in fourteen dialects, with the agreement between R and SQL verified by test. The IRB layer builds the default flag; calibrates the scorecard to a long-run default rate with rating grades, margins of conservatism and floors to give the probability of default (PD); models workout loss given default (LGD) in two stages with downturn and in-default estimates; models credit conversion factors from facility snapshots to give the exposure at default (EAD); and computes expected loss, risk weights, regulatory capital and expected credit loss from parameter tables selected by framework preset. The heavy numeric kernels (rank correlation of wide weight of evidence tables, exact concordance counts for Somers' D, streamed expected credit loss paths) are compiled with 'RcppArmadillo'. The scorecard methodology follows Siddiqi (2017) <doi:10.1002/9781119282396> and Thomas et al. (2017) <doi:10.1137/1.9781611974560>.

License MIT + file LICENSE

URL https://github.com/evandeilton/scorecraft,

https://evandeilton.github.io/scorecraft/

BugReports https://github.com/evandeilton/scorecraft/issues

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports data.table (>= 1.14.0), OptimalBinningWoE (>= 1.13.4), xgboost, stats, utils, graphics, parallel, Rcpp (>= 1.0.10)

LinkingTo Rcpp, RcppArmadillo

Suggests glmnet, lightgbm, ranger, DBI, odbc, RSQLite, duckdb, openxlsx, betareg, bit64, survival, testthat (>= 3.0.0), knitr, rmarkdown, withr

LazyData true

LazyDataCompression xz

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Jose Evandeilton Lopes [aut, cre, cph]

Maintainer Jose Evandeilton Lopes <evandeilton@gmail.com>

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

Date/Publication 2026-10-10 18:43:21 UTC

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

RemoteRef HEAD

RemoteSha 1c86b6ccf25f1253a6faa2f600caad1b2d1e4c78

Contents

predict.scr_align 4
predict.scr_grades 5
predict.scr_pd 6
scr_align 7
scr_apply 9
scr_bands 11
scr_bin 16
scr_bin_continuous 17
scr_calibrate 19
scr_capital 21
scr_claims 23
scr_classing_accept 27
scr_classing_apply 29

scr_classing_choose . . . 30
scr_classing_propose . . . 31
scr_classing_spec . . . 32
scr_classing_view . . . 33
scr_coarse_classing . . . 34
scr_compare . . . 36
scr_config . . . 37
scr_config_keys . . . 39
scr_connect . . . 40
scr_core . . . 41
scr_cutoff . . . 41
scr_decisions . . . 42
scr_default . . . 43
scr_default_rate . . . 45
scr_demo . . . 46
scr_demo_ead . . . 47
scr_demo_lgd . . . 48
scr_demo_lgd_cashflows . . . 49
scr_demo_panel . . . 50
scr_demo_portfolio . . . 51
scr_demo_rates . . . 52
scr_detection . . . 53
scr_ead . . . 55
scr_ead_data . . . 57
scr_ead_downturn . . . 59
scr_ead_validate . . . 60
scr_ecl . . . 61
scr_el . . . 63
scr_elbe . . . 64
scr_export . . . 65
scr_fetch . . . 68
scr_funnel . . . 69
scr_gains . . . 70
scr_grades . . . 71
scr_irb_params . . . 73
scr_irb_rw . . . 74
scr_iv . . . 76
scr_leakage . . . 77
scr_lgd . . . 78
scr_lgd_downturn . . . 80
scr_lgd_floor . . . 81
scr_lgd_pools . . . 82
scr_lgd_validate . . . 83
scr_master_scale . . . 84
scr_maturity . . . 85
scr_metrics . . . 88
scr_migration . . . 90
scr_mix_shift . . . 91

scr_moc . . . 94
scr_model . . . 95
scr_monitor . . . 96
scr_monitoring_plan . . . 98
scr_operating . . . 98
scr_overlap . . . 103
scr_pd . . . 106
scr_pd_pit_ttc . . . 108
scr_pd_stress . . . 109
scr_pd_validate . . . 110
scr_presets . . . 112
scr_psi . . . 112
scr_rag . . . 114
scr_rag_plan . . . 118
scr_reasons . . . 120
scr_reject . . . 121
scr_result . . . 122
scr_run . . . 123
scr_runset . . . 124
scr_sa_rw . . . 125
scr_score_cross . . . 126
scr_score_gains . . . 129
scr_score_metrics . . . 130
scr_scorecard . . . 131
scr_segments . . . 133
scr_select . . . 137
scr_selected . . . 139
scr_split . . . 140
scr_sql . . . 141
scr_strategy . . . 144
scr_tiers . . . 147
scr_triage . . . 152
scr_uplift . . . 153
scr_verbose . . . 156
scr_workout . . . 157

Index **159**

`predict.scr_align` *Apply an alignment to raw scores*

Description

Apply an alignment to raw scores

Usage

```
## S3 method for class 'scr_align'
predict(object, raw, type = c("score", "prob"), ...)
```

Arguments

object	An object from scr_align().
raw	Raw scores on the same scale used in the fit.
type	"score" (default) returns points; "prob" returns the calibrated event probability implied by the alignment.
...	Ignored.

Value

A numeric vector of the length of raw.

See Also

Other production: scr_apply(), scr_export(), scr_monitor(), scr_monitoring_plan(), scr_reasons(), scr_sql()

Examples

```
set.seed(3)
y   <- stats::rbinom(2000, 1, 0.15)
raw <- stats::qlogis(0.15) + 1.3 * y + stats::rnorm(2000)
al  <- scr_align(raw, y)
head(predict(al, raw))
head(predict(al, raw, type = "prob"))
```

---

predict.scr_grades *Grade a score vector with the cut points of an scr_grades object*

Description

Grade a score vector with the cut points of an scr_grades object

Usage

```
## S3 method for class 'scr_grades'
predict(object, score, type = c("grade", "pd"), ...)
```

Arguments

object	An scr_grades() object.
score	Numeric production scores.
type	"grade" (integer grade) or "pd" (calibrated individual PD).
...	Ignored.

Value

A vector of the length of `score`.

See Also

Other irb-pd: `predict.scr_pd()`, `scr_calibrate()`, `scr_grades()`, `scr_master_scale()`, `scr_migration()`, `scr_moc()`, `scr_pd()`, `scr_pd_pit_ttc()`, `scr_pd_validate()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
sc <- scr_scorecard(res)
gr <- scr_grades(sc, scr_calibrate(sc, target = 0.06), n_grades = 7, min_defaults = 10)
predict(gr, score = c(480, 560, 640))
predict(gr, score = c(480, 560, 640), type = "pd")
```

`predict.scr_pd` *Predict grade and PD from an scr_pd object*

Description

Predict grade and PD from an scr_pd object

Usage

```
## S3 method for class 'scr_pd'
predict(
  object,
  newdata = NULL,
  score = NULL,
  type = c("grade", "pd", "pd_final", "score"),
  ...
)
```

Arguments

`object`	An `scr_pd()` object.
`newdata`	A table with the source columns of the scorecard, scored with `scr_apply()`; ignored when `score` is given.
`score`	Production scores, as an alternative to `newdata`.
`type`	`"grade"`, `"pd"` (calibrated individual PD), `"pd_final"` (grade PD after MoC and floor) or `"score"`.
`...`	Ignored.

Value

A vector of the length of the input.

See Also

Other irb-pd: predict.scr_grades(), scr_calibrate(), scr_grades(), scr_master_scale(), scr_migration(), scr_moc(), scr_pd(), scr_pd_pit_ttc(), scr_pd_validate()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
sc <- scr_scorecard(res)
pd <- scr_pd(scr_grades(sc, n_grades = 6, min_defaults = 10))
predict(pd, score = c(480, 560, 640), type = "pd_final")
head(predict(pd, newdata = scr_demo[1:10, ], type = "grade"))
```

---

scr_align *Stage 5: align a raw score to the declared scale*

Description

Takes the raw score of **any** engine (the scorecard logit, the output of a tree challenger, a legacy score) to the scale defined by `base_score`, `base_odds` and `pdo`, recording `odds_orientation` on the object. This is what makes two scorecards directly comparable. It runs automatically inside scr_scorecard(); it is exposed to align other scores to the same scale.

Usage

```
scr_align(
  raw,
  y,
  base_score = 600,
  base_odds = 50,
  pdo = 20,
  direction = c("higher_is_safer", "higher_is_riskier"),
  method = c("regression", "direct"),
  n_bands = 10L,
  laplace = 0.5,
  weights = NULL
)
```

Arguments

raw	Raw score: an event logit (or any score on which a higher value means a higher probability of the event).
y	0/1 outcome vector (numeric or logical), same length as raw.
base_score	Score at which the odds are base_odds.
base_odds	Odds at base_score, positive, in the orientation of direction.
pdo	Points that double the odds, positive.
direction	"higher_is_safer" or "higher_is_riskier".
method	"regression" (default) or "direct".
n_bands	Bands of the calibration regression.
laplace	Smoothing of the counts per band.
weights	Optional non-negative weights per observation (sample reweighting), of the length of raw.

Value

An scr_align object with base_score, base_odds, pdo, direction, odds_orientation, factor, offset, sign, calibration (method, intercept, slope, r2, n_bands, bands) and the final coefficients a and b of score = a + b * raw. Use predict.scr_align() to apply it.

Mechanism

With method = "regression" (default): the raw score is banded by quantiles on the reference data, the empirical log-odds of every band is computed with Laplace smoothing in the orientation direction implies, and a regression weighted by band size fits

$$\ln(\text{odds}) = I + S \cdot \text{raw}.$$

This absorbs sample reweighting, miscalibration of the WOE fit and prior shift. It then composes with the PDO map:

$$\text{factor} = \text{pdo}/\ln 2, \quad \text{offset} = \text{base_score} - \text{factor} \cdot \ln(\text{base_odds}),$$

$$\text{score} = \text{offset} + \text{factor}\,(I + S \cdot \text{raw}) = a + b \cdot \text{raw}.$$

With method = "direct", the model is assumed calibrated: I = 0 and S is the sign of the direction (-1 under higher_is_safer, +1 under higher_is_riskier), that is, ln(odds) is raw itself in the right orientation.

Odds orientation

base_odds is always expressed in the orientation direction implies: non-event:event ("safe:event") under higher_is_safer, event:non-event ("event:safe") under higher_is_riskier. The same word, odds, changes meaning between the two, and that is the most common sign trap in the literature; hence the object records odds_orientation explicitly.

References

Siddiqi, N. (2006). *Credit Risk Scorecards*. Wiley, chapter 6.

See Also

Other stages: scr_bin(), scr_cutoff(), scr_model(), scr_reject(), scr_scorecard(), scr_select(), scr_split(), scr_strategy(), scr_triage()

Examples

```
set.seed(3)
y   <- stats::rbinom(4000, 1, 0.15)
raw <- stats::qlogis(0.15) + 1.3 * y + stats::rnorm(4000, sd = 1.2)
al  <- scr_align(raw, y, base_score = 600, base_odds = 50, pdo = 20)
al
head(predict(al, raw))
head(predict(al, raw, type = "prob"))

# propensity: more points = more event, odds event:non-event
scr_align(raw, y, base_score = 500, base_odds = 1/9, pdo = 40,
          direction = "higher_is_riskier")
```

---

scr_apply *Apply the WOE transformation or the scorecard to new data*

Description

Materializes in R exactly what the production SQL does: the frozen Stage 1 pre-processing (training median, special-population flags, "MISSING") followed by the frozen Stage 2 binning and, for a scorecard, by the points. Nothing is refitted. The two paths, R and SQL, produce the same numbers, and a test guarantees it.

Usage

```
scr_apply(x, newdata, ...)

## S3 method for class 'scr_result'
scr_apply(
  x,
  newdata,
  features = scr_selected(x),
  what = c("woe", "bin", "both"),
  ...
)

## S3 method for class 'scr_scorecard'
scr_apply(x, newdata, what = c("score", "points", "woe", "all"), ...)
```

```
## S3 method for class 'scr_ead'
scr_apply(x, newdata, what = c("all", "ead", "pool"), ...)

## S3 method for class 'scr_lgd'
scr_apply(x, newdata, what = c("pool", "lgd", "all"), ...)

## S3 method for class 'scr_pd'
scr_apply(x, newdata, ...)

## S3 method for class 'scr_study'
scr_apply(x, newdata, score = "score", numbered = TRUE, ...)
```

Arguments

x	An object from scr_select() (returns WOE/bin of the approved variables), from scr_scorecard() (returns score and points), or a score study from scr_bands() or scr_tiers() (returns the band or tier of a score).
newdata	New table with the source columns of the requested variables. The target column is not needed.
...	Passed on to the methods.
features	For scr_result: which variables to transform. Defaults to the approved ones.
what	For scr_result: "woe", "bin" or "both". For scr_scorecard: "score", "points", "woe" or "all". For scr_lgd: "pool" (pool and pool LGDs), "lgd" (adds the predicted LGD) or "all" (adds the cure probability and the severity). For scr_ead: "all" (default), "ead" (pool, measure, applied CCF, predicted EAD and the floor flag) or "pool" (pool and measure).
score	For scr_study: name of the score column of newdata. newdata may also be a numeric vector of scores.
numbered	For a tiers study: TRUE (default) returns the tier labels with their order in front ("01.very high"), FALSE the plain labels. Band labels are intervals and never get a prefix.

Value

A data.table with one row per row of newdata.

Output columns

For scr_result: <f>_woe and/or <f>_bin per variable. For scr_scorecard, "score" gives link (logit), prob (model probability), score (exact, a + b * logit) and score_points (base plus the whole points per bin); "points" gives score, score_points and <f>_points; "woe" gives link, score and <f>_woe; "all" gives everything.

IRB models

scr_pd returns score, score_points, grade, pd (calibrated individual PD), pd_be and pd_final of the grade. scr_lgd returns pool, lgd_lra, lgd_dt, lgd_final and, with what, p_cure, severity and lgd_pred. scr_ead returns pool, measure, utilisation, undrawn, ccf_applied, ead_model,

ead_floor, ead_predicted and ead_floor_binding; the predicted EAD is never below the drawn amount. scr_capital() reads pd_final, lgd_final and ead_predicted from these outputs in its list form.

Score studies

For a score study (scr_bands(), scr_tiers()), newdata is returned (as a copy) with tier, the band or tier number, and tier_label. The intervals are left-closed: score >= cut is the upper side, and a missing score gives a missing tier.

The labels of a tiers study carry their order, "01." for the tier with the highest event rate (the first row of the tiers table) down to the tier with the lowest, so they sort from the event-richest tier under any objective and direction; tier is unchanged and still rises with the event rate. The result joins to the tier_label column of the tiers table. numbered = FALSE returns the plain labels (its label column).

See Also

Other production: predict.scr_align(), scr_export(), scr_monitor(), scr_monitoring_plan(), scr_reasons(), scr_sql()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
new <- head(scr_demo, 50)
str(scr_apply(res, new)[, 1:3])
sc <- scr_scorecard(res)
head(scr_apply(sc, new))
head(scr_apply(sc, new, what = "points"))
```

---

scr_bands *Percentile study of a score*

Description

Cuts the score into bands of equal share (or into tail percentiles) **frozen on the reference sample**, and reads every band on the reference and on the study samples: volume, event rate with a Jeffreys interval, lift, capture (recall), the cumulative non-event share (false positive rate), KS, the band WOE and IV, odds, the PSI term against the reference and a one-sided Fisher exact test of rank order against the previous band. The summary adds the AUC, Gini and KS of every sample with a bootstrap interval.

Usage

```
scr_bands(x, ...)

## S3 method for class 'scr_scorecard'
scr_bands(
  x,
  n_bands = NULL,
  spacing = c("uniform", "tail"),
  tail_probs = NULL,
  sample = "holdout",
  reference = "train",
  breaks = NULL,
  level = NULL,
  n_boot = NULL,
  seed = NULL,
  max_cells = 1e+05,
  boot_cells = 10000,
  ...
)

## S3 method for class 'data.frame'
scr_bands(
  x,
  score = "score",
  y = "y",
  objective = "risk",
  direction = NULL,
  weight = NULL,
  value = NULL,
  sample = NULL,
  reference = NULL,
  study = NULL,
  counts = FALSE,
  n = "n",
  events = "events",
  n_bands = 20L,
  spacing = c("uniform", "tail"),
  tail_probs = NULL,
  breaks = NULL,
  level = 0.95,
  n_boot = 200L,
  seed = NULL,
  max_cells = 1e+05,
  value_events = NULL,
  boot_cells = 10000,
  ...
)
```

Arguments

x	An object from scr_scorecard(), or a data.frame with one row per scored case (or one row per score value with counts = TRUE).
...	Passed on to the methods; an unknown argument is an error.
n_bands	Number of equal-share bands. For a scorecard, NULL uses config$study_bands (20).
spacing	"uniform" (equal shares) or "tail" (the shares of tail_probs, counted from the event-rich side).
tail_probs	Cumulative shares from the event-rich side for spacing = "tail". NULL uses 0.001, 0.005, 0.01, 0.02, 0.05, 0.10, 0.20 and 0.50.
sample	For a scorecard: the study sample(s), "holdout" (default) and/or "train". For a data.frame: the name of a column with sample labels, or NULL (all rows are one sample, reference and study at once).
reference	For a scorecard: the sample the bands are frozen on ("train"). For a data.frame: the label of the reference sample; NULL takes the first level of the sample column (the levels of a factor in their order, numbers in numeric order, text sorted).
breaks	Explicit ascending cut points; overrides n_bands and spacing. Infinite values are ignored.
level	Confidence level of the intervals. For a scorecard, NULL uses config$study_level (0.95).
n_boot	Bootstrap resamples of the AUC interval (0 skips it). For a scorecard, NULL uses config$n_boot.
seed	Seed of the bootstrap. A number is local to the call (the user's random stream is restored on exit); NULL draws from the user's stream and advances it. For a scorecard, NULL uses config$seed.
max_cells	Largest number of distinct score values kept exactly.
boot_cells	Largest number of score cells resampled exactly by the bootstrap (default 10,000; Inf for no pooling). See the section Method.
score, y	Column names of the score and of the 0/1 outcome (NA allowed).
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).
direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
weight	Optional column of non-negative case weights.
value	Optional column of a value per case (an amount, a balance): adds the value captured per band. With counts = TRUE, the value per score cell.
study	Labels of the study samples; NULL takes every level other than the reference.
counts	TRUE when x is pre-aggregated: one row per score value with the columns score, n and events (and, optionally, value and value_events).
n, events	Column names of the counts when counts = TRUE.
value_events	With counts = TRUE: the column of the value of the events per score cell.

Value

An object of class `c("scr_study_bands", "scr_study", "list")`:

`table` One row per sample and band, event-richest band first: `sample`, `band`, `label`, `score_lo`, `score_hi`, `n`, `pct`, `cum_pct`, `events`, `rate`, `rate_lo`, `rate_hi`, `cum_rate`, `lift`, `lift_lo`, `lift_hi`, `cum_lift`, `capture`, `cum_nonevent_pct`, `ks`, `pct_event`, `pct_nonevent`, `woe`, `iv`, `odds`, `log_odds`, `psi`, `p_reversal`, `p_reversal_adj` and, with `value`, `value`, `value_events`, `value_capture` (cumulative share of the event value) and `value_precision` (cumulative event value over cumulative value).

`summary` One row per sample: `sample`, `n`, `events`, `rate`, `auc`, `auc_lo`, `auc_hi`, `gini`, `gini_lo`, `gini_hi`, `ks`, `iv`, `psi` (against the reference), `n_bands_requested`, `n_bands_effective` and `reversals` (bands with `p_reversal_adj` below 0.05).

`cuts` The ascending cut points.

`codes`, `code_labels` Band number and label of every interval in ascending score order, used by `scr_apply()` and `scr_sql()`.

`objective`, `direction`, `level`, `spacing`, `reference`, `samples`, `target`, `call` The settings.

`n_bands_requested`, `n_bands_effective`, `quantized`, `hist` The band counts, whether the scores were pooled, and the count table the study was computed from.

Method

The scored rows are aggregated once into a table of counts per distinct score (per sample); every statistic is then computed from that table, so the cost is one pass over the rows plus work proportional to the number of distinct scores. With more than `max_cells` distinct scores, the scores are first pooled into `max_cells` cells of equal weighted share.

The cut targets are cumulative shares counted from the event-rich side of the score (the high scores under `higher_is_riskier`, the low ones under `higher_is_safer`): `k / n_bands` with `spacing = "uniform"`, or the shares in `tail_probs` with `spacing = "tail"` (default 0.1%, 0.5%, 1%, 2%, 5%, 10%, 20% and 50%). Each cut is placed midway between two adjacent distinct reference scores, at the boundary nearest to its target, so a group of tied scores is never split and a target is hit within the share of one score value. Targets that land on the same boundary give one cut: `n_bands_effective` can be smaller than `n_bands_requested`, and both are reported. A band is left-closed, `[lo, hi)`: `score >= cut` is the upper side, the convention of `scr_cutoff()`. `breaks` given explicitly are used as they are; the bands of `scr_score_gains()` (`breaks = sc$breaks`) are right-closed, so a score equal to a break falls one band higher here.

The band table lists the event-richest band first (`band = 1`). With e_b events and m_b non-events in band b, totals E and M, and overall rate R:

- `rate` = $e_b/(e_b + m_b)$, with the Jeffreys interval `rate_lo`, `rate_hi`: the Beta$(e_b + 1/2, m_b + 1/2)$ quantiles, 0 and 1 at the edges (Brown, Cai and DasGupta, 2001). Under weights, the counts are scaled to the Kish effective size $(\sum w)^2/\sum w^2$.
- `lift` = `rate` / R (its interval divides the rate bounds by R); `cum_rate` and `cum_lift` accumulate from the first band.
- `capture` = $\sum_{j \le b} e_j/E$ (recall), `cum_nonevent_pct` = $\sum_{j \le b} m_j/M$ (false positive rate) and `ks` their absolute difference.

- pct_event, pct_nonevent and woe = $\ln(e_b/E) - \ln(m_b/M)$ as in scr_strategy() (0.5 is added to every band only when a band lacks events or non-events); iv = (pct_event - pct_nonevent) * woe.
- odds and log_odds in the orientation of the scale (non-events per event under higher_is_safer, events per non-event under higher_is_riskier, 0.5 added to each count), as in scr_score_gains().
- psi: the band term of the PSI against the reference shares (see scr_psi()); NA on the reference itself.
- p_reversal: one-sided Fisher exact test that the band has a **higher** event rate than the previous, event-richer band (a reversal of the rank order); p_reversal_adj is Holm-adjusted over the bands of the sample. The tests use the unweighted counts.

Rows with a missing or infinite score, or a zero weight, are not counted; rows with a missing outcome count in the volume (n, pct, the PSI) but not in the rates.

The bootstrap of the AUC draws the event and non-event counts of every score value from multinomial laws with the observed shares (the law of a row bootstrap stratified by outcome), with the unweighted class counts as sizes; a given seed is local to the call, while seed = NULL draws from, and advances, the user's random stream. Up to boot_cells cells of the count table the bootstrap is exact. Above it, the resamples run on boot_cells cells of equal share (adjacent cells pooled) and are shifted to the point estimate of the full table, so the interval is an approximation. The point estimates use every cell of the count table: every distinct score, unless max_cells pooled the scores into cells. boot_cells = Inf keeps the bootstrap exact, at a cost per resample proportional to the number of cells.

References

Brown, L. D., Cai, T. T. and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133. doi:10.1214/ss/1009213286

DeLong, E. R., DeLong, D. M. and Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, 44(3), 837-845.

Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.

Yurdakul, B. and Naranjo, J. (2020). Statistical properties of the population stability index. *Journal of Risk Model Validation*, 14(4), 89-100.

See Also

scr_tiers() for a small number of policy tiers, scr_rag() for traffic lights, scr_apply() and scr_sql() to assign the bands in production.

Other score-studies: scr_claims(), scr_detection(), scr_maturity(), scr_mix_shift(), scr_operating(), scr_overlap(), scr_rag(), scr_rag_plan(), scr_score_cross(), scr_segments(), scr_tiers(), scr_uplift()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 20)
```

```
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
b <- scr_bands(sc, n_bands = 10)
b
b$table[sample == "holdout", .(band, label, n, rate, lift, capture, ks)]

# tail percentiles, from a data.frame
d <- data.frame(score = sc$samples$holdout$score, y = sc$samples$holdout$y)
scr_bands(d, spacing = "tail", n_boot = 0)$table[, .(band, label, pct, rate, capture)]
```

`scr_bin` *Stage 2: optimal binning, screening, hold-out revalidation and pruning*

Description

Fits the bins **on the training rows only** (cut points and WOE use the target), in parallel by column, and applies in sequence:

Usage

```
scr_bin(triage, config = scr_config())
```

Arguments

`triage`	An object from `scr_triage()`.
`config`	An object from `scr_config()`.

Details

1. **Screening**, native to the engine: eight admission rules (`IV_BELOW_MIN`, `IV_SUSPICIOUS`, `NOT_MONOTONIC`, `TOO_FEW_BINS`, `TOO_MANY_BINS`, `SMALL_BIN`, `DEGENERATE_BIN`, `BINNING_ERROR`).
2. **Hold-out revalidation** with frozen bins: IV recomputed on the same labels, train/hold-out PSI (the fixed threshold decides; the n-adjusted one is reported) and the fraction of hold-out without a bin.
3. **Redundancy pruning** by rank correlation on the WOE space, ranked by hold-out IV. Under `allow_derived_final = FALSE` the derived flags leave before this step (`derived_excluded`), so that a flag that cannot be delivered never prunes a real column.

Value

An `scr_bins` object with `fit` (an obwoe object), `screen` (`summary` and `full`), `holdout`, `prune`, `pool` (eligible for the models), `derived_excluded`, the counts `binned`, `pos_screen` and `pos_holdout` (the survivors of each gate, in order), the WOE matrices `woe_train`/`woe_holdout`, the originating `triage` and the `config`.

Parallelism

Columns are split into `config$nthread` chunks, each chunk is binned by a worker and the fits are merged. The result is identical to the serial one (a test pins this): the engine is deterministic per column.

See Also

Other stages: `scr_align()`, `scr_cutoff()`, `scr_model()`, `scr_reject()`, `scr_scorecard()`, `scr_select()`, `scr_split()`, `scr_strategy()`, `scr_triage()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1)
sp <- scr_split(scr_demo, "default", date_col = "ref_date", drop = "id")
bn <- scr_bin(scr_triage(sp, cfg), cfg)
bn
head(bn$holdout)
```

---

scr_bin_continuous *Bin drivers against a continuous target (LGD, CCF)*

Description

Supervised binning for a bounded continuous target, with the result in the shape of an obwoe object, so that the `scr_apply()` and `scr_sql()` machinery (`OptimalBinningWoE::obwoe_apply()` and `OptimalBinningWoE::obwoe_sql()`) reproduces the bin statistic unchanged. The `woe` slot of every bin carries the target mean of the bin (or its logit with `scale = "logit"`); `iv` carries the bin's share of the between-bin sum of squares, so `total_iv` is the eta-squared of the driver, in `[0, 1]`.

Usage

```
scr_bin_continuous(
  data,
  target,
  features,
  train_idx = NULL,
  holdout_idx = NULL,
  min_bins = 2L,
  max_bins = 6L,
  min_share = 0.05,
  min_n = 30L,
  monotone = c("auto", "increasing", "decreasing", "none"),
  scale = c("mean", "logit"),
  nthread = 1L,
  alpha = 0.05
)
```

Arguments

data	A data.frame or data.table.
target	Column name of the continuous target.
features	Column names of the drivers.
train_idx, holdout_idx	Row indices; NULL uses every row for training and skips the revalidation.
min_bins, max_bins	Target range of bins per driver.
min_share	Minimum share of training rows per bin.
min_n	Minimum number of training rows per bin.
monotone	"auto" (direction from the Spearman sign), "increasing", "decreasing" or "none".
scale	"mean" (bin mean in the woe slot) or "logit".
nthread	Parallel workers by driver, through the package backend.
alpha	Alpha of the PSI critical value in the revalidation.

Details

Numeric drivers must not contain missing values: run scr_triage() (or impute) first, exactly as the scorecard pipeline does. Categorical missing values become the level "NA", as in the engine. When a holdout_idx is given, the frozen bins are revalidated: the hold-out bin means are recomputed, the PSI of the bin shares is reported with the sample-size-adjusted critical value, a driver whose hold-out means break the training order is flagged UNSTABLE_HOLDOUT, one whose bin shares shift (fixed PSI flag "shift", PSI at or above 0.25) is flagged PSI_ACTION, and one with more than 1% of hold-out rows outside the bins UNBINNED_HOLDOUT.

Value

An object of class scr_cbins: fit (the obwoe-shaped object), summary (one row per driver: feature, type, n_bins, eta2, direction, converged, and after revalidation eta2_holdout, psi, psi_flag, holdout_ok, holdout_reason), holdout (bin table per driver with train and hold-out means), scale and target. summary keeps the engine columns (algorithm, total_iv, iterations, error) and, after revalidation, psi_critical, psi_flag_adjusted and pct_unbinned.

See Also

Other irb-ead: scr_ead(), scr_ead_data(), scr_ead_downturn(), scr_ead_validate()

Examples

```
set.seed(1)
d <- data.frame(x = runif(600), g = sample(c("a", "b", "c", "d"), 600, TRUE))
d$y <- pmin(1, pmax(0, 0.2 + 0.6 * d$x + (d$g == "d") * 0.2 + rnorm(600, 0, 0.1)))
cb <- scr_bin_continuous(d, "y", c("x", "g"), train_idx = 1:400, holdout_idx = 401:600)
cb
cb$fit$results$x$bin
cb$fit$results$x$woe    # bin means of y
```

`scr_calibrate` *Calibrate the alignment to a central tendency*

Description

Re-anchors the probability of default of a scorecard to a long-run average default rate (the central tendency, CT) without touching the points: the result is a **new** alignment (`I*`, `S*`) such that `predict(alignment, raw, type = "prob")` is the calibrated PD, while the scorecard keeps its own alignment for the score. Four methods:

Usage

```
scr_calibrate(
  x,
  target,
  sample_rate = NULL,
  method = NULL,
  ar_target = NULL,
  segment = NULL,
  raw = NULL,
  y = NULL,
  sample = "holdout"
)
```

Arguments

`x`	An `scr_scorecard()` (uses the ln(odds) and outcome of `sample`), an `scr_align()` (pass `raw` and, for the two-parameter methods, `y`) or a numeric vector of event ln(odds) (aligned directly to the default 600/50/20 scale).
`target`	The central tendency: a number in `(0, 1)` or an `scr_dr` from `scr_default_rate()` (its `lra$mean` is used). With `segment`, a named vector with one CT per segment.
`sample_rate`	Event rate of the calibration sample; `NULL` uses the mean of `y`.
`method`	`"intercept"`, `"logodds_ab"`, `"scaling"` or `"qmm"`; `NULL` uses `config$pd_calibration`.
`ar_target`	Target accuracy ratio for `"logodds_ab"` and `"qmm"`.
`segment`	Optional vector of segment labels, one per calibration row: one alignment per segment is fitted as well.
`raw`, `y`	Raw ln(odds) and 0/1 outcome when `x` is not a scorecard.
`sample`	Sample of the scorecard used for the calibration.

Details

`"intercept"` The prior-correction shift of King and Zeng (2001), $\delta = \ln[\tau(1-\bar{y})/((1-\tau)\bar{y})]$, added to the event ln(odds); `S` unchanged, so the rank order and every discrimination statistic are untouched. The closed form is exact on the odds; when the calibration sample is available the shift is refined by a one-dimensional root so that the mean PD equals the CT exactly (the closed form is reported as `shift_prior`).

`"logodds_ab"` Tasche (2013): `ln(odds*) = a + b ln(odds)`, with `(a, b)` solving `mean(PD*) = CT` and implied accuracy ratio equal to `ar_target` (default: the accuracy ratio observed on the sample). With the observed accuracy ratio this is Tasche's quasi-moment matching (QMM) proper. The implied AUC is the probability that a default has a higher PD than a non-default when the PDs are true: each score carries weight `PD*` among the defaults and `1 - PD*` among the non-defaults, ties counted one half.

`"qmm"` The outcome-free variant of the same two equations: the target accuracy ratio is the implied AR of the current PDs, so no outcome is needed and the implied discriminatory power of the uncalibrated curve is carried over to the new level.

`"scaling"` `PD* = PD * CT / ybar`. The proportional rescaling is not a logit map, so the slope is the least-squares projection of `logit(PD*)` on the ln(odds) and the intercept is solved to the CT.

Value

An object of class `scr_pd_calibration`: `alignment` (the new `scr_align`), `alignment_before`, `method`, `ct`, `target_source`, `sample_rate`, `shift` (change of the event intercept), `shift_prior` (the closed-form King-Zeng shift), `slope_ratio` (`S* / S`), `mean_pd_before`, `mean_pd_after`, `ar_before`, `ar_after` (observed), `ar_implied_before`, `ar_implied_after`, `n`, `segments` (table and alignments when `segment` is given), `ledger`. Also `ar_target`, `note` and `sample`.

References

King, G. and Zeng, L. (2001). Logistic regression in rare events data. *Political Analysis*, 9(2), 137-163.

Tasche, D. (2013). The art of probability-of-default curve calibration. *Journal of Credit Risk*, 9(4), 63-103.

See Also

Other irb-pd: `predict.scr_grades()`, `predict.scr_pd()`, `scr_grades()`, `scr_master_scale()`, `scr_migration()`, `scr_moc()`, `scr_pd()`, `scr_pd_pit_ttc()`, `scr_pd_validate()`

Examples

```
set.seed(1)
l <- stats::qlogis(0.12) + stats::rnorm(2000)
y <- stats::rbinom(2000, 1, stats::plogis(l))
cal <- scr_calibrate(l, target = 0.04, y = y)
cal
mean(predict(cal$alignment, l, type = "prob"))
scr_calibrate(l, target = 0.04, y = y, method = "logodds_ab", ar_target = 0.55)
```

scr_capital *Expected loss, risk-weighted assets and capital of a portfolio*

Description

Runs scr_irb_rw() on every exposure, aggregates by segment, compares the IRB result with the standardized approach for the output floor, reconciles regulatory expected loss with the provision stock (shortfall deducted from capital; excess eligible as tier 2 up to 0.6 % of the IRB risk-weighted assets), measures the impact of each input floor, runs a fixed sensitivity grid and reports the name concentration of the book. The parameter tables come from params; the object records whether they were edited.

Usage

```
scr_capital(
  x,
  pd = "pd",
  lgd = "lgd",
  ead = "ead",
  segment = NULL,
  asset_class = config$asset_class,
  m = NULL,
  defaulted = NULL,
  elbe = NULL,
  provisions = NULL,
  ltv = NULL,
  rating = NULL,
  sales = NULL,
  fi = NULL,
  transactor = NULL,
  grade = NULL,
  id = NULL,
  claim = NULL,
  granular = TRUE,
  params = scr_irb_params(config$framework),
  config = scr_config(),
  keep_rows = FALSE
)
```

Arguments

x	A table of exposures (data.frame or data.table) or the list form described above.
pd, lgd, ead	Column names of the probability of default, loss given default and exposure at default.
segment	Optional column name of the reporting segment.

asset_class — A column name or a single asset class (see scr_irb_rw()).

m, defaulted, elbe, provisions, ltv, rating, sales, fi, transactor, grade, id — Optional column names: effective maturity, default flag, best estimate of expected loss, provision stock, loan-to-value, external rating, annual sales, financial-institution flag, transactor flag, PD grade (defines the SQL pools together with segment) and exposure identifier.

claim — Optional column name: the claim type of each exposure under the foundation approach (a row of params$lgd_firb); the supervisory LGD then replaces lgd.

granular — TRUE, FALSE or a column name: whether the retail exposures belong to a granular regulatory retail pool (the standardized comparison uses the non-granular weight otherwise).

params — An scr_irb_params() object; defaults to the preset of config$framework.

config — An scr_config() object (capital_approach, capital_target_ratio, capital_output_floor, capital_sensitivity, nthread, verbose).

keep_rows — Keep the per-exposure table in the object.

Value

An object of class scr_capital: a list with exposures (per-exposure table, only with keep_rows = TRUE), segments (the reconciliation table: segment, n, ead, pd_mean, lgd_mean, m, r_mean, k_mean, rw, rwa_irb, rwa_sa, irb_sa_ratio, el, provisions, shortfall_excess), pools (one row per segment and grade with the constants the SQL emits), totals (n, ead, el, rwa_irb, rwa_sa, irb_sa_ratio, output_floor, rwa_floor, rwa_reported, floor_binding, headroom, density, target_ratio, capital, provisions, shortfall, excess, tier2_addback, tier2_cap, hhi, n_eff, max_share, granular), floors (floor, n_hit, ead_hit, delta_rwa), sensitivity (shock, rwa, delta, delta_pct), concentration (share of EAD and RWA by segment), framework, approach, params, config, ledger, model_card and, after scr_export(), files. segments and totals also carry n_defaulted; totals also el_rate and rwa_irb_no_floors; concentration has segment, n, ead, rwa, ead_share, rwa_share and hhi_contribution; columns records the column names the SQL reads.

Inputs

x is either a table of exposures, the remaining arguments naming its columns, or a list list(pd = , lgd = , ead = , data =) whose elements are fitted models with an scr_apply() method (the PD, LGD and EAD objects of the IRB modules) and data the table to apply them to. In the list form each model present fills the corresponding vector from the columns pd_final, lgd_final and ead_predicted of its scr_apply() output, and the provenance is written to the ledger; elements that are NULL fall back to the named columns of data.

asset_class is a column name of x or a single class applied to every row. Segment means are weighted by EAD. The sensitivity grid shocks the PD (x1.10, x1.25, x1.50), the LGD (+5 percentage points), the EAD (+10 %), removes the input floors, scales the correlation (x1.25) and stresses the PD with the one-factor model at q = 0.95 and 0.99 (scr_pd_stress(), the stressed PD then re-entering the function so that the correlation follows it).

References

Basel Committee on Banking Supervision (2023). *The Basel Framework*, CRE31, CRE35 (treatment of expected losses and provisions), RBC20 (output floor).

See Also

Other irb-capital: `scr_ecl()`, `scr_el()`, `scr_irb_rw()`, `scr_pd_stress()`, `scr_sa_rw()`

Examples

```
cfg <- scr_config(verbose = FALSE)
cap <- scr_capital(scr_demo_portfolio, segment = "segment", asset_class = "asset_class",
                m = "m", defaulted = "defaulted", elbe = "elbe", provisions = "provision",
               ltv = "ltv", rating = "rating", sales = "sales", transactor = "transactor",
                   grade = "grade", id = "id", config = cfg)
cap
cap$segments[, c("segment", "n", "rw", "irb_sa_ratio")]
cap$floors
```

---

scr_claims *Probability statements about the event rate of score groups*

Description

Tests claims such as "the customers scoring 625 or more respond at a rate of at least 60%" or "tier 'low' defaults at no more than 2%" on a study sample, and writes each verdict as one English sentence with the observed rate and its one-sided bound.

Usage

```
scr_claims(x, claims, ...)

## S3 method for class 'scr_study'
scr_claims(
  x,
  claims,
  level = NULL,
  adjust = c("holm", "none"),
  type = c("average", "floor"),
  sample = NULL,
  floor_bins = 10L,
  ...
)

## S3 method for class 'scr_scorecard'
scr_claims(
  x,
```

```
  claims,
  level = NULL,
  adjust = c("holm", "none"),
  type = c("average", "floor"),
  sample = "holdout",
  reference = "train",
  n_bands = NULL,
  max_cells = 1e+05,
  floor_bins = 10L,
  ...
)

## S3 method for class 'data.frame'
scr_claims(
  x,
  claims,
  level = 0.95,
  adjust = c("holm", "none"),
  type = c("average", "floor"),
  sample = NULL,
  score = "score",
  y = "y",
  objective = "risk",
  direction = NULL,
  weight = NULL,
  reference = NULL,
  study = NULL,
  counts = FALSE,
  n = "n",
  events = "events",
  n_bands = 20L,
  max_cells = 1e+05,
  floor_bins = 10L,
  ...
)
```

Arguments

x	An object from scr_bands() or scr_tiers(), an object from scr_scorecard(), or a data.frame with one row per scored case (or one row per score value with counts = TRUE).
claims	A data.frame of claims; see the section Claims.
...	Passed on to the methods; an unknown argument is an error.
level	Confidence level of the tests and of the one-sided bounds. For a scorecard, NULL uses config$study_level (0.95); for a score study, NULL uses the level of the study.
adjust	"holm" (default) or "none": multiplicity adjustment across the claims.

type	"average" (the rate of the whole group) or "floor" (the rate at the weakest end of the group under a monotone fit of the rate); see the section Average and floor.
sample	For a score study: the sample the claims are read on, NULL for its first study sample. For a scorecard: that sample ("holdout"). For a data.frame: the name of a column with sample labels, as in scr_bands(); the claims are read on study.
floor_bins	Pre-bins of equal share of a group for type = "floor": the resolution of the floor (default 10; 1 makes the floor the average).
reference	For a scorecard: the sample the bands are frozen on ("train"). For a data.frame: the label of the reference sample; NULL takes the first level of the sample column (the levels of a factor in their order, numbers in numeric order, text sorted).
n_bands	Bands of the internal percentile study (their labels and numbers can be used in claims$label). For a scorecard, NULL uses config$study_bands (20).
max_cells	Largest number of distinct score values kept exactly.
score, y	Column names of the score and of the 0/1 outcome (NA allowed).
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).
direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
weight	Optional column of non-negative case weights.
study	For a data.frame: the label of the sample the claims are read on; NULL takes the first label other than the reference.
counts	TRUE when x is pre-aggregated: one row per score value with the columns score, n and events (and, optionally, value and value_events).
n, events	Column names of the counts when counts = TRUE.

Value

An object of class c("scr_claims", "list"):

table One row per claim: name, group (the label or the score range), type, score_lo and score_hi (the range tested), n (rows with a known outcome; their weighted volume under weights), n_eff (Kish effective size, the n of the test; equal to n without weights), events, rate, bound (one-sided Jeffreys bound), op, rate_claimed, p_value and p_adj (test of the claim), p_refute and p_refute_adj (the opposite test), verdict and statement (under weights, the sentence quotes the effective n of the test next to the weighted volume).

sample, reference, level, adjust, type, floor_bins, objective, direction, target, call The settings (floor_bins is used by type = "floor" only).

Claims

claims is a data.frame with one row per claim:

op ">=" (the event rate is at least rate) or "<=" (at most rate).

rate The claimed event rate, in (0, 1).

label A band or tier of the study: its label (the label column of the study table, or the numbered tier_label of a tiers study) or its number (the band or tier column).

`score_lo`, `score_hi` Instead of a label, a score range `[score_lo, score_hi)`; a missing end is open. A claim with neither a label nor a score range is on the whole sample.

`name` Optional: a name for the printed statement.

Test

Each claim is read on the rows of the group with a known outcome in the study sample. With weights, the counts are taken to the Kish effective size $n = (\sum w)^2 / \sum w^2$ and $x = \hat{p}\, n$ effective events (without weights, the counts themselves). The claim "rate >= r" is the alternative to H_0 : $p \leq r$, tested by the exact one-sided binomial p-value $P(X \geq x)$, $X \sim \text{Binomial}(n, r)$; x and n are rounded to whole numbers for the binomial only. The refutation is the opposite test, $P(X \leq x)$. For "rate <= r" the two tests swap. With `adjust = "holm"`, the claim p-values and the refutation p-values are each Holm-adjusted across the claims. They are two Holm families, the tests of the claims and the tests of the refutations, each controlled at `1 - level`; a claim and its refutation can never both be significant.

The verdict is `"supported"` when the adjusted p-value of the claim is below `1 - level`, `"refuted"` when the adjusted p-value of the refutation is, and `"not proven"` otherwise (also for a group without a row with a known outcome). `bound` is the one-sided Jeffreys bound at `level` in the direction of the claim, on the unrounded effective counts: the lower bound, the Beta$(x + 1/2, n - x + 1/2)$ quantile `1 - level`, for "rate >= r" (0 when $x = 0$), and the upper bound, its quantile `level`, for "rate <= r" (1 when $x = n$). The bound describes the uncertainty; the verdict comes from the tests.

Average and floor

`type = "average"` reads the claim on the event rate of the whole group. `type = "floor"` reads it on the weakest end of the group under a monotone fit of the rate: a claim that holds there holds for the part of the group where it is hardest to meet, not only for the average.

The reference rows of the group are cut into `floor_bins` pre-bins of equal share (tie-safe, as the bands of `scr_bands()`), and their event rates are fitted by pool adjacent violators: the rate is made monotone along the score, rising toward its event-rich end. The weakest end is the run of pre-bins with the lowest fitted rate of the group for "rate >= r", with the highest for "rate <= r". Its score range is found on the reference sample and the claim is tested on the rows of the study sample in that range, so the end is not chosen on the rows it is tested on (unless the study sample is the reference). When the fitted rate is flat over the group, the weakest end is the whole group.

Two limits follow from the fit. A dip of the rate inside the group is pooled with its neighbors by the monotone fit, so the floor does not see a weak pocket in the middle of the group, only the end the fit calls weakest. And `floor_bins` sets the resolution: the floor speaks for runs of pre-bins, about one part in `floor_bins` of the group or more, never for a single customer (a fit on single score values would spike at its ends and read the floor on a handful of rows).

Input

An object from `scr_bands()` or `scr_tiers()` is read as it is: labels refer to its bands or tiers, and `sample` picks one of its samples (by default its first study sample). A scorecard or a `data.frame` is first summarized by a percentile study (`scr_bands()`, `n_bands` bands frozen on the reference, no bootstrap) whose count table keeps every distinct score up to `max_cells`; the finite ends of the score ranges of `claims` are forced as cell edges, so a score range is exact even when more distinct

scores are pooled into cells. A score range read on a study whose cells were pooled must not cut through a cell, or the call is an error.

References

Brown, L. D., Cai, T. T. and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133. doi:10.1214/ss/1009213286

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2), 65-70.

Kish, L. (1965). *Survey Sampling*. Wiley.

See Also

`scr_bands()` and `scr_tiers()` for the groups, `scr_operating()` to choose a cut under constraints.

Other score-studies: `scr_bands()`, `scr_detection()`, `scr_maturity()`, `scr_mix_shift()`, `scr_operating()`, `scr_overlap()`, `scr_rag()`, `scr_rag_plan()`, `scr_score_cross()`, `scr_segments()`, `scr_tiers()`, `scr_uplift()`

Examples

```
set.seed(1)
x <- rnorm(4000)
d <- data.frame(score = round(500 + 50 * x),
                y = rbinom(4000, 1, plogis(-0.5 + 1.2 * x)))
claims <- data.frame(op = c(">=", ">=", "<="), rate = c(0.60, 0.80, 0.25),
                     score_lo = c(550, 550, NA), score_hi = c(NA, NA, 450),
                     name = c("top converts", "top converts well", "bottom is cold"))
cl <- scr_claims(d, claims, objective = "propensity")
cl
cl$table[, c("name", "n", "rate", "bound", "p_adj", "verdict")]

# the same claims at the weakest end of each group, not only on its average
scr_claims(d, claims, objective = "propensity", type = "floor")$table$verdict

# claims on the tiers of a study, by label
tr <- scr_tiers(d, objective = "propensity", n_tiers = 3)
scr_claims(tr, data.frame(label = "high", op = ">=", rate = 0.5))
```

---

scr_classing_accept *Accept or discard a proposal*

Description

`reason` is mandatory. Accepting replaces the variable's current bins; the previous accepted proposal is marked `superseded`. A `BLOCKED` proposal needs `override = TRUE`, and the override is itself a ledger row.

Usage

```
scr_classing_accept(lab, proposal, reason, override = FALSE)

scr_classing_discard(lab, proposal, reason)
```

Arguments

lab	An object from scr_coarse_classing().
proposal	An object from scr_classing_propose().
reason	Free text, at least 5 characters.
override	Accept a BLOCKED proposal.

Value

The updated lab, invisibly.

See Also

scr_coarse_classing() for a complete session, from lab to scorecard.

Other classing: scr_classing_apply(), scr_classing_choose(), scr_classing_propose(), scr_classing_spec(), scr_classing_view(), scr_coarse_classing(), scr_decisions()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
p <- scr_classing_propose(lab, "ds_region",
                          groups = list(edge = c("NORTH", "SOUTH"),
                                        core = c("EAST", "WEST", "CENTRE")))
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
scr_classing_view(lab, "ds_region")
# a second proposal on the same variable, rejected with its reason
p2 <- scr_classing_propose(lab, "ds_region",
                           groups = list(c("NORTH", "SOUTH", "EAST"), c("WEST", "CENTRE")))
lab <- scr_classing_discard(lab, p2, reason = "no business rationale for this grouping")
scr_decisions(lab)
```

scr_classing_apply *Commit the lab into a new selection result*

Description

Returns a new scr_result in which the accepted manual entries replace the optimal ones inside fit (the automatic fit is frozen as fit_auto), the screening and hold-out rows of those variables are recomputed with the very same pipeline functions, the final shortlist is the one implied by scr_classing_choose(), and the funnel, gains, SQL and summary are rebuilt with a provenance column. scr_selected() on the result returns the final list (which = "consensus" still gives the automatic one). The ledger travels with the result and into scr_scorecard() and scr_export(). The input result is not modified.

Usage

```
scr_classing_apply(lab)
```

Arguments

lab	An object from scr_coarse_classing().

Value

An scr_result with a lab component (ledger, spec, shortlist, source).

See Also

scr_coarse_classing() for a complete session, from lab to scorecard.

Other classing: scr_classing_accept(), scr_classing_choose(), scr_classing_propose(), scr_classing_spec(), scr_classing_view(), scr_coarse_classing(), scr_decisions()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
p <- scr_classing_propose(lab, "ds_region",
                          groups = list(edge = c("NORTH", "SOUTH"),
                                        core = c("EAST", "WEST", "CENTRE")))
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
res2 <- scr_classing_apply(lab)
scr_selected(res2)
scr_decisions(res2)
```

scr_classing_choose *Choose the final variable list manually*

Description

The final list is (consensus shortlist + force) - drop, then intersected with keep when given. force is allowed only for variables that reached binning; a variable failed for IV_SUSPICIOUS (the leakage ceiling) and a derived __sp flag under allow_derived_final = FALSE are refused unless override = TRUE. reason is one string for every variable named, or a character vector named by variable.

Usage

```
scr_classing_choose(
  lab,
  keep = NULL,
  drop = NULL,
  force = NULL,
  reason = NULL,
  override = FALSE
)
```

Arguments

lab	An object from scr_coarse_classing().
keep	Variables to keep (restricts the final list).
drop	Variables to remove from the final list.
force	Variables to add to the final list.
reason	Mandatory when drop or force is given.
override	Allow a refused force.

Value

The updated lab, invisibly.

See Also

scr_coarse_classing() for a complete session, from lab to scorecard.

Other classing: scr_classing_accept(), scr_classing_apply(), scr_classing_propose(), scr_classing_spec(), scr_classing_view(), scr_coarse_classing(), scr_decisions()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
lab <- scr_classing_choose(lab, drop = "vl_score_10",
                           reason = "not available at decision time")
lab
```

`scr_classing_propose` *Propose manual bins for a variable*

Description

Exactly one of `breaks`, `groups`, `merge`, `split` or `reset` per call; `missing_to` and `other_to` compose with a categorical instruction. The instruction is resolved against the current bins into an absolute specification, the WOE is refitted on the training rows, the bins are applied frozen to the hold-out, and the comparison against the optimal bins is printed. The proposal is a value: nothing changes in the lab until `scr_classing_accept()`.

Usage

```
scr_classing_propose(
  lab,
  variable,
  breaks = NULL,
  groups = NULL,
  merge = NULL,
  split = NULL,
  missing_to = NULL,
  other_to = NULL,
  reset = FALSE
)
```

Arguments

`lab`	An object from `scr_coarse_classing()`.
`variable`	A variable of the lab.
`breaks`	Numeric: interior cut points, `(-Inf, b1]`, `(b1, b2]`, ...
`groups`	Categorical: a list of character vectors, one per bin; names are display labels. Every training category must be assigned, or `other_to` must name the catch-all bin.
`merge`	Bin ids to merge (adjacent for numerics).
`split`	`c(id, at)`: split numeric bin `id` at `at`.

missing_to	Categorical: fold the "MISSING" category into this bin.
other_to	Categorical: the bin that receives every training category not listed in groups.
reset	TRUE proposes a return to the optimal bins.

Value

An scr_classing_proposal with id, variable, instruction (the resolved instruction as text), entry (the hand-built bins), checks, optimal (the checks of the optimal bins), compare, verdict (ACCEPTABLE, REVIEW or BLOCKED), warnings and blocking.

See Also

scr_coarse_classing() for a complete session, from lab to scorecard.

Other classing: scr_classing_accept(), scr_classing_apply(), scr_classing_choose(), scr_classing_spec(), scr_classing_view(), scr_coarse_classing(), scr_decisions()

---

scr_classing_spec *Classing specification as a long table, with its file round trip*

Description

One row per bin of every variable in the lab, optimal and manual, with the authoritative columns a reviewer may edit (lower/upper for numerics, categories/is_other for categoricals, reason) and context columns that are regenerated on read. Open ends are written as NA. scr_classing_read() validates a file back into a spec and scr_classing_import() turns every variable whose bins differ from the lab's current ones into a proposal, so a spreadsheet edit never enters silently.

Usage

```
scr_classing_spec(lab, file = NULL)

scr_classing_read(file, sep = "%;%")

scr_classing_import(lab, file)
```

Arguments

lab	An object from scr_coarse_classing(), or an scr_result returned by scr_classing_apply().
file	For scr_classing_spec(), an optional .csv or .xlsx path to write the table to. For scr_classing_read(), the path to read. For scr_classing_import(), a path or an scr_classing_spec object.
sep	Bin separator of the categories column (the configuration's bin_separator). It is validated (no empty category, no category in two bins) and recorded on the spec, so that scr_classing_import() refuses a spec read with a different separator.

Value

A `data.frame` of class `scr_classing_spec`.

`scr_classing_import()` returns a named list of proposals (one per variable whose bins differ from the lab's current ones), each to be accepted or discarded.

See Also

`scr_coarse_classing()` for a complete session, from lab to scorecard.

Other classing: `scr_classing_accept()`, `scr_classing_apply()`, `scr_classing_choose()`, `scr_classing_propose()`, `scr_classing_view()`, `scr_coarse_classing()`, `scr_decisions()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
p <- scr_classing_propose(lab, "ds_region",
                          groups = list(edge = c("NORTH", "SOUTH"),
                                        core = c("EAST", "WEST", "CENTRE")))
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
sp <- scr_classing_spec(lab)
sp
# round trip through a file: a fresh lab receives the manual bins as proposals
f <- tempfile(fileext = ".csv")
scr_classing_spec(lab, file = f)
props <- scr_classing_import(scr_coarse_classing(res), scr_classing_read(f))
names(props)
unlink(f)
```

---

scr_classing_view *Inspect the current bins of a variable in the lab*

Description

Prints the current bins (optimal, or the accepted manual ones) with train and hold-out side by side and a text bar chart of the event rate, or, without `variable`, one line per variable of the lab.

Usage

```
scr_classing_view(lab, variable = NULL)
```

Arguments

`lab`	An object from `scr_coarse_classing()`.
`variable`	A variable name, or `NULL` for the overview.

Value

Invisibly, the bins table (`variable` given) or the overview table.

See Also

`scr_coarse_classing()` for a complete session, from lab to scorecard.

Other classing: `scr_classing_accept()`, `scr_classing_apply()`, `scr_classing_choose()`, `scr_classing_propose()`, `scr_classing_spec()`, `scr_coarse_classing()`, `scr_decisions()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
scr_classing_view(lab)
scr_classing_view(lab, "ds_region")
```

`scr_coarse_classing` *Coarse classing lab: manual binning and manual variable choice*

Description

Opens a lab on an `scr_select()` result. Inside it the analyst inspects the optimal bins of any binned variable (`scr_classing_view()`), proposes new breaks or groupings (`scr_classing_propose()`), reads the comparison against the optimal bins, accepts or discards each proposal with a mandatory reason (`scr_classing_accept()`, `scr_classing_discard()`), chooses the final variable list (`scr_classing_choose()`) and commits everything to a new `scr_result` (`scr_classing_apply()`) that the rest of the pipeline consumes unchanged: `scr_scorecard()`, `scr_apply()`, `scr_sql()`, `scr_export()`.

Usage

```
scr_coarse_classing(
  x,
  features = NULL,
  laplace = 0,
  max_iv_loss = NULL,
  author = Sys.info()[["user"]]
)
```

Arguments

x	An object from scr_select().
features	Variables the lab covers. Default: every variable that reached binning (names(xfitresults)), so a variable failed by screening can be rebinned and forced in with a reason.
laplace	Smoothing added to the bin counts when recomputing WOE. 0 (default) is exactly the engine's formula.
max_iv_loss	Advisory threshold: a manual bin whose hold-out IV falls more than this fraction below the optimal one raises IV_LOSS_VS_OPTIMAL. NULL uses config$lab_max_iv_loss.
author	Free text recorded in the ledger.

Value

An scr_classing object (the lab), with a print method that summarizes the session: variables touched, before/after IV, verdicts, reasons, pending proposals and the final choice.

Contract of a manual bin

A manual bin is recomputed **on the training rows only** (hold-out rows can never define a bin), with the engine's own WOE formula (ln(%event / %non-event), event-oriented, so glm coefficients stay positive), then revalidated on the hold-out with the bins frozen (IV, PSI with both thresholds, unbinned share) and screened with the eight engine rules, so the lab and the pipeline can never disagree. Numeric intervals are right-closed, (a, b], exactly as the engine and its SQL. Re-declaring the optimal cut points of a numeric reproduces the engine's WOE exactly; for a categorical the engine applies a small internal smoothing of its own, so the raw log-ratio of the lab differs from it in the third decimal.

What is never allowed silently

An empty bin, a degenerate bin (no events or no non-events, unless laplace > 0), a bin below lab_min_bin_pct_hard, a manual IV crossing iv_max (the lab must not manufacture leakage), a category left unassigned, a missing reason. Those block the proposal (BLOCKED); accepting one needs override = TRUE, and the override is itself a ledger row.

See Also

Other classing: scr_classing_accept(), scr_classing_apply(), scr_classing_choose(), scr_classing_propose(), scr_classing_spec(), scr_classing_view(), scr_decisions()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
lab <- scr_coarse_classing(res)
lab
scr_classing_view(lab, "ds_region")
p <- scr_classing_propose(lab, "ds_region",
                          groups = list(edge = c("NORTH", "SOUTH"),
```

```
                                        core = c("EAST", "WEST", "CENTRE")))
p
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
lab <- scr_classing_choose(lab, drop = "vl_score_10",
                           reason = "not available at decision time")
lab
res2 <- scr_classing_apply(lab)
scr_selected(res2)
scr_selected(res2, which = "consensus")
sc <- scr_scorecard(res2)
sc$model_card$binning_algorithm
```

`scr_compare` *Compare runs across targets*

Description

One row per target, with the funnel, the hold-out performance of the best model (with CI) and the warning signs.

Usage

```
scr_compare(x)
```

Arguments

`x`	An object from `scr_run()`, or a named list of `scr_result`.

Value

A `data.table` with one row per successful target.

See Also

Other portfolio: `scr_core()`, `scr_run()`, `scr_runset`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
r1 <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                 date_col = "ref_date")
r2 <- scr_select(scr_demo, "churn", config = cfg, drop = c("id", "default"),
                 date_col = "ref_date")
scr_compare(list(default = r1, churn = r2))
scr_core(list(default = r1, churn = r2), min_targets = 2)
```

`scr_config` *Pipeline configuration*

Description

Builds the configuration object that crosses every stage, from `scr_split()` to `scr_export()`. A preset sets the tightness of the selection funnel; any individual key can be overridden through `...`.

Usage

```
scr_config(preset = c("moderate", "aggressive", "lazy"), ...)
```

Arguments

`preset`	One of `"moderate"` (default), `"aggressive"` or `"lazy"`.
`...`	Overrides of any configuration key, by name. `NULL` means "keep the preset value", not "delete the key". An unknown name is an error, on purpose: a silent override leaves dead configuration in the file.

Value

An object of class `scr_config`: a named list with every key resolved.

Presets

A preset touches four keys and nothing else: `target_max`, `min_votes`, `corr_cutoff` and `iv_min`.

preset	variables at the end	`min_votes`	`corr_cutoff`	`iv_min`
`"aggressive"`	10 to 15	3	0.60	0.03
`"moderate"`	10 to 25	2	0.70	0.02
`"lazy"`	10 to 40	1	0.80	0.02

Use `scr_presets()` to see the resolved table and `scr_config_keys()` for the full key dictionary.

Risk or propensity (`objective`)

The two literatures use the same mathematics with opposite conventions. In credit and fraud, `target = 1` is the **bad** case and the scorecard is built so that more points mean less risk. In propensity, `target = 1` is the **good** case and the campaign list needs more points to mean a higher chance of engaging.

	`"risk"` (default)	`"propensity"`
`target = 1` means	undesirable event	desirable event
Points scale	more points = safer	more points = more likely
Derived `direction`	`"higher_is_safer"`	`"higher_is_riskier"`
`odds_orientation`	`safe:event`	`event:safe`

`objective` **does not touch the selection.** It does not change the modeled target, the cut points, the IV or the shortlist; it acts on the direction of the points scale, on the odds orientation of the alignment and on the vocabulary of the reports. To model the other class as the event, the argument is `event_level` in `scr_split()` and `scr_select()`, and that one, unlike this, rewrites everything.

Binning algorithm (`algorithm`)

`"jedi"` is the default and stays exposed side by side with the alternatives, never hidden behind an `"auto"`. Choices with distinct properties: `"ivb"`, `"dp"` and `"sblp"` are provably optimal for categoricals; `"cm"` (ChiMerge) and `"fetb"` have a principled stopping rule; `"ir"` (isotonic) guarantees monotone WOE; `"fast_mdlp"` is the faithful Fayyad-Irani. The full list is in `OptimalBinningWoE::obwoe_algorithms()`. A numeric-only or categorical-only algorithm applies where it is valid and the other type falls back to `"jedi"`.

The Information Value gate

`iv_min` Admission floor. Fails with `IV_BELOW_MIN`.

`iv_max` Admission ceiling. Fails with `IV_SUSPICIOUS`. `1.00` (default) tolerates a legitimately strong predictor and cuts the absurd; `0.50` is the engine default, calibrated for credit default.

`iv_suspect` Only the threshold of the report warning. Fails nothing.

The real leakage detector is `allow_degenerate = FALSE`: a bin with no events or no non-events is the symptom that has no innocent explanation.

Scorecard scale

`base_score`, `base_odds` and `pdo` are one statement: *at* `base_score` *points the odds are* `base_odds`*, and every* `pdo` *points they double*. `base_odds` is always expressed in the orientation `direction` implies (non-event:event under `higher_is_safer`; event:non-event under `higher_is_riskier`), and the alignment object records `odds_orientation` so this is never implicit. The classic 600/50/20 is Siddiqi's (2006) textbook example, not a parameter published by any bureau. `align_method = "regression"` (default) takes the raw score to that scale by regressing empirical log-odds on score bands, which absorbs reweighting, miscalibration and prior shift; `"direct"` assumes the model is calibrated and uses the logit as is.

References

Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.

See Also

`scr_select()` to use the configuration, `scr_presets()` to compare presets, `scr_config_keys()` for the key dictionary.

Other configuration: `scr_config_keys()`, `scr_presets()`, `scr_verbose()`

Examples

```
cfg <- scr_config()
cfg

# propensity: more points = more likely to have the event
scr_config(objective = "propensity")$objective

# NULL keeps the preset value (here, iv_min = 0.03 from aggressive)
scr_config("aggressive", iv_min = NULL)$iv_min

# a wrong name fails loudly instead of becoming dead configuration
try(scr_config(iv_maximum = 1))
```

---

scr_config_keys *Dictionary of configuration keys*

Description

One row per `scr_config()` key, with the stage it acts on, the default value and what it controls.

Usage

```
scr_config_keys(stage = NULL)
```

Arguments

`stage`	Optional filter by stage: `0` to `7` for the scorecard pipeline, `8` to `12` for the IRB models, `13` for the score studies. `NULL` returns everything.

Value

A `data.frame` with `key`, `stage`, `default` and `description`.

See Also

Other configuration: `scr_config()`, `scr_presets()`, `scr_verbose()`

Examples

```
head(scr_config_keys(), 8)
scr_config_keys(stage = 5)
```

scr_connect *Connect to a database (ODBC DSN or any DBI driver)*

Description

With dsn, a thin wrapper around DBI::dbConnect() over odbc::odbc() with one deliberate choice: bigint = "numeric". Under the **odbc** default a BIGINT column arrives as integer64, and is.numeric() of an integer64 is FALSE: typing would treat the column as a categorical of very high cardinality. With driver, any DBI driver object is accepted (e.g. RSQLite::SQLite(), duckdb::duckdb()), which is how the database path is tested without a DSN.

Usage

```
scr_connect(dsn = NULL, driver = NULL, timeout = 20, ...)
```

Arguments

dsn	Name of the DSN configured on the system. Ignored when driver is given.
driver	Optional DBI driver object, used instead of ODBC.
timeout	Connection timeout in seconds (ODBC only).
...	Extra arguments passed on to DBI::dbConnect() (for example dbname = ":memory:" for SQLite).

Value

A DBI connection. Close it with DBI::dbDisconnect().

See Also

Other database: scr_fetch()

Examples

```
con <- scr_connect(driver = RSQLite::SQLite(), dbname = ":memory:")
d <- scr_demo; d$ref_date <- as.character(d$ref_date)   # SQLite has no Date type
DBI::dbWriteTable(con, "dtm", d)
dt <- scr_fetch(con, "dtm", sample_frac = 0.5, seed = 42)
nrow(dt)
DBI::dbDisconnect(con)
```

scr_core *Variables that cross several targets*

Description

Which variables were approved on how many targets. A stable core across targets is the best argument in favor of a variable.

Usage

```
scr_core(x, min_targets = 2L)
```

Arguments

x	An object from scr_run(), or a named list of scr_result.
min_targets	Minimum number of targets to enter the result.

Value

A data.table with feature, n_targets, targets and mean_rank.

See Also

Other portfolio: scr_compare(), scr_run(), scr_runset

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
r1 <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                 date_col = "ref_date")
r2 <- scr_select(scr_demo, "churn", config = cfg, drop = c("id", "default"),
                 date_col = "ref_date")
scr_core(list(default = r1, churn = r2), min_targets = 2)
```

scr_cutoff *Stage 6: cut-off sweep with frozen cuts*

Description

For each candidate cut, what happens in each sample: the fraction of the population on the safe side (approval), the event rate on both sides, the events avoided (share of events falling on the risky side), the non-events lost and the KS at the cut. The candidate cuts are quantiles of the score **on train**, applied frozen to the hold-out: both samples answer on the same numbers, and the comparison between them measures the stability of the decision, not a sample difference.

Usage

```
scr_cutoff(x, n_cuts = NULL, cuts = NULL)
```

Arguments

x	An object from scr_scorecard().
n_cuts	Number of candidate cuts. NULL uses config$cutoff_n.
cuts	Explicit vector of cuts; overrides n_cuts.

Details

The "safe side" is the high-score side under higher_is_safer (credit) and the low-score side under higher_is_riskier (fraud, propensity).

Value

An scr_cutoff object with table (one row per sample and cut) and direction.

See Also

Other stages: scr_align(), scr_bin(), scr_model(), scr_reject(), scr_scorecard(), scr_select(), scr_split(), scr_strategy(), scr_triage()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
ct <- scr_cutoff(sc, n_cuts = 10)
ct
st <- scr_strategy(sc, revenue_good = 1080, loss_bad = 4500)
st
rj <- scr_reject(sc)
rj
```

---

scr_decisions *Decision ledger of a lab, a result or a scorecard*

Description

Returns the append-only ledger of manual decisions: every proposal accepted, discarded or superseded, every forced or dropped variable and every override, each with its reason.

Usage

```
scr_decisions(x)
```

Arguments

x	An scr_classing lab, an scr_result from scr_classing_apply() or an scr_scorecard fitted on one.

Value

A data.table, one row per decision (append-only), or an empty one when no manual decision exists.

See Also

scr_coarse_classing() for a complete session, from lab to scorecard.

Other classing: scr_classing_accept(), scr_classing_apply(), scr_classing_choose(), scr_classing_propose(), scr_classing_spec(), scr_classing_view(), scr_coarse_classing()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
p <- scr_classing_propose(lab, "ds_region",
                          groups = list(edge = c("NORTH", "SOUTH"),
                                        core = c("EAST", "WEST", "CENTRE")))
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
scr_decisions(lab)
scr_decisions(scr_classing_apply(lab))
scr_decisions(res)   # no manual decision: an empty ledger
```

---

scr_default *Build the default flag from a monthly panel*

Description

Applies the standard definition of default to a panel with one row per unit (id) and month (date): a unit enters default when dpd reaches default_days and the arrears are material (both default_abs in currency units and default_rel as a share of exposure, when arrears and exposure are supplied), or when utp (unlikeliness to pay) is TRUE. It leaves default after default_probation consecutive months without a trigger (default_probation_restructured when restructured was TRUE at any point of the event). With obligor supplied and default_level = "obligor", a unit whose obligor has more than default_pulling of its exposure in default is pulled into default too, and a defaulted obligor defaults all its units.

Usage

```
scr_default(
  data,
  id,
  date,
  dpd = NULL,
  arrears = NULL,
  exposure = NULL,
  utp = NULL,
  restructured = NULL,
  obligor = NULL,
  config = scr_config()
)
```

Arguments

data	A data.frame or data.table, one row per id and date.
id, date	Column names of the unit identifier and the month.
dpd	Column name of days past due (integer). Optional when utp is given.
arrears, exposure	Column names of the overdue amount and the total exposure, both optional; when given, the materiality test applies.
utp	Column name of a logical unlikeliness-to-pay flag, optional.
restructured	Column name of a logical distressed-restructuring flag, optional.
obligor	Column name of the obligor when id is a facility, optional; enables the pulling effect.
config	A scr_config(); keys default_*.

Details

Rows must be monthly; gaps are tolerated (the probation counts observed months). The result keeps the row-level flags: they are the product.

Value

An object of class scr_default: flags (id, date, default 0/1, event_id, trigger, months_in_default, cured), events (one row per event: event_id, id, start, end, trigger, cured, months), summary, ledger and config.

See Also

Other irb-parameters: scr_default_rate(), scr_irb_params()

Examples

```
d <- scr_default(scr_demo_panel, id = "id", date = "ref_date", dpd = "dpd",
                 arrears = "arrears", exposure = "exposure",
                 restructured = "restructured",
                 config = scr_config(verbose = FALSE))
d
head(d$events)
```

`scr_default_rate` *One-year default rates by cohort and the long-run average*

Description

From a flagged panel (an `scr_default()` object or any table with a 0/1 default column by unit and month), computes for every cohort start the population of non-defaulted units, the share that defaults within `horizon` months, optionally by `grade` or `segment` (as observed at the cohort start) and exposure-weighted when `exposure` is given. The long-run average is the arithmetic mean of the cohort rates. When the analyst proposes an adjusted value (`lra_adjusted`, for instance after judging that the period lacks bad years), it is benchmarked against the larger of the last five years' mean and the whole period's mean, and a flag records when it sits below that benchmark.

Usage

```
scr_default_rate(
  x,
  id = "id",
  date = "date",
  default = "default",
  horizon = 12L,
  by = NULL,
  grade = NULL,
  segment = NULL,
  exposure = NULL,
  lra_adjusted = NULL,
  config = scr_config()
)
```

Arguments

`x`	An `scr_default` or a `data.frame`/`data.table`.
`id, date, default`	Column names (ignored for an `scr_default`).
`horizon`	Months of the default window after the cohort start.
`by`	Cohort frequency: `"month"`, `"quarter"` or `"year"`.
`grade, segment`	Optional column names observed at the cohort start.
`exposure`	Optional column name; adds exposure-weighted rates.

lra_adjusted	Optional adjusted long-run average proposed by the analyst, in [0, 1]; benchmarked and flagged, never applied.
config	A scr_config(); only pd_dr_by is read (the default of by).

Value

An object of class scr_dr: table (cohort rates, by grade or segment when given), portfolio (one row per cohort: n, defaults, dr), lra (mean, weighted_mean, recent5_mean, benchmark, adjusted, flag_below_benchmark, min, max, sd, n_cohorts, years), horizon and by.

See Also

Other irb-parameters: scr_default(), scr_irb_params()

Examples

```
d <- scr_default(scr_demo_panel, id = "id", date = "ref_date", dpd = "dpd",
                 config = scr_config(verbose = FALSE))
dr <- scr_default_rate(d, by = "quarter")
dr
dr$table
```

---

scr_demo *Synthetic example data*

Description

A fabricated table so that every example and every vignette runs without a database. On purpose, it carries the defects real data has: sentinel -999 in several masses (vl_hist_*), missing values (vl_partial_*), a column that only degrades in the last period (vl_late), constant, near-constant, exact duplicate, high cardinality, a redundant pair and pure noise. Without them, the audit funnel would have nothing to show.

Usage

```
scr_demo
```

Format

A data.frame with 4,200 rows and 41 columns:

id Identifier (never a candidate; goes in drop).

ref_date Monthly reference date, six periods; the key of the out-of-time split.

vl_score_01 **to** vl_score_12 Numerics with decreasing signal.

vl_hist_01 **to** vl_hist_05 Numerics with an increasing mass of sentinel -999; the signal is in the absence.

vl_partial_01 **to** vl_partial_03 Numerics with genuine NA.

`vl_late` Numeric with NA/sentinel in the last period only.

`vl_noise_01` **to** `vl_noise_06` Pure noise.

`vl_constant`, `vl_near_const`, `ds_constant`, `vl_duplicate`, `vl_redundant`, `ds_high_card` Structural pathologies.

`ds_region`, `ds_band`, `ds_channel`, `ds_optin` Categoricals with signal; `ds_optin` has NA.

`default` Risk target (0/1, about 14% events).

`churn` Propensity target (0/1, about 29% events), for the portfolio examples.

Source

Synthetic. Generated by `data-raw/scr_demo.R`, seed `20260903`, in the package source repository https://github.com/evandeilton/scorecraft.

See Also

Other data: `scr_demo_ead`, `scr_demo_lgd`, `scr_demo_lgd_cashflows`, `scr_demo_panel`, `scr_demo_portfolio`, `scr_demo_rates`

Examples

```
str(scr_demo[, 1:6])
mean(scr_demo$default)
```

---

scr_demo_ead *Synthetic monthly facility snapshots for the EAD/CCF module*

Description

1,200 revolving facilities (cards, overdrafts and revolving lines) of 950 obligors, observed monthly (snapshots dated the first day of the month) over 30 months. Utilization follows a facility-specific level with a mild drift; about 10% of the facilities default, with the utilization ramping up over the twelve months before the default month at an intensity that depends on the drivers a CCF model is expected to find (utilization, product, months on book, days past due). Some defaulters are fully drawn or over the limit at the reference date, some repay before default (negative realized CCF), some have their limit cut before default, a share of all facilities gets a limit change and a few facilities originate inside the window (fast defaults). Built for `scr_ead_data()`, `scr_ead()` and `scr_ead_validate()`.

Usage

```
scr_demo_ead
```

Format

A `data.frame` with 34,811 rows and 9 columns:

`facility_id` Facility identifier.

`obligor_id` Obligor identifier (a few obligors hold two facilities).

`ref_date` Month (first day), 30 periods from 2023-01.

`limit` Limit of the facility at the month end.

`drawn` Drawn amount at the month end, gross; may exceed the limit.

`product` `"card"`, `"overdraft"` or `"line"`.

`months_on_book` Age of the facility in months.

`dpd` Days past due at the month end (multiples of 30).

`defaulted` 0/1 flag, 1 from the default month onward.

Source

Synthetic. Generated by `data-raw/scr_demo_ead.R`, seed `20260904`, in the package source repository https://github.com/evandeilton/scorecraft.

See Also

Other data: `scr_demo`, `scr_demo_lgd`, `scr_demo_lgd_cashflows`, `scr_demo_panel`, `scr_demo_portfolio`, `scr_demo_rates`

Examples

```
head(scr_demo_ead)
length(unique(scr_demo_ead$facility_id[scr_demo_ead$defaulted == 1]))
```

---

scr_demo_lgd *Synthetic default events for the workout LGD examples*

Description

900 default events on three products (unsecured, mortgage, auto) observed until 2026-06-30, with the drivers a workout LGD model reads: collateral, loan-to-value, time on book, the worst delinquency before default and the region. Cures return to performing within six months; non-cures recover according to product-specific profiles until a close date; events still running at the observation date are open, a few of them older than the maximum recovery period. Thirty facilities default twice after a cure, half of them within nine months, so that `scr_workout()` merges the two spells into one event. Built for `scr_workout()`, `scr_lgd()` and the in-default examples.

Usage

```
scr_demo_lgd
```

Format

A `data.frame` with 900 rows and 12 columns:

`default_id` Default event identifier.

`facility_id` Facility identifier; repeated for the second defaults.

`default_date` Date of default.

`ead` Exposure at default.

`product` `"unsecured"`, `"mortgage"` or `"auto"`.

`collateral_value` Collateral value at default; `0` when unsecured.

`ltv` Loan-to-value at default; `0` when unsecured.

`months_on_book` Months since origination at default.

`prior_dpd_max` Worst days past due in the year before default (multiples of 30).

`region` Region of the facility.

`status` `"closed"`, `"cured"` or `"open"` at the observation date.

`close_date` Date the workout closed or the cure was confirmed; `NA` when open.

Source

Synthetic. Generated by `data-raw/scr_demo_lgd.R`, seed `20260904`, in the package source repository https://github.com/evandeilton/scorecraft.

See Also

Other data: scr_demo, scr_demo_ead, scr_demo_lgd_cashflows, scr_demo_panel, scr_demo_portfolio, scr_demo_rates

Examples

```
table(scr_demo_lgd$product, scr_demo_lgd$status)
```

---

scr_demo_lgd_cashflows

Synthetic post-default cash flows of `scr_demo_lgd`

Description

Long table of the cash flows of every default event of scr_demo_lgd: recoveries spread over up to four years for unsecured facilities, a collateral sale between one and three years in for mortgages, a repossession sale within nine months for cars, direct costs at the start of the workout and at the sale, and a few drawings after default. Cures carry only the payments that brought the facility back to performing.

Usage

```
scr_demo_lgd_cashflows
```

Format

A `data.frame` with 5,737 rows and 4 columns:

`default_id` Default event identifier, matching scr_demo_lgd.

`date` Date of the cash flow.

`amount` Amount, always positive.

`type` `"recovery"`, `"direct_cost"` or `"drawing"`.

Source

Synthetic. Generated by `data-raw/scr_demo_lgd.R`, seed `20260904`, in the package source repository https://github.com/evandeilton/scorecraft.

See Also

Other data: scr_demo, scr_demo_ead, scr_demo_lgd, scr_demo_panel, scr_demo_portfolio, scr_demo_rates

Examples

```
head(scr_demo_lgd_cashflows)
table(scr_demo_lgd_cashflows$type)
```

---

scr_demo_panel *Synthetic monthly panel for the default engine and PD calibration*

Description

600 obligors observed over 36 months. Days past due evolve as a chain whose slip probability depends on a latent risk; arrears are proportional to the exposure, with a few obligors whose arrears stay below the absolute materiality threshold on purpose; 25 obligors are restructured from month 18; `score` is a behavioral score (higher = safer) with genuine rank-ordering power. Built for `scr_default()`, `scr_default_rate()` and the PD grade examples.

Usage

```
scr_demo_panel
```

Format

A `data.frame` with 21,600 rows and 7 columns:

`id` Obligor identifier.

`ref_date` Month (first day), 36 periods from 2023-01.

`dpd` Days past due at the snapshot (multiples of 30).

`arrears` Overdue amount.

`exposure` Total exposure of the obligor (constant).

`restructured` Logical distressed-restructuring flag.

`score` Behavioral score, higher is safer.

Source

Synthetic. Generated by `data-raw/scr_demo_panel.R`, seed `20260904`, in the package source repository https://github.com/evandeilton/scorecraft.

See Also

Other data: `scr_demo`, `scr_demo_ead`, `scr_demo_lgd`, `scr_demo_lgd_cashflows`, `scr_demo_portfolio`, `scr_demo_rates`

Examples

```
head(scr_demo_panel)
mean(scr_demo_panel$dpd >= 90)
```

---

scr_demo_portfolio *Synthetic exposure snapshot for expected loss, capital and ECL*

Description

5,000 exposures in six segments that map one-to-one to the asset classes of the IRB risk-weight function. The PD is a grade PD (constant within asset class and grade; the ten geometric grades start below the regulatory floors on purpose) and the LGD a pool value constant within the segment, so that segment by grade is a homogeneous pool and the production SQL of `scr_capital()` reproduces R exactly. About 3 % of the exposures are in default with a best estimate of expected loss and a provision close to it; the other columns feed the standardized comparison and the accounting stage rule of `scr_ecl()`.

Usage

```
scr_demo_portfolio
```

Format

A `data.frame` with 5,000 rows and 21 columns:

`id` Exposure identifier.

`segment` Reporting segment (`retail_loans`, `mortgages`, `cards_revolver`, `cards_transactor`, `corporate_large`, `corporate_sme`).

`asset_class` Asset class of the risk-weight function.

`grade` PD grade, `G01` (safest) to `G10`.

`pd` One-year PD of the grade (decimal).

`lgd` Downturn LGD of the pool (decimal).

`ead`, `drawn`, `undrawn` Exposure at default and its split.

`m` Effective maturity in years (corporates only).

`sales` Annual sales in millions (corporates only).

`ltv` Loan-to-value at origination (mortgages only).

`rating` External rating (large corporates; `NA` unrated).

`transactor` Logical: revolving facility repaid in full monthly.

`defaulted` 0/1 default flag (about 3 %).

`elbe` Best estimate of expected loss of the defaulted rows.

`provision` Provision stock.

`stage` Accounting stage 1, 2 or 3.

`dpd` Days past due.

`pd_orig` One-year PD at origination.

`eir` Annual effective interest rate.

Source

Synthetic. Generated by `data-raw/scr_demo_portfolio.R`, seed `20260904`, in the package source repository https://github.com/evandeilton/scorecraft.

See Also

Other data: `scr_demo`, `scr_demo_ead`, `scr_demo_lgd`, `scr_demo_lgd_cashflows`, `scr_demo_panel`, `scr_demo_rates`

Examples

```
head(scr_demo_portfolio)
table(scr_demo_portfolio$asset_class)
```

---

scr_demo_rates *Synthetic monthly reference rate series*

Description

A smooth annualized reference rate by month from 2019-01 to 2026-06, falling to two percent in 2020-2021 and rising above thirteen percent in 2022-2023, used by `scr_workout()` as the rate at the default date.

Usage

```
scr_demo_rates
```

Format

A `data.frame` with 90 rows and 2 columns:

`date` First day of the month.

`rate` Annual reference rate as a decimal.

Source

Synthetic. Generated by `data-raw/scr_demo_lgd.R`, seed `20260904`, in the package source repository https://github.com/evandeilton/scorecraft.

See Also

Other data: scr_demo, scr_demo_ead, scr_demo_lgd, scr_demo_lgd_cashflows, scr_demo_panel, scr_demo_portfolio

Examples

```
range(scr_demo_rates$rate)
```

---

scr_detection *Time to detection of fraud episodes*

Description

Reads a score on transactions grouped by entity (a card, an account, a device) and measures, for each alert threshold, how many fraud episodes the score detects, how many fraudulent transactions pass before the first alert, how long that takes and how much of the loss is prevented.

Usage

```
scr_detection(x, ...)

## S3 method for class 'data.frame'
scr_detection(
  x,
  entity,
  time,
  score = "score",
  y = "y",
  amount = NULL,
  thresholds = NULL,
  alert_shares = NULL,
  direction = "higher_is_riskier",
  max_cells = 1e+05,
  ...
)
```

Arguments

x	A data.frame with one row per transaction.
...	Not used; an unknown argument is an error.
entity	Name of the column that identifies the entity.
time	Name of the time column: a number, a Date or a POSIXct.
score, y	Column names of the score and of the 0/1 outcome (1 = a fraudulent row).
amount	Optional column of the amount of each row, for the loss.
thresholds	Score thresholds of the alerts.
alert_shares	Shares of all rows to alert, in (0, 1], turned into thresholds; may be given with or instead of thresholds.
direction	"higher_is_riskier" (default) or "higher_is_safer".
max_cells	Largest number of distinct score values kept exactly in the count table of the alert shares.

Value

An object of class c("scr_detection", "list"):

table One row per threshold: threshold, target_share, alert_share, episodes, detected, pct_detected, median_events_before, mean_events_before, median_time, loss_before, loss_total and pct_loss_prevented.

episodes One row per episode: entity, start, rows (from the start), events, amount (of its event rows), peak_score (the highest score from the start; the lowest under higher_is_safer) and, for the strictest threshold, detected, detect_time, time_to_detect, events_before and loss_before (NA when the episode is not detected).

direction, entity, time, score, target, amount, n_rows, n_dropped, n_episode_rows, call The settings and the row counts; n_episode_rows is the number of rows of the episodes, from their starts.

Episodes

An episode is an entity with at least one event row. It starts at its first event row in time order (rows at the same time keep their order in x), and the rows of the entity before that start are ignored. At a threshold t, the episode is detected at the first row from its start whose score is on the alert side: score >= t under higher_is_riskier, score < t under higher_is_safer. The detecting row may be any row of the entity, an event or not.

Table

One row per threshold, the strictest (fewest alerts) first:

- alert_share: the share of **all** rows of x on the alert side of the threshold, the workload it costs;
- episodes, detected and pct_detected;
- median_events_before and mean_events_before: the event rows of the episode before the detecting row, over the detected episodes (0 when the first event row is alerted);

- median_time: the median time from the start of the episode to the detecting row, in the unit of time (days for a Date, seconds for a POSIXct);
- with amount: loss_before, the amount of the event rows before the detecting row, plus the whole amount of the episodes never detected; loss_total, the amount of every event row; and pct_loss_prevented = 1 - loss_before / loss_total. The detecting row and the rows after it count as prevented.

alert_shares are turned into thresholds on all rows, tie-safe as in scr_bands(): the boundary between two distinct scores nearest to the share (target_share keeps the share asked for, alert_share the one realized).

Rows with a missing entity or time, or a missing or infinite score, are left out (n_dropped); a missing outcome is not an event, and a missing amount counts as 0.

See Also

scr_overlap() for the rules against the score, scr_operating() for the threshold under a review capacity.

Other score-studies: scr_bands(), scr_claims(), scr_maturity(), scr_mix_shift(), scr_operating(), scr_overlap(), scr_rag(), scr_rag_plan(), scr_score_cross(), scr_segments(), scr_tiers(), scr_uplift()

Examples

```
local({
  set.seed(1)
  n <- 30000
  d <- data.frame(card = sample(3000, n, TRUE), ts = sort(runif(n, 0, 30)))
  # 60 cards are compromised from some day on; their later rows are fraud
  hit <- sample(3000, 60)
  since <- runif(3000, 5, 25)
  d$y <- as.integer(d$card %in% hit & d$ts >= since[d$card])
  d$score <- round(100 * plogis(rnorm(n, -2 + 2 * d$y)))
  d$amount <- round(rexp(n, 1 / 60), 2)
  dt <- scr_detection(d, entity = "card", time = "ts", amount = "amount",
                      alert_shares = c(0.01, 0.03, 0.10))
  print(dt)
  head(dt$episodes)
})
```

---

scr_ead *Estimate CCF pools from the reference data set*

Description

Splits the reference data set by reference date (the most recent cohorts form the hold-out), bins every driver against the realized CCF with the continuous binner (scr_bin_continuous()) on the training rows, revalidates the frozen bins on the hold-out and admits a driver when it passes the named rules TOO_FEW_DEFAULTS, NO_SEPARATION, NOT_MONOTONIC and UNSTABLE_HOLDOUT. The

cells of the cross of the admitted drivers are ordered by their predicted CCF and merged, adjacent cells first, down to `config$ccf_n_pools` pools with at least `config$ccf_min_defaults` defaults each. Rows in the limit-factor measure form their own pool `LF`.

Usage

```
scr_ead(x, drivers, config = scr_config(), holdout = 0.3, params = NULL)
```

Arguments

`x`	An `scr_ead_data()` object.
`drivers`	Column names of the candidate drivers (columns of `x$rds`).
`config`	A `scr_config()`.
`holdout`	Hold-out share, by whole reference dates.
`params`	An `scr_irb_params()` object; `NULL` uses the preset of `config$framework`.

Details

Per pool the estimate is the long-run (default-weighted) average of the realized values on the training rows, `lra`; `moc_est` is the one-sided normal estimation-error margin at `config$ccf_moc_alpha`; `ccf_dt` is the downturn value (equal to `lra` until `scr_ead_downturn()` is run); `ccf_final = max(lra, ccf_dt) + moc_est`; `ccf_floor = params$ccf_floor_fraction * config$ccf_sa_ccf`; and `ccf_applied = max(ccf_final, ccf_floor)`. For the `LF` pool the floor depends on the utilization and is applied per row by `scr_apply()`.

Value

An object of class `scr_ead`: `pools` (the pool table), `cells` (every cell of the cross with its pool), `bins` (the obwoe-shaped fit of the admitted drivers), `bins_all` (the fit of every driver), `drivers` (admission table), `holdout` (frozen bins on the hold-out), `rds` (the reference rows with `sample` and `pool`), `metrics` (per sample: `rmse`, `mae`, `gauc` with a bootstrap interval, `spearman`, `ead_rmse`, `ead_mae`, `adequacy`, `cear`), `split`, `funnel`, `data_summary`, `downturn`, `ledger`, `model_card`, `params`, `config`, `meta`. Also `survivors` (the admitted drivers) and `lra` (the long-run averages of the data set); `metrics` also carries `n`, `n_main`, `gauc_se`, `somers_d` and `share_floor_binding`.

See Also

Other irb-ead: `scr_bin_continuous()`, `scr_ead_data()`, `scr_ead_downturn()`, `scr_ead_validate()`

Examples

```
cfg <- scr_config(verbose = FALSE, n_boot = 20, nthread = 1)
ed <- scr_ead_data(scr_demo_ead, facility_id = "facility_id", date_col = "ref_date",
                   limit = "limit", drawn = "drawn", defaulted = "defaulted",
                   drivers = c("product", "months_on_book", "dpd"), config = cfg)
m <- scr_ead(ed, drivers = c("utilisation_ref", "product", "months_on_book"), config = cfg)
m
m$pools
m$drivers
```

scr_ead_data *Build the realized-CCF reference data set from facility snapshots*

Description

One row per default event (or per event and reference date under the variable-horizon comparison), with the facility as it stood at the reference date and the realized exposure at default (EAD) at the default date, from which the realized credit conversion factor (CCF) follows. The reference date follows `config$ccf_horizon`: `"fixed"` takes the snapshot `ccf_horizon_months` before the default month (the nearest earlier snapshot when that month is missing; the first snapshot for a facility younger than the horizon, flagged `FAST_DEFAULT`); `"cohort"` takes the start of the calendar cohort window in which the default falls; `"variable"` takes every snapshot in the horizon before the default, for comparison only.

Usage

```
scr_ead_data(
  snapshots,
  facility_id,
  obligor_id = NULL,
  date_col,
  limit,
  drawn,
  default_date = NULL,
  defaulted = NULL,
  drivers = NULL,
  config = scr_config(),
  keep_rows = FALSE
)
```

Arguments

`snapshots`	A `data.frame` or `data.table` with one row per facility and month.
`facility_id`, `date_col`, `limit`, `drawn`	Column names of the facility identifier, the snapshot month, the limit and the drawn amount.
`obligor_id`	Column name of the obligor, optional. With `config$default_level = "obligor"` a default of any facility of the obligor is a default of all its facilities observed at that date.
`default_date`	Either the name of a column of `snapshots` holding the default date of the facility (`NA` when it never defaults) or a `data.frame` with the facility identifier column (same name as `facility_id`) and a `default_date` column, one row per event.
`defaulted`	Column name of a 0/1 default flag per snapshot, alternative to `default_date`: every run of ones opens an event at its first month.

drivers	Column names measured at the reference date and carried into the data set (candidate drivers of the pools). utilisation_ref, limit_ref, drawn_ref and horizon_months are always available.
config	A scr_config(); keys ccf_*, post_default_drawings_in, default_level.
keep_rows	If TRUE, keeps every candidate event with its exclusion rule in rows.

Details

The realized measure per row follows config$ccf_measure: under "auto" the undrawn-limit factor (CCF) when the utilization at the reference date is below ccf_u_star and the limit factor (LF) at or above it; rows with nothing undrawn or over the limit at the reference date are always routed to the limit factor (ZERO_UNDRAWN, OVER_LIMIT_AT_REF), never dropped. The raw realized value is kept in ccf_raw; ccf carries the value after the optional floor (ccf_floor_realised) and cap (ccf_cap_realised), both logged in the funnel (NEGATIVE_CCF_FLOORED, CCF_ABOVE_ONE). The realized EAD is the drawn amount at the default date, uncapped; with post_default_drawings_in = "ccf" it is the maximum drawn amount over the default event when defaulted is given.

Value

An object of class scr_ead_data: rds (one row per event: event_id, facility_id, obligor_id, ref_date, default_date, cohort, horizon_months, fast_default, limit_ref, drawn_ref, undrawn_ref, utilisation_ref, limit_default, limit_change, ead_realised, measure, ccf_raw, ccf, rule, drivers), funnel (rule, action, n, share), summary (simple and exposure-weighted averages by cohort and measure, with a total row), lra (long-run averages and shares), meta, ledger, config and rows (with keep_rows = TRUE).

References

Basel Committee on Banking Supervision (2023). *The Basel Framework*, CRE32 and CRE36. Moral, G. (2006). EAD estimates for facilities with explicit limits. In Engelmann, B. and Rauhmeier, R. (eds), *The Basel II Risk Parameters*. Springer.

See Also

Other irb-ead: scr_bin_continuous(), scr_ead(), scr_ead_downturn(), scr_ead_validate()

Examples

```
cfg <- scr_config(verbose = FALSE)
ed <- scr_ead_data(scr_demo_ead, facility_id = "facility_id", obligor_id = "obligor_id",
                   date_col = "ref_date", limit = "limit", drawn = "drawn",
                   defaulted = "defaulted",
                   drivers = c("product", "months_on_book", "dpd"), config = cfg)
ed
ed$funnel
head(ed$rds[, c("facility_id", "ref_date", "default_date", "utilisation_ref", "measure", "ccf")])
```

scr_ead_downturn *Downturn CCF per pool*

Description

Quantifies the downturn component of the CCF from user-supplied downturn periods. "type1" (observed impact) takes, per pool, the default-weighted average of the realized values of the training events whose default date falls in the periods (the hold-out stays independent) and sets ccf_dt = max(lra, observed); "type3" (long-run average plus add-on) sets ccf_dt = lra + add_on; "none" resets ccf_dt = lra. The pool table is recomputed (ccf_final, ccf_applied) and the ledger records the periods, the method and the reason.

Usage

```
scr_ead_downturn(
  x,
  periods = NULL,
  method = NULL,
  add_on = 0.15,
  reason = NULL
)
```

Arguments

x	An scr_ead() object.
periods	A data.frame with start and end dates of the downturn periods (needed for "type1").
method	"type1", "type3" or "none"; NULL uses config$ccf_downturn.
add_on	Add-on of the "type3" method, in CCF units.
reason	Text justifying the periods and the method; mandatory.

Value

The scr_ead object with downturn (a list with method, periods, add_on and the per-pool table: pool, lra, n_downturn, dt_observed, dt_type3, ccf_dt), the updated pools and a new ledger row. The table also carries ccf_final and ccf_applied; the object n_rows_in_periods and reason.

See Also

Other irb-ead: scr_bin_continuous(), scr_ead(), scr_ead_data(), scr_ead_validate()

Examples

```
cfg <- scr_config(verbose = FALSE, n_boot = 20, nthread = 1)
ed <- scr_ead_data(scr_demo_ead, facility_id = "facility_id", date_col = "ref_date",
                   limit = "limit", drawn = "drawn", defaulted = "defaulted",
                   drivers = c("product", "months_on_book"), config = cfg)
m <- scr_ead(ed, drivers = c("utilisation_ref", "product"), config = cfg)
m2 <- scr_ead_downturn(m, periods = data.frame(start = as.Date("2024-01-01"),
                                               end = as.Date("2024-12-01")),
                       reason = "2024 chosen as the stress year of the demo panel")
m2$downturn$table
```

`scr_ead_validate` *Validate CCF pools: calibration, discrimination, back-testing and stability*

Description

Per pool and in total, compares realized and predicted values on the validation rows (the hold-out of the model by default): simple and exposure-weighted averages, the one-sided t-test of realized above predicted (under-estimation) with its p-value, the EAD adequacy ratio (sum of realized EAD over sum of predicted EAD) and traffic lights (red at or below `lights[1]`, amber at or below `lights[2]`, green above; adequacy green at or below `adequacy_lights[1]`, amber up to `adequacy_lights[2]`, red above; grey when the value is missing). Adds the discrimination block (gAUC with a bootstrap interval against the development value, Spearman correlation, cumulative EAD accuracy ratio), the back-test by cohort and the stability of the pool distribution and of the driver bins (`scr_psi()`, fixed and sample-size-adjusted thresholds). The numeric limits of the lights are a convention of the package, stated as such in the output.

Usage

```
scr_ead_validate(
  x,
  newdata = NULL,
  lights = c(0.01, 0.05),
  adequacy_lights = c(1, 1.05)
)
```

Arguments

`x`	An `scr_ead()` object.
`newdata`	`NULL` (the hold-out rows of `x`), an `scr_ead_data()` object or its `rds` table.
`lights`	Two increasing p-value thresholds: red at or below the first, amber at or below the second.
`adequacy_lights`	Two increasing adequacy-ratio thresholds.

Value

An object of class `scr_ead_validation`: `calibration`, `discrimination`, `backtest`, `stability`, `summary` (test, statistic, p, light; the light is `"grey"` when the test has no result), `light` (the worst light of the summary: red, then amber, then green; `"grey"` when no test has a result), `n`, `source`.

See Also

Other irb-ead: `scr_bin_continuous()`, `scr_ead()`, `scr_ead_data()`, `scr_ead_downturn()`

Examples

```
cfg <- scr_config(verbose = FALSE, n_boot = 20, nthread = 1)
ed <- scr_ead_data(scr_demo_ead, facility_id = "facility_id", date_col = "ref_date",
                   limit = "limit", drawn = "drawn", defaulted = "defaulted",
                   drivers = c("product", "months_on_book"), config = cfg)
m <- scr_ead(ed, drivers = c("utilisation_ref", "product"), config = cfg)
v <- scr_ead_validate(m)
v
v$calibration
```

---

scr_ecl *Expected credit loss with stage allocation*

Description

Discrete-time expected credit loss of every exposure:

Usage

```
scr_ecl(
  pd_term,
  lgd,
  ead,
  eir = 0,
  stage = NULL,
  dpd = NULL,
  pd_orig = NULL,
  scenarios = NULL,
  weights = NULL,
  prepay = NULL,
  rho = 0.15,
  t_max = NULL,
  segment = NULL,
  id = NULL,
  config = scr_config(),
  keep_rows = FALSE
)
```

Arguments

pd_term	Marginal monthly PDs: a matrix n x T, or a vector of length n (a flat hazard recycled over t_max months), or a single number.
lgd, ead	Vectors of length n (or one) or matrices n x T.
eir	Annual effective interest rate, vector of length n or one.
stage	Optional stage vector (1, 2, 3); NULL applies the rule.
dpd	Days past due (the rule); optional.
pd_orig	12-month PD at origination (the rule); optional.
scenarios	Named list of scenario shocks (see Details); NULL runs the base case only.
weights	Scenario weights; equal when NULL.
prepay	Monthly prepayment hazard: NULL, a vector or an n x T matrix.
rho	Asset correlation used by scenario shocks with z.
t_max	Term in months when pd_term is a vector (default config$ecl_horizon_months).
segment	Optional character vector of length n for a segment summary.
id	Optional identifier vector of length n.
config	An scr_config() object (ecl_* keys, verbose).
keep_rows	Keep the per-exposure table.

Details

$$ECL_H = \sum_{t=1}^{H} S(t-1)\, h_t\, LGD_t\, EAD_t\, (1+r)^{-t/12}, \qquad S(t) = \prod_{s \le t} (1 - h_s - p_s)$$

with h_t the marginal monthly default hazard, p_t an optional prepayment hazard and r the annual effective interest rate (config$ecl_discount = "none" switches the discounting off). The 12-month figure uses H = config$ecl_horizon_months, the lifetime figure the full term T. Stage 1 exposures carry the 12-month loss, stages 2 and 3 the lifetime loss; stage 3 exposures are credit-impaired and carry LGD_1 * EAD_1. When stage is NULL the rule is: stage 3 if dpd >= config$ecl_stage_dpd[2], stage 2 if dpd >= config$ecl_stage_dpd[1] or the 12-month PD now over the one at origination (pd_orig) is at least config$ecl_sicr_ratio, else stage 1.

Scenarios are a named list of shocks applied to the base inputs, each a list with any of pd_mult (non-negative multiplier of the hazards, capped at one), z (systematic factor of the one-factor model applied to the hazards with correlation rho, negative in a bad year), lgd_add (added to the LGD, the result floored at zero) and ead_mult (non-negative); weights (normalized to one) give the probability-weighted result. When a shocked hazard plus the prepayment hazard exceeds one, the exit probability of that month is capped at one. The z shock is applied to each monthly hazard, not to the annual PD; because the Vasicek map is non-linear, the implied 12-month stressed PD is higher than the one obtained by stressing the annual PD with the same z and rho (convert the annual PD yourself when that is wanted).

Value

An object of class `scr_ecl`: a list with `exposures` (only with `keep_rows = TRUE`: `id`, `segment`, `stage`, `ead`, `pd_12m`, `pd_life`, `ecl_12m`, `ecl_life`, `ecl`), `stages` (`stage`, `n`, `ead`, `ecl_12m`, `ecl_life`, `ecl`, `coverage`), `segments` (when a segment is given), `scenarios` (`scenario`, `weight`, `ecl_12m`, `ecl_life`, `ecl`), `totals` (`n`, `ead`, `ecl_12m`, `ecl_life`, `ecl`, `coverage`, `share_stage2`, `share_stage3`), `horizon`, `t_max`, `discount`, `stage_rule`, `ledger` and `config`.

References

International Accounting Standards Board (2014). *IFRS 9 Financial Instruments*, section 5.5 and paragraphs B5.5.1-B5.5.55.

See Also

Other irb-capital: `scr_capital()`, `scr_el()`, `scr_irb_rw()`, `scr_pd_stress()`, `scr_sa_rw()`

Examples

```
cfg <- scr_config(verbose = FALSE)
d <- scr_demo_portfolio
h <- 1 - (1 - d$pd)^(1 / 12)       # flat monthly hazard from the annual PD
e <- scr_ecl(h, d$lgd, d$ead, eir = d$eir, dpd = d$dpd, pd_orig = d$pd_orig,
             t_max = 36L, segment = d$segment, config = cfg)
e
e$stages
```

---

scr_el *Expected loss per exposure*

Description

The primitive every other function of the module uses: `pd * lgd * ead` for performing exposures and `elbe * ead` for defaulted ones, `elbe` being the best estimate of expected loss. A defaulted exposure without `elbe` uses `lgd` (PD equal to one). Arguments are recycled to a common length.

Usage

```
scr_el(pd, lgd, ead, defaulted = NULL, elbe = NULL)
```

Arguments

`pd`, `lgd`, `ead`	Numeric vectors: probability of default, loss given default (decimals) and exposure at default (currency).
`defaulted`	Optional 0/1 or logical vector.
`elbe`	Optional vector with the best estimate of expected loss of the defaulted rows (decimal of `ead`); ignored on performing rows.

Value

A numeric vector with the expected loss in currency.

See Also

Other irb-capital: scr_capital(), scr_ecl(), scr_irb_rw(), scr_pd_stress(), scr_sa_rw()

Examples

```
scr_el(c(0.01, 0.02), 0.45, c(1000, 2000))
scr_el(0.02, 0.45, 1000, defaulted = TRUE, elbe = 0.6)
```

---

scr_elbe *ELBE and in-default LGD on a grid of months since default*

Description

For every pool and every reference age tau of the grid, the expected loss best estimate is the mean realized LGD of the training defaults of the pool that were still in workout at tau (so that at tau = 0 it equals the pool's long-run average), and the in-default LGD adds the unexpected-loss increment

$$\Delta^{UL}(\tau) = \max(0,\ \mathrm{LGD}^{DT} - \mathrm{LRA})\ \frac{\rho(T_{\max}) - \rho(\tau - 1)}{\rho(T_{\max})}$$

read from the recovery profile of the pool's product mix, where $\rho(\tau-1)$ is the cumulative discounted recovery rate of the months before age tau (zero at tau = 0): the downturn uplift shrinks as the recoveries come in. The consistency table checks that lgd_in_default at tau = 0 reproduces the pool's lgd_dt.

Usage

```
scr_elbe(x, grid = NULL)
```

Arguments

x	An scr_lgd() object.
grid	Months since default; NULL uses lgd_elbe_grid.

Value

An object of class scr_elbe: table (months_since_default, pool, n_open, share_open, recovered_share, elbe, delta_ul, lgd_in_default), consistency (per pool at tau = 0), grid, t_max.

See Also

Other irb-lgd: scr_lgd(), scr_lgd_downturn(), scr_lgd_floor(), scr_lgd_pools(), scr_lgd_validate(), scr_workout()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
e <- scr_elbe(m)
e
```

---

scr_export *Write the deliverables*

Description

For an `scr_result`: the selection workbook (`selection_<target>.xlsx`: funnel, gains, screening, hold-out, models, votes, consensus, ledger, redundancy), the WOE SQL and the executive summary in Markdown. For an `scr_scorecard`: three workbooks, as separate files, plus the score SQL:

Usage

```
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_capital'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_claims'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_classing'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_detection'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_ead'
scr_export(x, dir, stamp = TRUE, validation = NULL, tag = "ccf", ...)

## S3 method for class 'scr_result'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_scorecard'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_lgd'
scr_export(
  x,
  dir,
  stamp = TRUE,
```

```
  validation = NULL,
  elbe = NULL,
  tag = "model",
  ...
)

## S3 method for class 'scr_maturity'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_mix_shift'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_operating'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_overlap'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_pd'
scr_export(x, dir, stamp = TRUE, validation = NULL, ...)

## S3 method for class 'scr_rag'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_score_cross'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_segments'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_study'
scr_export(x, dir, stamp = TRUE, rag = NULL, ...)

## S3 method for class 'scr_uplift'
scr_export(x, dir, stamp = TRUE, ...)
```

Arguments

x	An object from scr_select(), scr_scorecard(), scr_coarse_classing(), scr_pd(), scr_lgd(), scr_ead(), scr_capital(), scr_bands(), scr_tiers(), scr_rag(), scr_claims(), scr_operating(), scr_score_cross(), scr_mix_shift(), scr_segments(), scr_maturity(), scr_uplift(), scr_overlap() or scr_detection().
dir	Output directory. Created if it does not exist.
stamp	If TRUE (default), writes to a timestamped subdirectory, preserving earlier runs.
...	For scr_scorecard: precomputed cutoff, strategy, reject and monitor objects, and revenue_good/loss_bad for the default strategy table.
validation	For the IRB models (scr_pd, scr_lgd, scr_ead): the matching validation object (scr_pd_validate(), scr_lgd_validate(), scr_ead_validate());

NULL runs it on the hold-out where possible.

tag — For scr_lgd and scr_ead: the file tag (lgd_<tag>.xlsx, ead_<tag>.xlsx). scr_pd names its files after the target (pd_<target>.xlsx) and scr_capital after the framework (capital_<framework>.xlsx).

elbe — For scr_lgd: an scr_elbe() object; NULL computes it.

rag — For scr_study: an optional scr_rag() object whose lights are written to the same workbook.

Details

scorecard_<target>.xlsx Score_Summary (with odds_orientation), Final_Scorecard, Coefficients, Sign_Check, Alignment, Alignment_Bands, Model_Card, Challenger and Swap_Set (when a challenger exists), Coarse_Classing and Decision_Ledger (after a lab commit).

validation_<target>.xlsx Score_Gains_Frozen, Variable_Gains_IV, Discrimination_CI, Stability_PSI_Timeline, Stability_CSI_Timeline, Stability_Variables, Calibration, Calibration_Bands, Performance_By_Vintage, Rank_Order_Diagnostics.

strategy_<target>.xlsx Population_Scope, Band_Coverage, Cutoff_Sweep, Strategy_Bands, Reject_Sensitivity, Monitoring_Plan.

For an scr_classing lab: one workbook (classing_<target>.xlsx) with the specification, the bins, the checks and the decision ledger. The IRB models write one workbook and one SQL file each (pd_<target>.xlsx, lgd_<tag>.xlsx, ead_<tag>.xlsx, capital_<framework>.xlsx), with the validation, the ledger and the model card as sheets.

A score study (scr_bands(), scr_tiers()) writes one workbook, study_bands_<target>.xlsx or study_tiers_<target>.xlsx, with its summary, its table, the cuts and the settings (plus the ledger and the stability of the tiers, and the lights of rag when given); a set of lights from scr_rag() writes rag_<target>.xlsx; claims from scr_claims() write claims_<target>.xlsx; an operating point from scr_operating() writes operating_<target>.xlsx (curve, optimum, constraints); two crossed scores from scr_score_cross() write score_cross_<score_a>_<score_b>.xlsx (cross table, overlap, overlap rates, association). The other score studies write one workbook each: mix_shift_<target>.xlsx from scr_mix_shift(), segments_<target>.xlsx from scr_segments(), maturity_<event>.xlsx from scr_maturity(), uplift_<target>.xlsx from scr_uplift(), overlap_<target>.xlsx from scr_overlap() and detection_<target>.xlsx from scr_detection().

The timeline and vintage sheets need the date column of the split; when it is absent they carry an availability row instead of a fabricated number.

Value

The object x, with $files filled, invisibly.

See Also

Other production: predict.scr_align(), scr_apply(), scr_monitor(), scr_monitoring_plan(), scr_reasons(), scr_sql()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
out <- file.path(tempdir(), "scorecraft-example")
res <- scr_export(res, out, stamp = FALSE)
basename(unlist(res$files))
sc <- scr_export(scr_scorecard(res), out, stamp = FALSE)
basename(unlist(sc$files))
```

`scr_fetch` *Fetch a table with reproducible server-side sampling*

Description

Sampling happens on the server, not in R: pulling a million rows to discard ninety percent of them pays the network cost twice. `max_rows` is a memory guard: when it binds, the requested fraction is reduced on the server and the reduction is reported. The random expression follows the connection class (`rand(seed)` on Spark/Databricks/MySQL, `random()` on PostgreSQL/DuckDB, an integer-modulo expression on SQLite, which has no seedable `random()`); pass `sample_expr` to override it.

Usage

```
scr_fetch(
  con,
  table,
  sample_frac = 1,
  seed = NULL,
  max_rows = NULL,
  sample_expr = NULL,
  verbose = NULL
)
```

Arguments

`con`	A DBI connection, from `scr_connect()`.
`table`	Qualified table name.
`sample_frac`	Fraction of rows to fetch, in (0, 1].
`seed`	Seed of the server-side random function, where supported.
`max_rows`	Row cap. `NULL` switches it off.
`sample_expr`	Optional SQL expression yielding a uniform number in `[0, 1)`, used as `WHERE <sample_expr> <= sampl`
`verbose`	`TRUE`/`FALSE` to echo (or not) the query for this call; `NULL` follows `scr_verbose()`.

Value

A `data.table` with the fetched table.

See Also

Other database: `scr_connect()`

Examples

```
con <- scr_connect(driver = RSQLite::SQLite(), dbname = ":memory:")
d <- scr_demo; d$ref_date <- as.character(d$ref_date)
DBI::dbWriteTable(con, "dtm", d)
nrow(scr_fetch(con, "dtm", sample_frac = 0.5, seed = 42))
nrow(scr_fetch(con, "dtm", max_rows = 1000))
DBI::dbDisconnect(con)
```

---

scr_funnel *Audit funnel: every input variable and its fate*

Description

The central deliverable. One row per input column, plus one per derived variable, with the descriptive profile, the verdict of every gate, the votes of every model and `exit_stage`, the exact stage at which the variable failed. No candidate disappears from the report.

Usage

```
scr_funnel(x, only_selected = FALSE, cols = "essentials")
```

Arguments

`x`	An object from `scr_select()`.
`only_selected`	If `TRUE`, returns only the approved ones.
`cols`	`"essentials"` (default) gives a lean view; `"all"` gives everything; or pass a vector of names.

Value

A `data.table` ordered with the approved first.

Values of `exit_stage`

`00.config` Never competed: it was in `drop`.

`01.triage` Constant, near-constant, high cardinality, exact duplicate, missing share above the ceiling, or no signal in the coarse IV.

`02.binning` The binning algorithm failed on this column.

`03.screening` Failed one of the eight admission rules.

`04.holdout` IV dropped out of sample, unstable PSI, or part of the hold-out falls in no bin.

`05.correlation` Redundant with a better-ranked variable.

`05b.derived_excluded` Passed everything, but is a column the pipeline created and `allow_derived_final = FALSE`.

`06.consensus` Not enough votes, or outside the top-N.

`07.approved` Entered the shortlist.

`08.manual_drop` In the automatic consensus, but removed by the analyst in the coarse classing lab (`scr_classing_choose()`).

See Also

Other accessors: `scr_gains()`, `scr_leakage()`, `scr_result`, `scr_score_gains()`, `scr_score_metrics()`, `scr_selected()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
head(scr_funnel(res, only_selected = TRUE))
table(scr_funnel(res, cols = "all")$exit_stage)
```

`scr_gains` *Gains table, at bin level*

Description

One row per variable and bin, with counts, event rate, WOE, IV, lift, cumulative KS, precision and recall, plus the hold-out IV and the PSI of the variable.

Usage

```
scr_gains(x, only_selected = TRUE)
```

Arguments

`x`	An object from `scr_select()`.
`only_selected`	If `TRUE` (default), only the approved variables.

Value

A `data.table` at bin level.

See Also

Other accessors: `scr_funnel()`, `scr_leakage()`, `scr_result`, `scr_score_gains()`, `scr_score_metrics()`, `scr_selected()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
g <- scr_gains(res)
g[feature == scr_selected(res)[1], .(bin, count, pos_rate, woe, iv)]
```

---

scr_grades *Rating grades on the score*

Description

Cuts the production score into grades whose PD is monotone. The grade boundaries are score cut points, direction-aware: grade 1 is the safest (the highest scores under `higher_is_safer`). Three constructions: `"geometric"` builds a `scr_master_scale()` between percentiles 1 and 99 of the calibrated PD and converts its PD bounds into scores through the calibrated alignment; `"quantile"` cuts equal-count score bands (cut points moved half-way between neighboring scores, so a boundary never sits on an observed value); `"supplied"` grades by the PD bands of a given master scale.

Usage

```
scr_grades(
  x,
  calibration = NULL,
  master_scale = NULL,
  n_grades = NULL,
  method = NULL,
  min_obligors = NULL,
  min_defaults = NULL,
  monotone = TRUE,
  pd_source = NULL,
  sample = "holdout",
  dr = NULL
)
```

Arguments

x	An scr_scorecard().
calibration	An scr_calibrate() object (or its alignment); NULL uses the scorecard's own alignment.
master_scale	An scr_master_scale() for method = "supplied" (optional for "geometric").
n_grades, method, min_obligors, min_defaults, pd_source	NULL reads pd_n_grades, pd_grade_method, pd_min_obligors, pd_min_defaults and pd_source from the scorecard configuration.
monotone	Repair non-monotone grade PDs by pooling.
sample	Sample of the scorecard used to build the grades.
dr	Optional scr_dr with a grade column keyed by the final grades (see the section above).

Details

Grades below min_obligors obligors or min_defaults defaults are merged with the neighbor of closer default rate; the sequence of grade PDs is then repaired by pool-adjacent-violators when monotone = TRUE, and every merge is recorded in repairs. The grade PD (pd_be) is the long-run average of the grade default rates when a default-rate series by grade is given in dr (pd_source = "lra"), the sample default rate of the grade otherwise, or the mean of the calibrated individual PDs (pd_source = "mean_pd"). Concentration is reported as the Herfindahl index, the coefficient of variation of the grade shares and the Herfindahl-based hi index.

Value

An object of class scr_grades: table (grade, label, score_lo, score_hi, pd_lo, pd_hi, n, share, defaults, dr, pd_mean, pd_be, merged_from, and n_series, t_series when a series is given), breaks (ascending score cut points), band_grade (grade of every score band, ascending), direction, method, pd_source, master_scale, alignment (calibrated), alignment_score (the scorecard's), concentration (hhi, cv, hi, k), repairs, ledger, moc (empty, filled by scr_moc()), dr (the pooled series), rows (score, outcome and grade of the sample), scorecard, sample, ct, sample_rate. Also calibration (the scr_pd_calibration when one was given), n_grades_requested, min_obligors, min_defaults, target and config.

Two-pass workflow with a default-rate series

The series must be keyed by the final grades of this same call. Run scr_grades() once, grade the cohort panel with predict.scr_grades(), build the series with scr_default_rate() (grade =) and pass it as dr in a second call with identical arguments (or in scr_moc() and scr_pd_validate(), which read it the same way).

See Also

Other irb-pd: predict.scr_grades(), predict.scr_pd(), scr_calibrate(), scr_master_scale(), scr_migration(), scr_moc(), scr_pd(), scr_pd_pit_ttc(), scr_pd_validate()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
cal <- scr_calibrate(sc, target = 0.06)
gr <- scr_grades(sc, cal, n_grades = 7, min_defaults = 10)
gr
gr$table[, c("grade", "score_lo", "score_hi", "n", "dr", "pd_be")]
# grade a cohort panel with the score cut points (the panel score is a
# different scale; the demo only shows the mechanics)
head(predict(gr, score = scr_demo_panel$score))
```

---

scr_irb_params *IRB parameter tables by framework preset*

Description

Returns the numeric tables the internal ratings-based (IRB) functions read: probability of default (PD) floors, loss given default (LGD) input floors for own estimates, supervisory LGD values of the foundation approach, standardized credit conversion factors (CCF), asset-correlation parameters of the risk-weight function, maturity rules, the output floor and the standardized risk weights used for the floor comparison. Three presets ship: `"bcb"` (Brazil, BCB Resolutions 303/2023 and 229/2022), `"basel3_final"` (the consolidated Basel Framework in force from 2023) and `"crr3"` (the EU text applicable from 2025). The presets differ in a handful of cells, all visible with `print()`; users who need another jurisdiction edit the tables and pass the object to the functions that take `params`.

Usage

```
scr_irb_params(framework = c("bcb", "basel3_final", "crr3"))
```

Arguments

`framework`	`"bcb"`, `"basel3_final"` or `"crr3"`.

Value

An object of class `scr_irb_params`: a list with `framework`, `source` (one line), `pd_floor`, `lgd_floor`, `lgd_firb`, `ccf_sa`, `ccf_floor_fraction`, `correlation`, `scaling_factor`, `confidence`, `m_default`, `m_range`, `output_floor`, `sa_rw` and `modified` (logical, set by the functions that receive the object when its tables were edited).

Regulatory texts

The presets encode a reading of the texts below at the time of the release. Regulation changes and is interpreted by each supervisor: before any regulatory use, check every table against the texts in force for the jurisdiction and the portfolio, and edit the tables where they differ. The package implements the calculations; it does not give regulatory advice.

References

Basel Committee on Banking Supervision. *The Basel Framework*, chapters CRE31 to CRE36 (IRB approach) and RBC20 (output floor).

Regulation (EU) 2024/1623 (CRR3), amending Regulation (EU) No 575/2013.

European Banking Authority (2017). *Guidelines on PD estimation, LGD estimation and the treatment of defaulted exposures*, EBA/GL/2017/16.

European Banking Authority (2019). *Guidelines for the estimation of LGD appropriate for an economic downturn*, EBA/GL/2019/03.

Banco Central do Brasil. BCB Resolution 229/2022 and BCB Resolution 303/2023.

International Accounting Standards Board (2014). *IFRS 9 Financial Instruments*.

See Also

Other irb-parameters: `scr_default()`, `scr_default_rate()`

Examples

```
p <- scr_irb_params("bcb")
p
p$pd_floor
p2 <- p; p2$pd_floor$floor[p2$pd_floor$asset_class == "retail_other"] <- 0.001
```

`scr_irb_rw` *IRB risk weight of one or many exposures*

Description

The asymptotic single risk factor function, vectorized over exposures: PD floors by asset class, LGD input floors for own estimates (`approach = "airb"`; the unsecured column of `params$lgd_floor` unless `collateral` names another column, blended with `secured_share`), the asset correlation of the class (with the firm-size adjustment of `corporate_sme` from `sales` and the multiplier for large or unregulated financial institutions when `fi` is `TRUE`), the maturity adjustment for wholesale classes only (`m` clipped to `params$m_range`, `params$m_default` when missing or under the foundation approach) and

Usage

```
scr_irb_rw(
  pd,
  lgd,
  ead = 1,
  m = NULL,
  asset_class,
  sales = NULL,
  fi = FALSE,
```

```
  defaulted = NULL,
  elbe = NULL,
  params = scr_irb_params("bcb"),
  approach = c("airb", "firb"),
  apply_floors = TRUE,
  collateral = NULL,
  secured_share = NULL,
  claim = NULL
)
```

Arguments

pd, lgd, ead	Numeric vectors: probability of default, loss given default (decimals) and exposure at default (currency).
m	Effective maturity in years, non-negative (wholesale classes only, ignored and reported as NA on retail rows; NULL or NA uses params$m_default).
asset_class	One of "corporate", "corporate_sme", "bank", "sovereign", "hvcre", "retail_mortgage", "qrre_revolver", "qrre_transactor", "retail_other"; a scalar or a vector.
sales	Annual sales of corporate_sme obligors, in the unit of params$correlation$sme, clipped to its bounds. A missing value takes the upper bound, i.e. no firm-size adjustment: the adjustment requires reported sales (Basel Framework CRE31.9; CRR Article 153(4)).
fi	Logical: regulated financial institution above the size threshold, or unregulated one (correlation multiplier).
defaulted	Optional 0/1 or logical vector.
elbe	Optional vector with the best estimate of expected loss of the defaulted rows (decimal of ead); ignored on performing rows.
params	An scr_irb_params() object.
approach	"airb" (own LGD, floored) or "firb" (supervisory LGD supplied by the caller, no LGD floor, maturity fixed).
apply_floors	TRUE (all input floors), FALSE (none) or a subset of c("pd", "lgd", "m").
collateral	Optional column of params$lgd_floor naming the collateral type of each exposure ("financial", "receivables", "real_estate", "other_physical"); NULL means unsecured.
secured_share	Optional secured share in [0, 1] blending the unsecured and the collateral floors.
claim	Under "firb", an optional claim type per exposure naming a row of params$lgd_firb (for example "senior_unsecured" or "subordinated"); the supervisory LGD of that row replaces lgd. NULL keeps the caller's lgd. The row names differ by preset ("senior_unsecured" under "bcb", "senior_unsecured_corporate" and "senior_unsecured_fi" otherwise); see params$lgd_firb.

Details

$$K = \left[LGD \cdot N\left(\frac{G(PD) + \sqrt{R}\,G(0.999)}{\sqrt{1-R}} \right) - PD \cdot LGD \right] \cdot MA \cdot s$$

with `s = params$scaling_factor` and, for wholesale classes, $MA = (1+(M-2.5)\,b)/(1-1.5\,b)$, $b = (0.11852 - 0.05478 \ln PD)^2$ (`MA = 1` for retail). Below `PD = 1e-5`, reachable only without a PD floor (sovereigns), `b` is held at its value at `1e-5`: the regulatory `b` makes `1 - 1.5 b` vanish near `PD = 2.9e-6`, where the adjustment would explode and change sign. Defaulted rows carry `K = max(0, LGD - ELBE)` under `"airb"` and zero under `"firb"`; a missing `elbe` is taken equal to `lgd`. `RW = 12.5 K` and `RWA = RW * ead`.

Value

A `data.table` with one row per exposure: `pd_used`, `lgd_used` (after floors; PD one on defaulted rows), `m` (after clipping; `params$m_default` under `"firb"`; `NA` on retail rows), `r`, `b`, `ma`, `k`, `rw`, `rwa`; attribute `floors_hit` counts the rows where each floor was binding.

References

Basel Committee on Banking Supervision (2023). *The Basel Framework*, CRE31 (IRB approach: risk-weight functions) and CRE32 (risk components). BCBS (2005). *An explanatory note on the Basel II IRB risk weight functions*.

See Also

Other irb-capital: `scr_capital()`, `scr_ecl()`, `scr_el()`, `scr_pd_stress()`, `scr_sa_rw()`

Examples

```
scr_irb_rw(0.01, 0.45, m = 2.5, asset_class = "corporate")
scr_irb_rw(c(0.01, 0.02), c(0.20, 0.80), asset_class = c("retail_mortgage", "qrre_revolver"))
r <- scr_irb_rw(1e-4, 0.5, asset_class = "retail_other")
attr(r, "floors_hit")
# foundation approach: the supervisory LGD of the claim type
scr_irb_rw(0.01, lgd = 0, m = 2.5, asset_class = "corporate", approach = "firb",
           claim = "senior_unsecured")
```

`scr_iv` *Information Value of any grouping*

Description

Laplace smoothing by default. It is not cosmetic: without it, a single-class group (a normal situation in a small sentinel population) yields `Inf` and contaminates any ordering that depends on the IV.

Usage

```
scr_iv(g, y, laplace = 0.5)
```

Arguments

g	Group vector (any coercible type). Rows where g or y is NA are ignored, whatever the type of g.
y	0/1 outcome vector.
laplace	Smoothing constant added to each count. 0 switches it off.

Details

Implemented with tabulate() on integer codes rather than data.table aggregation: this function is called once per candidate variable, hundreds of times per run, and the fixed cost dominated the triage.

Value

Total IV, a scalar. Zero when fewer than two groups are populated.

See Also

Other metrics: scr_metrics(), scr_psi()

Examples

```
set.seed(1)
y <- stats::rbinom(1000, 1, 0.3)
g <- ifelse(stats::runif(1000) < 0.5 + 0.3 * y, "A", "B")
scr_iv(g, y)
```

---

scr_leakage *Leakage and suspicious-strength audit*

Description

Separates what the pipeline failed for excessive strength (IV_SUSPICIOUS, DEGENERATE_BIN) from what it admitted but deserves a second look (IV above config$iv_suspect). A bin with no events or no non-events is the symptom with no innocent explanation: the variable determines the outcome on part of the population.

Usage

```
scr_leakage(x, threshold = NULL)
```

Arguments

x	An object from scr_select().
threshold	Warning threshold. NULL uses config$iv_suspect.

Value

An scr_leakage object (a list with barred, degenerate, approved_suspect), with a print method.

See Also

Other accessors: scr_funnel(), scr_gains(), scr_result, scr_score_gains(), scr_score_metrics(), scr_selected()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
scr_leakage(res)
```

---

scr_lgd *Two-stage LGD model and pools on the reference data set*

Description

Fits the standard two-stage structure on the RDS of scr_workout():

$$\mathrm{LGD} = P(\text{cure} \mid x)\,\mathrm{LGD}^{\text{cure}} + \big(1 - P(\text{cure} \mid x)\big)\,\mathrm{E}[\mathrm{LGD} \mid \text{no cure}, x]$$

The **cure stage** is a binary model on is_cure with the scorecard machinery: optimal binning of the drivers on the training cohorts, WOE, hold-out revalidation with frozen bins and a logistic regression on the WOE columns with the sign check (every coefficient positive). The **severity stage** bins the same drivers against the realized LGD of the non-cures with scr_bin_continuous() (bin means, monotone, at least lgd_min_defaults_bin defaults per bin, hold-out revalidated) and fits a fractional logit (glm with a quasi-binomial family on the bin means) or, with lgd_severity = "beta", a beta regression through the betareg package. LGD^cure is the mean realized LGD of the cures on train (costs and the discount effect, never zero by decree).

Usage

```
scr_lgd(
  x,
  drivers,
  config = scr_config(),
  holdout = 0.3,
  date_col = "default_date"
)
```

Arguments

x	An scr_workout() object.
drivers	Column names of the RDS to use as drivers.
config	A scr_config(); keys lgd_*, the binning and hold-out keys of stage 2, max_abs_coef, n_boot, ci_level, seed, nthread.
holdout	Share of the cohorts held out (by default date).
date_col	Column of the RDS with the default date.

Details

The split is by cohort of default: the last holdout share of the default dates is the hold-out. Metrics on both samples: RMSE, MAE, R-squared, Spearman rho, Somers' D of the prediction with respect to the realized LGD (generalized AUC (D + 1) / 2) with a bootstrap confidence interval, and the loss capture ratio. Pools come from scr_lgd_pools(). The object carries a provisional downturn (type 3 add-on, or none, by configuration) and no floor until scr_lgd_downturn() and scr_lgd_floor() run.

Value

An object of class scr_lgd: split, drivers, cure (fit, features, coef, sign_check, bins, holdout), severity (fit, features, coef, engine, sign_check, bins), lgd_cure, has_cures, scored (one row per default: sample, p_cure, severity, lgd_pred, pool, lgd_real), bins_idx, samples (predicted vs realized by decile of the prediction), metrics, pools, downturn, floors, workout (the profile and summary of the RDS), model_card, ledger, config.

See Also

Other irb-lgd: scr_elbe(), scr_lgd_downturn(), scr_lgd_floor(), scr_lgd_pools(), scr_lgd_validate(), scr_workout()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max", "months_on_book", "region"),
             config = cfg)
m
m$pools[, c("pool", "n", "lra", "lra_ew", "moc_c", "lgd_dt")]
m$metrics
```

`scr_lgd_downturn` *Downturn LGD per pool*

Description

Quantifies the downturn per pool from user-supplied downturn periods. `method = "type1"` (observed impact): the default-weighted realized LGD of the training defaults whose default date falls inside the periods; a pool with fewer than ten such defaults falls back to type 3. `method = "type3"`: the long-run average plus `add_on`. `method = "none"`: the long-run average. The reference value (a challenger, not a bound) is the mean of the two worst calendar years of the pool. Both use the training rows only, so the hold-out stays independent evidence. The downturn LGD used for capital is

$$\mathrm{LGD}^{DT} = \min\bigl(1,\ \max(\mathrm{LRA} + \mathrm{MoC},\ \mathrm{DT} + \mathrm{MoC})\bigr)$$

and the impact `LGD^DT - min(1, LRA + MoC)` is reported per pool.

Usage

```
scr_lgd_downturn(
  x,
  periods = NULL,
  method = NULL,
  add_on = NULL,
  reason = NULL
)
```

Arguments

`x`	An `scr_lgd()` object.
`periods`	A table with `start` and `end` dates of the downturn periods. Required for `"type1"`.
`method`	`"type1"`, `"type3"` or `"none"`; `NULL` uses `lgd_downturn`.
`add_on`	Type-3 add-on; `NULL` uses `lgd_downturn_add_on`.
`reason`	Free text recorded in the ledger, mandatory: the choice of periods and method is an analyst decision.

Value

The `scr_lgd` object with `downturn` (`table` per pool: `lra`, `moc_c`, `dt_observed`, `n_downturn`, `dt_type3`, `reference_value`, `method_used`, `dt`, `lgd_dt`, `impact`, `below_reference`; `periods`, `method`, `add_on`, `status`, `reason`) and the pool columns `lgd_dt` and `lgd_final` updated.

See Also

Other irb-lgd: `scr_elbe()`, `scr_lgd()`, `scr_lgd_floor()`, `scr_lgd_pools()`, `scr_lgd_validate()`, `scr_workout()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
m <- scr_lgd_downturn(m, periods = data.frame(start = as.Date("2022-01-01"),
                                              end = as.Date("2023-12-31")),
                      reason = "reference rate above 13% in 2022-2023")
m$downturn$table
```

`scr_lgd_floor` *Input floor on the downturn LGD per pool*

Description

Applies the LGD input floor of the framework's parameter table, blended between the unsecured and the collateralized floor with the secured share of the exposure:

$$\text{floor} = \text{floor}_U\,(1 - s) + \text{floor}_S\,s, \qquad \text{LGD}^{\text{final}} = \max(\text{LGD}^{DT}, \text{floor})$$

A missing unsecured floor (residential mortgages, whose floor applies to the whole exposure) uses the collateral floor throughout; an asset class with no floor at all yields a floor of zero.

Usage

```
scr_lgd_floor(
  x,
  params = NULL,
  asset_class = NULL,
  secured_share = NULL,
  collateral = c("real_estate", "financial", "receivables", "other_physical")
)
```

Arguments

`x`	An `scr_lgd()` object.
`params`	An `scr_irb_params()` object; `NULL` uses the configured framework. Edits are detected and recorded in the ledger.
`asset_class`	Row of `params$lgd_floor`; `NULL` uses `asset_class` of the configuration.
`secured_share`	Secured share of the exposure in `[0, 1]`: one value or one per pool. `NULL` means unsecured.
`collateral`	Column of `params$lgd_floor` for the secured part: `"real_estate"`, `"financial"`, `"receivables"` or `"other_physical"`.

Value

The `scr_lgd` object with `floors` (`table` per pool with `lgd_dt`, `floor_unsecured`, `floor_secured`, `secured_share`, `floor`, `lgd_final`, `binding`; `asset_class`, `collateral`, `framework`, `params_modified`, `binding_share`) and the pool columns `floor` and `lgd_final` updated.

See Also

Other irb-lgd: scr_elbe(), scr_lgd(), scr_lgd_downturn(), scr_lgd_pools(), scr_lgd_validate(), scr_workout()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
m <- scr_lgd_floor(m, asset_class = "retail_other", secured_share = 0.4)
m$floors$table
```

---

scr_lgd_pools *LGD pools from the predicted LGD*

Description

Cuts the training predictions into n_pools quantile bands, merges the bands with fewer than min_defaults defaults into the neighbor with the closer long-run average, then merges adjacent bands whose long-run averages break the increasing order (pool-adjacent violators), so that the pools are ordered both in predicted and in realized LGD. Per pool: the default-weighted long-run average (the regulatory estimate), the exposure-weighted one, the standard error, the category-C margin of conservatism (one-sided 95% t interval on the mean) and their sum.

Usage

```
scr_lgd_pools(x, n_pools = NULL, min_defaults = NULL)
```

Arguments

x	An scr_lgd() object.
n_pools	Target number of pools; NULL uses lgd_n_pools.
min_defaults	Minimum defaults per pool; NULL uses lgd_min_defaults_bin.

Value

A data.table with one row per pool: pool, pred_lo, pred_hi, pred_mean, n, share, ead, lra, lra_ew, sd, se, moc_c, lra_moc, merged_from.

See Also

Other irb-lgd: scr_elbe(), scr_lgd(), scr_lgd_downturn(), scr_lgd_floor(), scr_lgd_validate(), scr_workout()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
scr_lgd_pools(m, n_pools = 4)
```

---

scr_lgd_validate *Validation battery of an LGD model*

Description

Runs the three blocks of the usual LGD validation on the hold-out sample (or on `newdata`) against the training reference:

Usage

```
scr_lgd_validate(x, newdata = NULL)
```

Arguments

`x`	An `scr_lgd()` object.
`newdata`	`NULL` (the hold-out), an `scr_workout()` object or a table with the drivers, `lgd_real`, `ead` and the default date.

Details

- **Calibration.** Per pool and for the portfolio, the one-sided t-test of realized against estimated LGD (the pool long-run average), where under-estimation is the failure: `p = 1 - Phi(t)`; the loss shortfall `1 - sum(LGD_real E) / sum(LGD_pred E)`; the coverage of the realized mean by the downturn LGD; the regression of realized on predicted.
- **Discrimination.** Somers' D / generalized AUC of the prediction with its bootstrap interval, compared with the training value through `S = (gAUC_init - gAUC_curr) / sigma_curr`; Spearman rho; the loss capture ratio; R-squared.
- **Stability.** PSI of the pool distribution and of the bins of every driver of both stages, with the fixed and the n-adjusted threshold.
- **Pools.** Homogeneity within a pool (Welch test between the halves of the pool split at its median prediction; a small p-value means a pool that still discriminates) and heterogeneity between adjacent pools (Welch test; a large p-value means pools that do not differ).

Traffic lights use the p-value thresholds of `config$pd_lights` (shared with the PD validation) (red at or below the first, amber at or below the second) and the fixed PSI thresholds.

Value

An object of class `scr_lgd_validation`: `calibration` (per pool), `portfolio`, `discrimination`, `stability` (`pools`, `drivers`), `homogeneity`, `heterogeneity`, `summary` (test, statistic, p, light; the light is `"grey"` when the test has no result, and a row that sums up several pools or drivers is the worst of their lights: red, then amber, then green, otherwise grey), `sample`, `n`.

See Also

Other irb-lgd: scr_elbe(), scr_lgd(), scr_lgd_downturn(), scr_lgd_floor(), scr_lgd_pools(), scr_workout()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
v <- scr_lgd_validate(m)
v
v$calibration
```

---

scr_master_scale *Master scale of PD grades*

Description

A grade structure with geometric midpoints and geometric-mean boundaries:

$$PD_k = PD_1\, r^{k-1}, \quad r = (PD_K/PD_1)^{1/(K-1)}, \quad \text{bound}_k = \sqrt{PD_k\, PD_{k+1}},$$

so that every grade doubles (or multiplies by r) the PD of the one before. With method = "supplied" the table comes from the user: a numeric vector of midpoints (boundaries derived as the geometric means) or a data.frame with pd_lo and pd_hi (and optionally pd_mid, label). Grade 1 is always the safest.

Usage

```
scr_master_scale(
  pd_min = 3e-04,
  pd_max = 0.3,
  n_grades = 10L,
  method = c("geometric", "supplied"),
  grades = NULL,
  labels = NULL
)
```

Arguments

pd_min, pd_max	PD midpoints of the first and the last grade.
n_grades	Number of grades.
method	"geometric" (default) or "supplied".
grades	For "supplied": a numeric vector of midpoints or a data.frame with pd_lo and pd_hi.
labels	Optional grade labels (default "1", "2", ...).

Value

A data.table of class scr_master_scale with grade, label, pd_lo, pd_mid, pd_hi, and the attributes ratio (the geometric ratio between consecutive midpoints) and method.

See Also

Other irb-pd: predict.scr_grades(), predict.scr_pd(), scr_calibrate(), scr_grades(), scr_migration(), scr_moc(), scr_pd(), scr_pd_pit_ttc(), scr_pd_validate()

Examples

```
ms <- scr_master_scale(0.0005, 0.25, n_grades = 8)
ms
scr_master_scale(method = "supplied", grades = c(0.001, 0.01, 0.05, 0.20))
```

---

scr_maturity *Maturity of the event by score band*

Description

Follows the units of each score band over time and estimates, at each horizon, the cumulative share that has had the event, allowing for units whose follow-up ends before the horizon (censoring). It shows how fast each band matures, where the curve flattens (the performance window) and how the discrimination of the score changes with the horizon.

Usage

```
scr_maturity(x, ...)

## S3 method for class 'data.frame'
scr_maturity(
  x,
  score = "score",
  time = "time",
  event = "event",
  horizons,
  objective = "risk",
  direction = NULL,
  n_bands = 5L,
  cuts = NULL,
  level = 0.95,
  weight = NULL,
  max_cells = 1e+05,
  ...
)
```

Arguments

x	A data.frame with one row per unit.
...	Passed on to the methods; an unknown argument is an error.
score, time, event	Column names of the score, of the time to the event or to the end of the follow-up, and of the 0/1 event indicator.
horizons	Times at which the incidence is read, in the unit of time (positive numbers).
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).
direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
n_bands	Equal-share bands of the score (tie-safe, the event-richest first) when cuts is not given.
cuts	Optional ascending cuts of the score, or an object from scr_bands() or scr_tiers(), whose cuts, numbers and labels are then used, with its objective and direction.
level	Confidence level of the intervals.
weight	Optional column of non-negative case weights.
max_cells	Largest number of distinct score values kept exactly.

Value

An object of class c("scr_maturity", "list"):

table One row per band and horizon, event-richest band first, then all units (band = NA, label = "all"): band, label, n, horizon, at_risk, events, censored, incidence, se, lo, hi and pct_of_final.

discrimination One row per horizon: horizon, n (units with a complete window), events, auc and gini.

cuts, codes, labels The ascending cuts, and the band number and label of every interval in ascending score order.

horizons, level, objective, direction, score, time, event, n, n_rows, n_dropped, weighted, call The settings: n is the volume used (the sum of the weights), n_rows the rows used and n_dropped those left out.

Data

One row per unit (a loan, a customer), with the score at the origin, time and event:

time A non-negative number of periods (days, months) from the origin to the event, or to the end of the follow-up.

event 1 when the event was observed at time, 0 when the follow-up ended there without it (censored).

From a monthly panel, take per unit the months from its opening date to its first month in default (event = 1), or to its last observed month when it never defaulted (event = 0); the example does it for scr_demo_panel.

Rows with a missing or infinite score, a missing time or event, or a zero weight are left out (n_dropped).

Incidence

Per band and for all units, the Kaplan-Meier estimate of the survival is $S(t) = \prod_{t_j \le t}(1 - d_j/n_j)$, with d_j the events at the distinct time t_j and n_j the units still followed just before it (units censored at t_j are at risk at t_j). `incidence` is $1 - S(h)$ at the horizon h, and `se` its standard error by Greenwood's formula, $S(h)\sqrt{\sum_{t_j \le h} d_j/(n_j(n_j - d_j))}$. The interval `lo`, `hi` is the complementary log-log interval of $S(h)$, $S^{\exp(\pm z\sigma)}$ with $\sigma = \sqrt{\sum d_j/(n_j(n_j - d_j))}/|\ln S|$, which stays inside [0, 1]. It is undefined when no event has happened by the horizon (`lo` = `0`, `hi` = `NA`) and when every unit has had the event (`NA`).

Under weights, d_j and n_j are weighted sums and each term of Greenwood's sum is computed on the Kish effective size of the risk set, $d_j/((n_j - d_j)\, n_j^{eff})$ with $n_j^{eff} = n_j^2/\sum w^2$; equal weights give the unweighted result.

Per band and horizon the table also counts `events` (events up to the horizon), `censored` (units censored before it) and `at_risk` (the other units: still followed at the horizon without the event), which add up to `n`. `pct_of_final` is the incidence over the incidence at the largest horizon: the share of the final events already seen.

Past the last follow-up time of a band nothing more is observed: the curve is carried flat, and a horizon beyond it repeats the last estimate with `at_risk` = `0`. Read such a row as "no information", not as a plateau of the event rate.

Discrimination

At each horizon, the units with a complete window are those with the event by the horizon and those followed for at least the horizon without it; units censored earlier are left out. `auc` and `gini` are those of the score for "event by the horizon" among them, from the counts per score value. The complete window ignores the censored units, so it is unbiased only when censoring does not depend on the score.

References

Greenwood, M. (1926). The natural duration of cancer. *Reports on Public Health and Medical Subjects*, 33, 1-26. HMSO.

Kalbfleisch, J. D. and Prentice, R. L. (2002). *The Statistical Analysis of Failure Time Data*, 2nd edition. Wiley. doi:10.1002/9781118032985

Kaplan, E. L. and Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282), 457-481. doi:10.1080/01621459.1958.10501452

See Also

`scr_bands()` and `scr_tiers()` for the bands, `scr_default()` and `scr_default_rate()` for the default flag and the cohort rates of a monthly panel.

Other score-studies: `scr_bands()`, `scr_claims()`, `scr_detection()`, `scr_mix_shift()`, `scr_operating()`, `scr_overlap()`, `scr_rag()`, `scr_rag_plan()`, `scr_score_cross()`, `scr_segments()`, `scr_tiers()`, `scr_uplift()`

Examples

```
# time and event from a monthly panel: months from the opening date to
# the first default (90 days past due), or to the last observed month
p <- scr_demo_panel[order(scr_demo_panel$id, scr_demo_panel$ref_date), ]
month <- 12 * as.integer(format(p$ref_date, "%Y")) + as.integer(format(p$ref_date, "%m"))
first <- !duplicated(p$id)
bad <- which(p$dpd >= 90)
bad <- bad[!duplicated(p$id[bad])]
u <- data.frame(id = p$id[first], score = p$score[first], open = month[first])
u$event <- as.integer(u$id %in% p$id[bad])
u$end <- as.vector(tapply(month, p$id, max)[as.character(u$id)])
u$end[u$event == 1] <- month[bad][match(u$id[u$event == 1], p$id[bad])]
u$time <- u$end - u$open

mt <- scr_maturity(u, horizons = c(6, 12, 24, 35), n_bands = 4)
mt
mt$discrimination
```

---

scr_metrics *AUC, KS and Gini of a score, with a bootstrap confidence interval*

Description

AUC through the Mann-Whitney U statistic with tie correction, computed on the table of counts per unique score: a WOE score is constant within the bin, so ties are the rule. Everything in `double` on purpose: with integer counts, `n1 * n0` overflows `2^31` from about 46 thousand observations per class and returns `NA`.

Usage

```
scr_metrics(
  score,
  y,
  higher_is_event = TRUE,
  ci = TRUE,
  n_boot = 200L,
  level = 0.95,
  seed = NULL,
  nthread = 1L
)
```

Arguments

`score`	Numeric vector with the score.
`y`	0/1 outcome vector (numeric or logical), same length as `score`. `NA` rows are dropped; any other value is an error.

`higher_is_event`
: If `TRUE` (default), a higher score means a higher probability of the event (logit, probability, propensity score). Pass `FALSE` for a credit points score (`higher_is_safer`), and the AUC is reported above 0.5 when the score ranks correctly.

`ci`
: Compute the confidence interval. `FALSE` returns point estimates only.

`n_boot`
: Number of bootstrap resamples.

`level`
: Confidence level.

`seed`
: Bootstrap seed, local to the call (the user's random stream is restored on exit); `NULL` draws from the user's stream.

`nthread`
: Parallel workers for the resamples on the rows; not used when the resamples are drawn on the counts (see Details). The result does not depend on it.

Details

The confidence interval is **always** computed by default: a bootstrap stratified by outcome, percentile method, with `n_boot` resamples. Gini is derived from AUC (`2 * AUC - 1`) inside each resample, never bootstrapped separately. DeLong's analytic variance is not used: the interval is a stratified percentile bootstrap, which also covers KS.

The resamples are drawn in one of two ways. With K distinct scores in the n rows used (those with a score and an outcome) and $K \leq n/10$ (many ties: scorecard points, a grade scale, a WOE score on a large sample), each class is redrawn on the counts, as a multinomial over the score values with its observed shares, at a cost of $O(K)$ per resample. Otherwise the rows of each class are resampled, at $O(n)$ per resample, over `nthread` workers. Rows of one class with the same score are exchangeable, so both are the same bootstrap stratified by outcome and differ only in how the draws are made: for a given seed, the bounds depend on which of the two ran.

The AUC is computed from the counts per unique score after one sort, so its cost is $O(n \log n)$, never the $O(n_1 n_0)$ of the pairwise definition.

Value

A list of class `scr_metrics` with `auc`, `ks`, `gini`, the bounds `auc_lo`/`auc_hi`, `ks_lo`/`ks_hi`, `gini_lo`/`gini_hi` (`NA` when `ci = FALSE`), `n`, `events`, `n_boot` and `level`. Everything is `NA_real_` when only one class is present or no valid case exists.

References

DeLong, E. R., DeLong, D. M. and Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves. *Biometrics*, 44(3), 837-845.

See Also

Other metrics: `scr_iv()`, `scr_psi()`

Examples

```
set.seed(1)
y <- rep(0:1, each = 500)
s <- stats::rnorm(1000) + 0.8 * y
```

```
m <- scr_metrics(s, y, n_boot = 50, seed = 1)
m
as.data.frame(m)
```

`scr_migration` *Migration matrix between two rating dates*

Description

Counts `N_ij` of obligors in grade `i` at the first date and grade `j` at the second, the row probabilities `p_ij`, the upper and lower matrix weighted bandwidths

$$MWB_{up} = \frac{\sum_{i<j} |i-j|\, N_i\, p_{ij}}{\sum_i \max(|i-K|, |i-1|)\, N_i \sum_{j>i} p_{ij}},$$

(and the mirror image for downgrades), the `z` statistic of every off-diagonal cell against its neighbor closer to the diagonal (a significantly positive value means the probability does not decay away from the diagonal) and the mobility summary. Values of `grade_t1` outside `1..K` count as `default`, `NA` as `closed`; both stay out of the bandwidths.

Usage

```
scr_migration(grade_t0, grade_t1, K = NULL)
```

Arguments

`grade_t0`, `grade_t1`	Integer grades at the two dates, same length.
`K`	Number of grades; `NULL` uses the largest grade observed.

Value

An object of class `scr_migration`: `matrix` (counts, `K` rows, `K + 2` columns), `p` (row probabilities), `n` (row totals), `mwb_upper`, `mwb_lower`, `z` (`K x K`), `n_significant` (cells with `z > 1.645`), `mobility` (`share_stable`, `share_up`, `share_down`, `mean_distance`, `share_default`, `share_closed`). Also `K`, the number of grades.

See Also

Other irb-pd: `predict.scr_grades()`, `predict.scr_pd()`, `scr_calibrate()`, `scr_grades()`, `scr_master_scale()`, `scr_moc()`, `scr_pd()`, `scr_pd_pit_ttc()`, `scr_pd_validate()`

Examples

```
set.seed(2)
g0 <- sample(1:5, 500, TRUE)
g1 <- pmin(5, pmax(1, g0 + sample(c(-1, 0, 0, 0, 1), 500, TRUE)))
g1[sample(500, 10)] <- NA
scr_migration(g0, g1, K = 5)
```

scr_mix_shift *Mix and rate effects of a change in the event rate*

Description

Splits the change of the event rate between a base and a comparison sample into the part due to the mix (the population moved across the score bands) and the part due to the rates (the bands themselves have a different event rate), on bands frozen on the base. With by, every period or segment is compared with the base.

Usage

```
scr_mix_shift(x, ...)

## S3 method for class 'scr_study'
scr_mix_shift(x, base = NULL, compare = NULL, level = NULL, ...)

## S3 method for class 'scr_scorecard'
scr_mix_shift(
  x,
  base = NULL,
  compare = NULL,
  by = NULL,
  n_bands = NULL,
  breaks = NULL,
  level = NULL,
  max_cells = 1e+05,
  ...
)

## S3 method for class 'data.frame'
scr_mix_shift(
  x,
  base = NULL,
  compare = NULL,
  by = NULL,
  n_bands = NULL,
  breaks = NULL,
  level = 0.95,
  score = "score",
  y = "y",
  objective = "risk",
  direction = NULL,
  weight = NULL,
  counts = FALSE,
  n = "n",
  events = "events",
```

```
  max_cells = 1e+05,
  ...
)
```

Arguments

x	An object from scr_bands(), scr_tiers() or scr_scorecard(), or a data.frame with one row per scored case (or one row per score value with counts = TRUE).
...	Passed on to the methods; an unknown argument is an error.
base	The base: a sample or a value of by (see the section Input). NULL takes the reference of a study, "train" for a scorecard, and the first value of by otherwise.
compare	The comparisons: samples or values of by. NULL takes every one but the base.
level	Confidence level: 1 - level is the significance of the PSI critical value and of the count of changed bands. NULL uses the level of the study or config$study_level (0.95).
by	Name of the column of periods or segments. Needed for a data.frame; for a scorecard, a column of its scored samples.
n_bands	Equal-share bands frozen on the base. NULL uses config$study_bands (20) for a scorecard and 10 for a data.frame.
breaks	Explicit ascending cut points; overrides n_bands.
max_cells	Largest number of distinct score values kept exactly.
score, y	Column names of the score and of the 0/1 outcome (NA allowed).
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).
direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
weight	Optional column of non-negative case weights.
counts	TRUE when x is pre-aggregated: one row per group and score value with the columns score, n and events.
n, events	Column names of the counts when counts = TRUE.

Value

An object of class c("scr_mix_shift", "list"):

table One row per comparison and band, event-richest band first: group, band, label, n_base, n_cmp (rows with a known outcome), pct_base, pct_cmp, rate_base, rate_cmp, mix_effect, rate_effect, total, p_rate and p_rate_adj.

summary One row per comparison: group, n_base, n_cmp, rate_base, rate_cmp, delta, mix_total, rate_total, share_mix, psi, psi_critical and bands_changed (bands with p_rate_adj below 1 - level).

cuts, base, groups, by, level, objective, direction, target, call The cuts and the settings.

Decomposition

With p_k the share of band k among the rows with a known outcome and r_k its event rate, on the base (b) and on the comparison (c):

$$mix_k = (p_{c,k} - p_{b,k})(r_{b,k} + r_{c,k})/2, \qquad rate_k = (r_{c,k} - r_{b,k})(p_{b,k} + p_{c,k})/2.$$

The weights are the midpoints of the two samples, so the effects carry no interaction term and add up exactly: $\sum_k mix_k + \sum_k rate_k = R_c - R_b$, the change of the overall rate. A band empty in one sample has no rate there; it takes the rate of the other sample, so its rate effect is 0 and its whole contribution is a mix effect (the columns `rate_base` and `rate_cmp` keep the missing rate as `NA`). `total` is `mix_effect` + `rate_effect`.

The summary adds, per comparison, `share_mix` = `mix_total` / `delta` (`NA` when the rate did not change; it can fall outside [0, 1] when the two effects have opposite signs) and the PSI of the band shares against the base with its n-adjusted critical value at `1 - level` (see `scr_psi()`).

Tests

`p_rate` is the two-sided p-value of the change of the rate of the band, on the unweighted counts: Fisher's exact test when the smallest expected count of the two-by-two table (sample by outcome) is below 5, and the pooled two-proportion z test otherwise. `p_rate_adj` is Holm-adjusted across the bands of the comparison. A band empty in either sample has no test.

Input

A score study An object from `scr_bands()` or `scr_tiers()` with at least two samples: its cuts and labels are used, `base` is one of its samples (default its reference) and `compare` the others.

A scorecard `base` and `compare` are scored samples (`"train"` against `"holdout"`). With `by`, a column of the scored samples such as `"date"`, the rows of both samples are pooled and grouped by that column; `base` is then one of its values (default the first) and `compare` the others.

A data.frame `by` names the column that tells the groups apart (a period, a segment or a sample label); `base` is one of its values (default the first level) and `compare` the others.

The values of `by` are read in the order of the levels of a factor, in numeric order for a numeric column and as sorted text otherwise (dates included); the first is the default base. Rows with a missing `by` value are left out. Rows with a missing outcome are left out of the shares and of the rates, so that the effects add up to the change of the rate.

References

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2), 65-70.

Yurdakul, B. and Naranjo, J. (2020). Statistical properties of the population stability index. *Journal of Risk Model Validation*, 14(4), 89-100.

See Also

`scr_bands()` for the bands, `scr_rag()` for the lights of a sample against its reference, `scr_segments()` for one score read on many segments.

Other score-studies: scr_bands(), scr_claims(), scr_detection(), scr_maturity(), scr_operating(), scr_overlap(), scr_rag(), scr_rag_plan(), scr_score_cross(), scr_segments(), scr_tiers(), scr_uplift()

Examples

```
local({
  set.seed(1)
  n <- 6000
  period <- rep(c("2025", "2026"), each = n / 2)
  # 2026 has riskier applicants (mix) and a higher rate at every score (rate)
  x <- rnorm(n, mean = ifelse(period == "2026", -0.3, 0))
  d <- data.frame(period = period, score = round(600 + 50 * x),
                  y = rbinom(n, 1, plogis(-2 - x + 0.3 * (period == "2026"))))
  ms <- scr_mix_shift(d, by = "period", n_bands = 5)
  print(ms)
  ms$table[, c("band", "label", "pct_base", "pct_cmp", "mix_effect", "rate_effect")]
})
```

---

scr_moc *Margin of conservatism, by category*

Description

Appends entries to the MoC ledger of an scr_grades() object. Category "C" (general estimation error) is quantified: "ci_timeseries" takes the upper bound of a one-sided level interval of the long-run average from the cohort series, $t_{q,T-1}\, sd(DR_t)/\sqrt{T}$ per grade; "ci_binomial" uses $z_q\sqrt{PD(1-PD)/n}$ on the obligors (or obligor-years when a series exists); "bootstrap" resamples the outcomes of the sample within each grade (drawn as the resampled default rate, Binomial(n, DR) / n, its exact distribution) and takes the level quantile of the default rate above the estimate. Categories "A" (data and methodological deficiencies) and "B" (changes in standards or environment) are expert quantities: value (one number or one per grade, in PD units) and a non-empty reason are mandatory. The ledger is append-only: A and B entries accumulate, a new C supersedes the previous one (kept with active = FALSE).

Usage

```
scr_moc(
  x,
  category = c("A", "B", "C"),
  method = NULL,
  level = NULL,
  value = NULL,
  reason = NULL,
  dr = NULL,
  n_boot = 200L,
  seed = NULL
)
```

Arguments

x	An scr_grades() object.
category	"A", "B" or "C".
method	For "C": "ci_timeseries", "ci_binomial" or "bootstrap"; NULL reads config$pd_moc_method.
level	One-sided confidence level; NULL reads config$pd_moc_level.
value	For "A"/"B": the add-on in PD units, length 1 or one per grade.
reason	Justification (mandatory for "A"/"B").
dr	Optional scr_dr by grade for "ci_timeseries", keyed by the final grades of x; NULL uses the series already stored in x$dr by scr_grades().
n_boot, seed	Bootstrap resamples and seed for "bootstrap".

Value

The scr_grades object with the entries appended to moc.

See Also

Other irb-pd: predict.scr_grades(), predict.scr_pd(), scr_calibrate(), scr_grades(), scr_master_scale(), scr_migration(), scr_pd(), scr_pd_pit_ttc(), scr_pd_validate()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
gr <- scr_grades(sc, n_grades = 6, min_defaults = 10)
gr <- scr_moc(gr, "C", method = "ci_binomial")
gr <- scr_moc(gr, "A", value = 0.002, reason = "missing unlikeliness-to-pay trigger before 2024")
gr$moc
```

---

scr_model *Stages 3 and 4: multi-strategy selection and consensus*

Description

Trains the classifiers enabled in the configuration on the WOE columns of the eligible pool, measures each on the hold-out (AUC/KS/Gini with a bootstrap CI) and combines the votes:

Usage

```
scr_model(bins, config = scr_config())
```

Arguments

bins	An object from scr_bin().
config	An object from scr_config().

Details

```
consensus_score = mean of the importance rank percentiles, weighted by the
                  hold-out Gini of each model
votes           = how many models elected the feature (top-K, or non-zero
                  coefficient in the elastic net)
```

The final cut respects [target_min, target_max]. If the strict consensus does not reach target_min, relaxation happens in **named**, recorded steps (min_votes reduced; completed by score), never resurrecting a feature failed by an earlier gate.

Value

An scr_models object with votes (one row per model and feature), metrics (one row per model, with CI), consensus (table, selected, meta) and the originating bins.

See Also

Other stages: scr_align(), scr_bin(), scr_cutoff(), scr_reject(), scr_scorecard(), scr_select(), scr_split(), scr_strategy(), scr_triage()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
sp <- scr_split(scr_demo, "default", date_col = "ref_date", drop = "id")
md <- scr_model(scr_bin(scr_triage(sp, cfg), cfg), cfg)
md
md$consensus$selected
```

---

scr_monitor *Monitor the scorecard on new data*

Description

Recomputes, per period of date_col (or for the whole data), the score PSI against train with frozen bands, the CSI of every variable with frozen bins plus the signed points shift and, when the target is present, the performance by vintage (event rate, AUC/KS/Gini with CI). Always reports both thresholds (fixed and n-adjusted). Schedules nothing: the analyst calls it when needed.

Usage

```
scr_monitor(
  x,
  newdata,
  date_col = NULL,
  target = NULL,
  alpha = NULL,
  n_boot = NULL,
  plan = NULL
)
```

Arguments

x	An object from scr_scorecard().
newdata	New table with the source columns.
date_col	Period column. NULL treats newdata as a single period. Rows with a missing date form a period of their own (NA, last).
target	Target column in newdata, for the performance by vintage. NULL skips it.
alpha	Level of the adjusted threshold. NULL (default) takes it from the plan.
n_boot	CI resamples per vintage. NULL uses the configuration.
plan	The monitoring contract: NULL (default) uses the plan stored in the scorecard (scr_monitoring_plan()); otherwise an item/value table, or the path of a strategy workbook written by scr_export(), whose Monitoring_Plan sheet is read. The fixed thresholds of the PSI and CSI flags, the alpha of the adjusted threshold and min_events_per_period come from it.

Value

An scr_monitor object with psi (score, per period), csi (per variable and period), vintage (or NULL; status says "insufficient" when a period has fewer events than the plan requires) and plan (the contract actually used).

See Also

Other production: predict.scr_align(), scr_apply(), scr_export(), scr_monitoring_plan(), scr_reasons(), scr_sql()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
mo <- scr_monitor(sc, scr_demo, date_col = "ref_date", target = "default")
mo
mo$psi
head(mo$csi)
```

scr_monitoring_plan *Monitoring plan read by scr_monitor()*

Description

A small item/value table with the thresholds and the frozen score bands of a scorecard. It is created by scr_scorecard() from the configuration, written to the Monitoring_Plan sheet of the strategy workbook by scr_export(), and **read back** by scr_monitor(): change a threshold in the sheet, pass the file (or the edited table) as plan, and the flags follow the plan, not the configuration.

Usage

```
scr_monitoring_plan(x, breaks = NULL)
```

Arguments

x	An object from scr_scorecard(), or a configuration from scr_config() plus breaks.
breaks	Frozen score bands, when x is a configuration.

Value

A data.frame of class scr_monitoring_plan with the items psi_score_fixed_moderate, psi_score_fixed_action, psi_adjusted_alpha, csi_variable_fixed_moderate, csi_variable_fixed_action, score_bands, min_events_per_period and threshold_source.

See Also

Other production: predict.scr_align(), scr_apply(), scr_export(), scr_monitor(), scr_reasons(), scr_sql()

Examples

```
plan <- scr_monitoring_plan(scr_config(), breaks = c(-Inf, 500, 550, 600, Inf))
plan
```

scr_operating *Operating point of a score under constraints*

Description

Accumulates the score from one end, one score value at a time, and finds the cut that maximizes the value of the decision under volume, budget, daily capacity and event-rate constraints: how many customers to target, how many alerts to raise, or how many applicants to approve.

Usage

```
scr_operating(x, ...)

## S3 method for class 'scr_scorecard'
scr_operating(
  x,
  side = NULL,
  gain_event = NULL,
  cost_select = 0,
  revenue_good = NULL,
  loss_bad = NULL,
  max_n = NULL,
  max_share = NULL,
  budget = NULL,
  max_per_day = NULL,
  day_quantile = 0.9,
  date = NULL,
  min_rate = NULL,
  max_rate = NULL,
  sample = "holdout",
  n_points = 200L,
  level = NULL,
  max_cells = 1e+05,
  ...
)

## S3 method for class 'data.frame'
scr_operating(
  x,
  side = NULL,
  gain_event = NULL,
  cost_select = 0,
  revenue_good = NULL,
  loss_bad = NULL,
  max_n = NULL,
  max_share = NULL,
  budget = NULL,
  max_per_day = NULL,
  day_quantile = 0.9,
  date = NULL,
  min_rate = NULL,
  max_rate = NULL,
  sample = NULL,
  n_points = 200L,
  score = "score",
  y = "y",
  objective = "risk",
  direction = NULL,
```

```
  weight = NULL,
  value = NULL,
  study = NULL,
  counts = FALSE,
  n = "n",
  events = "events",
  value_events = NULL,
  level = 0.95,
  max_cells = 1e+05,
  ...
)
```

Arguments

x	An object from scr_scorecard(), or a data.frame with one row per scored case (or one row per score value with counts = TRUE).
...	Passed on to the methods; an unknown argument is an error.
side	"event" or "safe"; NULL follows the objective and the direction (see the section Side).
gain_event	Gain per selected event (side "event").
cost_select	Cost per selected case (default 0).
revenue_good, loss_bad	Revenue per accepted non-event and loss per accepted event (side "safe").
max_n	Largest volume selected.
max_share	Largest share of the volume selected, in (0, 1].
budget	Largest cost, cost_select * n_sel; needs a positive cost_select.
max_per_day	Largest selected volume per day, read at the day_quantile quantile of the days; needs a date.
day_quantile	Quantile of the daily volume compared with max_per_day (0.9: nine days in ten within capacity).
date	For a data.frame: name of a date column, for the daily capacity. For a scorecard: a column of the scored sample; NULL uses its date column when present.
min_rate	Smallest event rate among the selected (side "event").
max_rate	Largest event rate among the accepted (side "safe").
sample	For a scorecard: the sample the curve is read on ("holdout"). For a data.frame: the name of a column with sample labels, as in scr_bands(); the curve is read on study.
n_points	About how many rows of the curve to keep.
level	Confidence level of the Jeffreys intervals. For a scorecard, NULL uses config$study_level (0.95).
max_cells	Largest number of distinct score values kept exactly.
score, y	Column names of the score and of the 0/1 outcome (NA allowed).
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).

direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
weight	Optional column of non-negative case weights.
value	Optional column of the value of every case: with no gain_event, the value of the selected events is the gain.
study	For a data.frame: the label of the sample the curve is read on; NULL takes the first label other than the first level (the reference of scr_bands()), or the only one.
counts	TRUE when x is pre-aggregated: one row per score value with the columns score, n and events (and, optionally, value and value_events).
n, events	Column names of the counts when counts = TRUE.
value_events	With counts = TRUE: the column of the value of the events per score cell.

Value

An object of class c("scr_operating", "list"):

curve The thinned curve (see the section Curve).

optimum One row: the columns of the curve at the optimum, binding, shadow_price and next_rate (the marginal rate of the next score value).

constraints One row per constraint given: constraint, limit, at_optimum (the constrained quantity at the optimum) and binding.

side, objective, direction, target, sample, level, gain_event, cost_select, revenue_good, loss_bad, day_quant The settings.

economics, value_column Whether the curve has a value, and whether it comes from the value column instead of gain_event.

select_high TRUE when the selection starts at the high scores (score >= cut), FALSE at the low ones (score < cut).

n_cells, n_days, quantized, weighted The number of candidate cuts (score values or pooled cells), the number of distinct dates (NA without a date), whether the scores were pooled into max_cells cells, and whether weights were used.

message NA, or the explanation of an infeasible or a loss-making optimum.

Side

side = "event" selects from the event-rich end of the score (targeting under propensity, alerting under fraud, collections under credit); side = "safe" accepts from the safe end (approval under credit). The default follows the objective and the direction: propensity selects from the event-rich end ("event"); risk with higher_is_riskier (fraud) alerts from the event-rich end ("event"); risk with higher_is_safer (credit) approves from the safe end ("safe").

The selected rows are score >= cut when the selection starts at the high scores and score < cut when it starts at the low ones, the convention of scr_cutoff(). Every cut sits between two adjacent distinct scores (or on a bucket edge when the scores were pooled into max_cells cells), as in scr_bands(); the last row of the curve selects every row (cut is -Inf or Inf).

Curve

One row per candidate cut, in increasing depth: `cut`, `depth` (share of the volume selected), `n_sel`, `events_sel`, `rate_sel` with its Jeffreys interval `rate_lo`, `rate_hi` (on the Kish effective size under weights), `capture` (share of all events selected), `lift` (`rate_sel` over the overall rate), `marginal_rate` (the event rate of the score value just added, smoothed by pool adjacent violators toward the event-rich end of the score), `cost` (`cost_select * n_sel`), `value` and `feasible`. With a date, `day_q` (the `day_quantile` quantile of the selected volume per day) and `pct_days_over` (share of days above `max_per_day`).

The economics:

- side `"event"`: `value = gain_event * events_sel - cost_select * n_sel`; with a `value` column and no `gain_event`, the sum of the value of the selected events replaces `gain_event * events_sel`.
- side `"safe"`: `value = revenue_good * nonevents_sel - loss_bad * events_sel - cost_select * n_sel`, the cumulative profit of `scr_strategy()` when `cost_select = 0` (a missing one of `revenue_good` and `loss_bad` counts as 0).

Without economics (`gain_event`, a `value` column, `revenue_good` or `loss_bad`), `value` is `NA`. Non-events are rows with a known outcome that are not events; rows with a missing outcome count in the volume and the cost only.

The curve is thinned to about `n_points` rows evenly spread in depth; the rows of the optimum, the last row meeting each constraint, the deepest row and the rows nearest 1%, 5%, 10%, 20% and 50% are always kept. The optimum and the constraints are evaluated on every cell boundary.

Constraints and optimum

`max_n` (`n_sel <= max_n`), `max_share` (`depth <= max_share`), `budget` (`cost_select * n_sel <= budget`), `max_per_day` (`day_q <= max_per_day`), `min_rate` (`rate_sel >= min_rate`, side `"event"`) and `max_rate` (`rate_sel <= max_rate`, the event rate among the accepted, side `"safe"`). A row is feasible when it meets every constraint given.

The optimum is the feasible row with the highest value (the smallest depth on a tie); without economics, the deepest feasible row. A constraint is `binding` when dropping it alone, the others kept, improves the optimum: a higher value, or a greater depth when there are no economics. A constraint that is slack at the optimum is therefore never named, and a constraint that stops the curve at the row that is the best anyway is not binding either. Constraints that stop the optimum at the same row bind jointly (none improves it alone) and are named together. When no constraint binds, `binding` is `"value"` with economics (no row is worth more than the optimum) and `"end of the curve"` without (every row is selected). `shadow_price` is the marginal value of the next score value beyond the optimum, per additional selected case: `gain_event * marginal_rate - cost_select` on side `"event"` (with a `value` column, the smoothed event value per case of that score value), `revenue_good * (1 - marginal_rate) - loss_bad * marginal_rate - cost_select` on side `"safe"`. It is what one more selected case is worth when a constraint binds; divide it by `cost_select` for the value of one more unit of budget. When no row is feasible, the optimum is `NA` and a warning names the constraints that the first row already breaks.

Daily capacity

A day is a distinct value of the date column: with a monthly date, read the capacity per month. For every candidate cut, the selected volume of each day is a cumulative sum over a table of counts per day and score value; `day_q` is its quantile across the days (type 7 of `stats::quantile()`). The quantile grows with the depth, so the deepest cut within `max_per_day` is found by bisection. Rows with a missing date count in the curve but not in the daily volumes. A scorecard uses the dates of its scored sample when they exist. A date-time column counts every distinct time as a day: convert it with `as.Date()` first.

References

Brown, L. D., Cai, T. T. and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133. doi:10.1214/ss/1009213286

Thomas, L. C., Crook, J. and Edelman, D. (2017). *Credit Scoring and Its Applications*, 2nd edition. SIAM. doi:10.1137/1.9781611974560

See Also

`scr_cutoff()` and `scr_strategy()` for the cut-off sweep and the strategy table of a scorecard, `scr_claims()` to test statements about the selected rates.

Other score-studies: `scr_bands()`, `scr_claims()`, `scr_detection()`, `scr_maturity()`, `scr_mix_shift()`, `scr_overlap()`, `scr_rag()`, `scr_rag_plan()`, `scr_score_cross()`, `scr_segments()`, `scr_tiers()`, `scr_uplift()`

Examples

```
set.seed(1)
x <- rnorm(5000)
d <- data.frame(score = round(500 + 50 * x),
                y = rbinom(5000, 1, plogis(-1.5 + 1.2 * x)),
                day = as.Date("2026-01-01") + sample(0:29, 5000, TRUE))
# targeting under propensity: a gain per responder, a cost per contact, a budget
op <- scr_operating(d, objective = "propensity", gain_event = 40, cost_select = 6,
                    budget = 6000)
op
op$optimum[, c("cut", "depth", "n_sel", "rate_sel", "value", "binding", "shadow_price")]

# alerting under a daily capacity: at most 40 alerts on nine days in ten
scr_operating(d, objective = "propensity", max_per_day = 40, date = "day")$optimum
```

---

scr_overlap *Overlap of rules and a score*

Description

Compares a set of rules (0/1 flags) with the alerts of a score at one cut: what each rule catches, what the score also catches, what only one of them catches, and which rules the score makes redundant. Built for fraud, where expert rules and a model alert on the same transactions.

Usage

```
scr_overlap(x, ...)

## S3 method for class 'data.frame'
scr_overlap(
  x,
  score = "score",
  y = "y",
  rules,
  alert_share = NULL,
  cut = NULL,
  value = NULL,
  objective = "risk",
  direction = "higher_is_riskier",
  retire_at = 0.95,
  level = 0.95,
  weight = NULL,
  max_cells = 1e+05,
  ...
)
```

Arguments

x	A data.frame with one row per case (a transaction).
...	Passed on to the methods; an unknown argument is an error.
score, y	Column names of the score and of the 0/1 outcome (NA allowed).
rules	Names of the rule columns: 0/1 numbers or logicals.
alert_share	Share of the rows the score alerts, in (0, 1].
cut	Instead of alert_share: the score cut of the alerts.
value	Optional column of a value per case (the amount).
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).
direction	"higher_is_riskier" (default, a fraud score) or "higher_is_safer"; NULL derives it from objective.
retire_at	Share of the events of a rule caught by the score from which the rule is a candidate for retirement.
level	Confidence level of the Jeffreys intervals.
weight	Optional column of non-negative case weights.
max_cells	Largest number of distinct score values kept exactly.

Value

An object of class c("scr_overlap", "list"):

table One row per rule (see the section Rules).

summary One row: n, events, n_score, share_score, precision_score, n_rules, share_rules, precision_rules, n_any, share_any, precision_any, recall_score, recall_rules, recall_any, incr_score, incr_rules and, with value, value_events, value_recall_score, value_recall_rules, value_recall_any, value_incr_score and value_incr_rules.

cut, alert_share, rules, retire_at, level, objective, direction, score, target, value, n, n_rows, n_dropped, n_p The cut and the settings; n_patterns is the number of distinct patterns of rule flags and score alert.

Score alerts

The score alerts the rows on its event-rich side: score >= cut under higher_is_riskier and score < cut under higher_is_safer, the convention of scr_cutoff(). Give cut, or alert_share, the share of the rows to alert: the cut is then the boundary between two distinct scores nearest to that share (tie-safe, as in scr_bands()), and the share realized is reported in the summary (share_score).

Rules

Per rule, the rows it flags are split by the score alert:

- n_rule, events_rule, precision_rule: every row the rule flags;
- n_both, events_both, precision_both: flagged by the rule and alerted by the score;
- n_rule_only, events_rule_only, precision_rule_only: flagged by the rule, not alerted by the score;
- n_score_only, events_score_only, precision_score_only: alerted by the score, not flagged by the rule;
- caught_by_score = events_both / events_rule, the share of the events of the rule that the score alerts too, and retire_candidate = caught_by_score >= retire_at (NA for a rule without events): the score already catches what the rule catches.

A precision is the event rate of the set, events over rows with a known outcome, with its Jeffreys interval (_lo, _hi; on the Kish effective size under weights). With value, value_rule, value_both, value_rule_only and value_score_only are the sums of the value over the events of each set, and caught_by_score_value the share in value. A missing rule flag counts as not flagged.

Summary

recall_score, recall_rules (any rule) and recall_any (the score or any rule) are the shares of all events alerted; incr_score = recall_any - recall_rules is what the score adds to the rules and incr_rules = recall_any - recall_score what the rules add to the score. The value_ columns are the same shares of the event value. The alert volumes are n_score, n_rules and n_any, each with its share of all rows and its precision.

Rows with a missing or infinite score, or a zero weight, are left out (n_dropped); rows with a missing outcome count in the volumes only.

Cost

The rows are counted once per distinct pattern of rule flags and score alert, and every set is a sum over that table. Time and memory after the pass grow with the number of distinct patterns times the number of rules (`n_patterns` is reported). A few dozen rules that seldom fire together give a small table; many dense, unrelated rules can give nearly one pattern per row, in which case pass the rules in smaller groups.

References

Brown, L. D., Cai, T. T. and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133. doi:10.1214/ss/1009213286

See Also

`scr_operating()` for the cut of the alerts under a capacity, `scr_detection()` for the time to detection, `scr_score_cross()` for two scores on the same rows.

Other score-studies: `scr_bands()`, `scr_claims()`, `scr_detection()`, `scr_maturity()`, `scr_mix_shift()`, `scr_operating()`, `scr_rag()`, `scr_rag_plan()`, `scr_score_cross()`, `scr_segments()`, `scr_tiers()`, `scr_uplift()`

Examples

```
local({
  set.seed(1)
  n <- 20000
  x <- rnorm(n)
  fraud <- rbinom(n, 1, plogis(-5 + 1.5 * x))
  d <- data.frame(score = round(100 * plogis(x + rnorm(n, sd = 0.5))), y = fraud,
                  amount = round(rexp(n, 1 / 80), 2),
                  # one rule the score covers, one that sees something else
                  rule_velocity = as.integer(x > 1.6),
                  rule_new_device = rbinom(n, 1, ifelse(fraud == 1, 0.3, 0.01)))
  ov <- scr_overlap(d, rules = c("rule_velocity", "rule_new_device"), alert_share = 0.05,
                    value = "amount")
  print(ov)
 ov$table[, c("rule", "n_rule", "precision_rule", "caught_by_score", "retire_candidate")]
})
```

---

scr_pd *The PD model: grades, margin of conservatism and the floor*

Description

Assembles the final grade table: `pd_be` from `scr_grades()`, the active entries of the MoC ledger by category (`A` and `B` summed over their entries, the latest `C`), `pd_moc = pd_be + A + B + C`, the PD floor of the asset class under the framework of `params`, and `pd_final = max(pd_moc, floor)`. With `philosophy = "pit"` the through-the-cycle `pd_moc` is converted with the one-factor bridge of `scr_pd_pit_ttc()` before the floor (`rho` and `z` required).

Usage

```
scr_pd(
  grades,
  moc = NULL,
  params = NULL,
  asset_class = NULL,
  philosophy = c("ttc", "pit"),
  rho = NULL,
  z = NULL
)
```

Arguments

grades	An scr_grades() object, after scr_moc().
moc	NULL uses grades$moc; otherwise a ledger in the same format.
params	An scr_irb_params(); NULL uses config$framework.
asset_class	Asset class of the floor; NULL uses config$asset_class.
philosophy	"ttc" (default) or "pit".
rho, z	Asset correlation and systematic factor for "pit".

Value

An object of class scr_pd: table (grade, label, score_lo, score_hi, n, share, defaults, dr, pd_be, moc_a, moc_b, moc_c, pd_moc, pd_ttc, pd_pit, floor, pd_final, floor_applied), breaks, band_grade, direction, alignment, alignment_score, master_scale, asset_class, framework, floor, philosophy, rho, z, moc_ledger, calibration, concentration, portfolio (weighted pd_be, pd_moc, pd_final, moc_bp, share_at_floor), scorecard, ledger, model_card. Also params_modified, repairs, dr, rows, pd_source, grade_method, sample, target, config and, after scr_export(), files.

See Also

Other irb-pd: predict.scr_grades(), predict.scr_pd(), scr_calibrate(), scr_grades(), scr_master_scale(), scr_migration(), scr_moc(), scr_pd_pit_ttc(), scr_pd_validate()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
gr <- scr_moc(scr_grades(sc, n_grades = 6, min_defaults = 10), "C", method = "ci_binomial")
pd <- scr_pd(gr)
pd
pd$table[, c("grade", "pd_be", "moc_c", "pd_final", "floor_applied")]
head(predict(pd, score = c(480, 560, 640), type = "pd_final"))
head(scr_apply(pd, head(scr_demo, 5)))
cat(tail(scr_sql(pd), 8), sep = "\n")
```

`scr_pd_pit_ttc` *One-factor bridge between point-in-time and through-the-cycle PD*

Description

Vasicek's conditional default probability:

$$PD_{PIT} = \Phi\left(\frac{\Phi^{-1}(PD_{TTC}) - \sqrt{\rho}\,z}{\sqrt{1-\rho}}\right),$$

and its inverse for `to = "ttc"`. A positive `z` is a benign state (lower PIT PD), a negative one a stressed state.

Usage

```
scr_pd_pit_ttc(pd, z, rho, to = c("pit", "ttc"))
```

Arguments

`pd`	Numeric PDs in `(0, 1)`.
`z`	Systematic factor (standard normal scale).
`rho`	Asset correlation in `(0, 1)`.
`to`	`"pit"` (input is TTC) or `"ttc"` (input is PIT).

Value

A numeric vector of the length of `pd`.

References

Vasicek, O. (2002). The distribution of loan portfolio value. *Risk*, 15(12), 160-162.

See Also

`scr_pd_stress()`, the same bridge with the systematic factor given as a quantile `q` rather than a value of `z`.

Other irb-pd: `predict.scr_grades()`, `predict.scr_pd()`, `scr_calibrate()`, `scr_grades()`, `scr_master_scale()`, `scr_migration()`, `scr_moc()`, `scr_pd()`, `scr_pd_validate()`

Examples

```
scr_pd_pit_ttc(c(0.01, 0.05), z = -2, rho = 0.15)
scr_pd_pit_ttc(scr_pd_pit_ttc(0.02, z = -1, rho = 0.1), z = -1, rho = 0.1, to = "ttc")
```

`scr_pd_stress` *Stressed PD of the one-factor model*

Description

The conditional PD at confidence `q`: `N((G(pd) + sqrt(rho) G(q)) / sqrt(1 - rho))`. With `q = 0.999` and the regulatory correlation it is the PD inside the risk-weight function, so that the capital requirement of a retail exposure is `lgd * (scr_pd_stress(pd, r, 0.999) - pd)`. Used by the sensitivity grid of `scr_capital()` and by the scenario engine of `scr_ecl()`. Arguments are recycled.

Usage

```
scr_pd_stress(pd, rho, q)
```

Arguments

`pd`	Numeric vector of unconditional PDs.
`rho`	Asset correlation in `[0, 1)`.
`q`	Confidence level in `(0, 1)`; `0.5` returns the median-year PD.

Value

A numeric vector of conditional PDs.

References

Vasicek, O. (2002). The distribution of loan portfolio value. *Risk*, 15(12), 160-162. Gordy, M. B. (2003). A risk-factor model foundation for ratings-based bank capital rules. *Journal of Financial Intermediation*, 12(3), 199-232.

See Also

`scr_pd_pit_ttc()`, the same bridge with the systematic factor given as a value of `z` rather than a quantile `q`.

Other irb-capital: `scr_capital()`, `scr_ecl()`, `scr_el()`, `scr_irb_rw()`, `scr_sa_rw()`

Examples

```
scr_pd_stress(0.02, rho = 0.15, q = c(0.5, 0.95, 0.99, 0.999))
```

scr_pd_validate *Validate a PD model on a cohort panel*

Description

Runs the standard battery on a monthly panel with the default flag and the grade (or the score) at every month: obligors non-defaulted at each cohort start form the population, the outcome is a default within horizon months, exactly as scr_default_rate() does.

Usage

```
scr_pd_validate(
  x,
  newdata,
  id = "id",
  date = "date",
  default = "default",
  grade = NULL,
  score = NULL,
  auc_init = NULL,
  cv_init = NULL,
  tests = c("jeffreys", "binomial", "normal", "hl", "multi_period", "auc",
    "concentration", "psi", "migration"),
  alpha = 0.05,
  lights = NULL,
  pd_column = c("pd_final", "pd_moc", "pd_be"),
  horizon = 12L,
  by = NULL,
  n_boot = NULL,
  seed = NULL
)
```

Arguments

x	An scr_pd() object.
newdata	A data.frame/data.table panel, one row per id and month.
id, date, default	Column names.
grade	Column name of the grade at every month; NULL derives it from score with the cut points of x.
score	Column name of the production score at every month, optional.
auc_init	Development AUC; NULL uses the scorecard's hold-out AUC.
cv_init	Development coefficient of variation; NULL uses the one of x.
tests	Subset of the battery to run.
alpha	Significance level of the binomial critical count.

lights	Two p-value thresholds (red at or below the first, amber at or below the second, green above; the convention shared with the LGD and EAD validations); NULL reads config$pd_lights. A missing p-value gives "grey".
pd_column	Grade PD tested: "pd_final" (default), "pd_moc" or "pd_be".
horizon, by	Cohort window in months and frequency (NULL reads config$pd_dr_by).
n_boot, seed	Bootstrap resamples and seed of the discrimination interval.

Details

Calibration Per grade (pooled over cohorts) and per cohort and grade: Jeffreys p = F_Beta(PD; D + 1/2, N - D + 1/2), the binomial P(X >= D) with its critical count at alpha, the normal z, and the traffic light on the Jeffreys p-value. Portfolio: the same tests on the totals, Hosmer-Lemeshow over the grades (K degrees of freedom: the grade PDs are not fitted on the validation sample), the multi-period normal test over the cohort differences DR_t - PD_t (BCBS Working Paper 14, 2005) and the Brier score.

Discrimination AUC, Gini and KS with a bootstrap interval (scr_metrics()) on the score when a score column exists, otherwise on the grade; the S statistic against auc_init ((AUC_init - AUC_curr) / se, with the DeLong standard error of the current AUC), p = 1 - Phi(S).

Stability PSI of the grade distribution against the development sample per cohort (scr_psi()); the migration matrix pooled over the cohorts whose end date is observed (scr_migration()); the concentration test on the coefficient of variation of the latest cohort against cv_init.

Value

An object of class scr_pd_validation: calibration (per grade, pooled), calibration_cohort (per cohort and grade), portfolio (per cohort), portfolio_tests (list: n, d, dr, pd, p_jeffreys, p_binomial, hl_chi2, hl_df, hl_p, multi_period_z, multi_period_p, brier), discrimination, stability (psi table, migration, concentration), summary (one row per test with statistic, p_value, light; the light is "grey" when the row has no testable result, such as a missing p-value or the descriptive migration bandwidth), light (the worst light of the summary: red, then amber, then green; "grey" when no row has a testable result), n_cohorts, alpha, lights. portfolio_tests also carries critical, z, p_normal, n_cohorts and pd_column; the object also has horizon, by, pd_column and target.

See Also

Other irb-pd: predict.scr_grades(), predict.scr_pd(), scr_calibrate(), scr_grades(), scr_master_scale(), scr_migration(), scr_moc(), scr_pd(), scr_pd_pit_ttc()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
pd <- scr_pd(scr_moc(scr_grades(sc, n_grades = 6, min_defaults = 10), "C", method = "ci_binomial"))
# the validation panel: default flag at every month plus the grade at the
# cohort start; here the behavioural score of the panel is graded with the
```

```
# cut points of the PD model
d <- scr_default(scr_demo_panel, "id", "ref_date", dpd = "dpd", config = cfg)
pnl <- merge(d$flags, scr_demo_panel[, c("id", "ref_date", "score")],
             by.x = c("id", "date"), by.y = c("id", "ref_date"))
pnl$grade <- predict(pd, score = pnl$score, type = "grade")
v <- scr_pd_validate(pd, pnl, id = "id", date = "date", default = "default",
                     grade = "grade", score = "score", by = "quarter")
v
v$summary
```

`scr_presets` *Selection presets, side by side*

Description

Returns the funnel keys resolved per preset, to compare before choosing. `target_min` and `iv_max` are shown for context; the presets leave them unchanged, as they do every other configuration key.

Usage

```
scr_presets()
```

Value

A `data.frame` with one row per preset.

See Also

Other configuration: `scr_config()`, `scr_config_keys()`, `scr_verbose()`

Examples

```
scr_presets()
```

`scr_psi` *Population stability index, with the fixed and the sample-size-adjusted threshold*

Description

`PSI = sum((p - q) * ln(p / q))` over bins frozen on the base. Reports both thresholds side by side: the traditional fixed one (`< 0.10 "stable"`, `0.10-0.25 "moderate"`, `>= 0.25 "shift"`) and the sample-size-adjusted critical value of Yurdakul and Naranjo (2020), under which the PSI is asymptotically `(1/n + 1/m) * chi-squared(B - 1)`. With `n = m = 1000` and ten bins the 5% critical value is 0.034, not 0.10; on a monthly base of a hundred thousand rows, `PSI = 0.01` is already significant. The fixed threshold remains what the market knows; the adjusted one is what the statistics support.

Usage

```
scr_psi(
  base,
  compare,
  levels = NULL,
  breaks = NULL,
  n_groups = 10L,
  alpha = 0.05,
  thresholds = c(0.1, 0.25)
)
```

Arguments

base	Reference vector (the "development" distribution).
compare	Vector to compare.
levels	For categorical vectors: the levels to consider. NULL uses the union of the observed ones.
breaks	For numeric vectors: frozen cut points. NULL derives n_groups quantiles of base.
n_groups	Number of bands when breaks = NULL.
alpha	Significance level of the adjusted threshold.
thresholds	The two fixed thresholds: below the first the flag is "stable", below the second "moderate", otherwise "shift".

Details

Rows where base or compare is NA, or that fall outside breaks or levels, are not counted. A band empty in both samples is left out of the index and of the degrees of freedom B - 1; when a populated band is empty on one side only, 0.5 is added to every populated band of both samples.

Value

A list of class scr_psi with psi, flag_fixed, critical (adjusted critical value), flag_adjusted ("stable" or "shift"), n_base, n_compare, n_bins (bands declared; the degrees of freedom count only the populated ones) and table (per band: n_base, n_compare, pct_base, pct_compare, psi_band). The thresholds and alpha used are stored and printed.

References

Yurdakul, B. and Naranjo, J. (2020). Statistical properties of the population stability index. *Journal of Risk Model Validation*, 14(4), 89-100.

See Also

Other metrics: scr_iv(), scr_metrics()

Examples

```
set.seed(2)
base <- stats::rnorm(5000)
new  <- stats::rnorm(5000, mean = 0.15)
p <- scr_psi(base, new)
p
p$table
```

---

scr_rag *Red / amber / green lights of a score against its reference*

Description

Reads a study sample against the reference the score was developed on and lights every check red, amber or green, in four families: discrimination, calibration, stability and, for a scorecard, the variables. With by, one set of lights per period or segment, from the same count table.

Usage

```
scr_rag(x, ...)

## S3 method for class 'scr_scorecard'
scr_rag(
  x,
  plan = NULL,
  sample = "holdout",
  reference = "train",
  by = NULL,
  level = NULL,
  min_events = 20L,
  n_bands = NULL,
  n_boot = NULL,
  seed = NULL,
  max_cells = 1e+05,
  boot_cells = 10000,
  ...
)

## S3 method for class 'data.frame'
scr_rag(
  x,
  score = "score",
  y = "y",
  prob = NULL,
  objective = "risk",
  direction = NULL,
  weight = NULL,
```

```
  sample = NULL,
  reference = NULL,
  study = NULL,
  by = NULL,
  plan = NULL,
  level = 0.95,
  min_events = 20L,
  n_bands = 10L,
  n_boot = 200L,
  seed = NULL,
  max_cells = 1e+05,
  boot_cells = 10000,
  ...
)

## S3 method for class 'scr_study'
scr_rag(
  x,
  plan = NULL,
  sample = NULL,
  level = NULL,
  min_events = 20L,
  n_boot = 200L,
  seed = NULL,
  boot_cells = 10000,
  ...
)
```

Arguments

x	An object from scr_scorecard(), scr_bands() or scr_tiers(), or a data.frame with one row per scored case.
...	Passed on to the methods; an unknown argument is an error.
plan	Thresholds table, as returned by scr_rag_plan(); NULL uses the defaults for the objective.
sample	For a scorecard: the study sample ("holdout"). For a data.frame: the name of a column with sample labels, or NULL. For a score study: the study samples, NULL for every sample but the reference.
reference	For a scorecard: the sample the bands are frozen on ("train"). For a data.frame: the label of the reference sample; NULL takes the first level of the sample column (the levels of a factor in their order, numbers in numeric order, text sorted).
by	Name of a column holding periods or segments (for a scorecard, a column of its scored samples such as "date").
level	Confidence level of the intervals. For a scorecard or a score study, NULL uses config$study_level or the level of the study.
min_events	Fewest events, and non-events, of the study group for a lit result.

n_bands	Bands frozen on the reference for the PSI, the rank order and the band calibration. For a scorecard, NULL uses config$score_groups. A score study uses its own cuts.
n_boot	Bootstrap resamples of the Gini ratio interval. For a scorecard, NULL uses config$n_boot.
seed	Seed of the bootstrap. A number is local to the call (the user's random stream is restored on exit); NULL draws from the user's stream and advances it. For a scorecard, NULL uses config$seed.
max_cells	Largest number of distinct score values kept exactly.
boot_cells	Largest number of score cells resampled exactly by the bootstrap (default 10,000; Inf for no pooling). See the section Method.
score, y	Column names of the score and of the 0/1 outcome (NA allowed).
prob	For a data.frame: optional column with the expected event probability of every case.
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).
direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
weight	Optional column of non-negative case weights.
study	Labels of the study samples; NULL takes every level other than the reference.

Value

An object of class scr_rag:

table One row per check: sample, group, family, metric, level ("score", a variable name or a band label), value, lo, hi, benchmark (the reference value or critical value it is read against), light ("green", "amber", "red", "grey" or "none") and reason.

summary One row per sample and group: the light of discrimination, calibration, stability, variables and overall, and the reason of the overall light.

plan The thresholds used.

objective, direction, level, min_events, reference, study, by, cuts, target, call The settings.

Checks

Discrimination gini_ratio, Gini(study) / Gini(reference), with a bootstrap interval from independent resamples of both samples (the count bootstrap of scr_bands(), exact up to boot_cells distinct scores and an approximation above); auc_change_p, the one-sided p-value of the S-test

$$S = (AUC_{ref} - AUC_{study}) / \sqrt{se_{study}^2 + se_{ref}^2},$$

with both DeLong standard errors computed from the counts per score value. This is the two-sample form: both AUCs are estimated here, on independent samples. The ECB (2019) instructions take the initial AUC as fixed (only the current AUC's standard error), which suits a development AUC taken from documentation; applied to an estimated reference AUC, that form rejects too often. ks is reported without a light.

Calibration Needs an expected probability: the alignment of a scorecard, or the `prob` column of a data.frame; otherwise both lights are `"grey"` ("no expected probability"). `oe_ratio`, observed over expected events, with the Jeffreys interval of the observed rate; `band_calibration`, the number of bands whose Jeffreys test rejects the band mean expected probability. Under risk both checks are one-sided, against under-prediction only (over-prediction is prudent); under propensity they are two-sided (see `scr_rag_plan()`). Every band is also listed, without a light.

Stability `score_psi` over bands frozen on the reference, lit by effect and significance; `rank_order`, the number of Holm-significant reversals between adjacent bands.

Variables For a scorecard read on its hold-out against train: the CSI of every variable over its frozen bins (same rule as the PSI), `iv_ratio` = IV(study) / IV(reference), and `woe_sign_flip`, the number of bins holding at least 5% of the study volume whose WOE changes sign (amber at most).

One convention holds for every light: a light is amber or red only when the confidence interval shows the metric beyond the threshold (for a p-value, when the test rejects); with too few events it is `"grey"`. The thresholds and rules are in `scr_rag_plan()`.

A group with fewer than `min_events` events, or non-events, in the study sample gets `"grey"` lights, with the values still reported. Within a family the worst light wins (red, then amber, then green; `"grey"` only when nothing is lit). The overall light is the worst of discrimination and calibration; stability and variables can raise it to amber, never to red, and an overall without any lit discrimination or calibration check is `"grey"`. The `reason` of the summary says how the overall light was formed, for example "calibration not tested" when no expected probability is available.

With `by`, a group of the study sample is compared with the same group of the reference when the reference has it (a segment), and with the whole reference otherwise (a new period). Without a sample column, every group is compared with the whole data. The groups are listed in the order of their labels, those of a numeric column in numeric order.

References

Brown, L. D., Cai, T. T. and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133. doi:10.1214/ss/1009213286

DeLong, E. R., DeLong, D. M. and Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, 44(3), 837-845.

European Central Bank (2019). *Instructions for reporting the validation results of internal models: IRB Pillar I models for credit risk*. ECB Banking Supervision.

Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.

Yurdakul, B. and Naranjo, J. (2020). Statistical properties of the population stability index. *Journal of Risk Model Validation*, 14(4), 89-100.

See Also

Other score-studies: `scr_bands()`, `scr_claims()`, `scr_detection()`, `scr_maturity()`, `scr_mix_shift()`, `scr_operating()`, `scr_overlap()`, `scr_rag_plan()`, `scr_score_cross()`, `scr_segments()`, `scr_tiers()`, `scr_uplift()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
rg <- scr_rag(sc)
rg
rg$summary

# a data.frame with a sample column and an expected probability
d <- rbind(data.frame(sample = "dev", sc$samples$train[, c("score", "y", "prob")]),
           data.frame(sample = "new", sc$samples$holdout[, c("score", "y", "prob")]))
scr_rag(d, prob = "prob", sample = "sample", n_boot = 50)$summary
```

---

scr_rag_plan *Thresholds of the red / amber / green lights*

Description

The editable table read by `scr_rag()`: one row per metric with its thresholds and the rule that turns a value into a light. Edit a value, or drop a row to leave a metric out, and pass the table as `plan`.

Usage

```
scr_rag_plan(objective = c("risk", "propensity"))
```

Arguments

`objective`	`"risk"` or `"propensity"`: the Gini ratio thresholds are looser under propensity (0.90 / 0.80 against 0.95 / 0.90), and the calibration checks are one-sided under risk, two-sided under propensity (see the section Calibration under risk and propensity).

Value

A `data.frame` with `family`, `metric`, `green`, `red`, `green_hi`, `red_hi` (upper edges of the two-sided rule), `higher_better`, `rule` and `note`.

Rules

Every rule follows one convention: a light is amber or red only when the confidence interval shows the metric beyond the threshold (for a p-value, when the test rejects); with too few events it is `"grey"`.

`ci` With `higher_better`: green when the upper bound of the interval reaches `green`, red when it stays below `red`, amber in between; the mirror image on the lower bound when lower is better.

`threshold` The same on the value alone.

`p_value` Red at or below `red`, amber at or below `green`, green above.

`interval` Two-sided: green when the interval meets `[green, green_hi]`, red when it lies entirely outside `[red, red_hi]`, amber otherwise.

`psi` Effect and significance: red when the index reaches `red` **and** exceeds the n-adjusted critical value (Yurdakul and Naranjo, 2020), amber when it reaches `green` and exceeds it, green otherwise. Significance alone never colors a light, which on a large sample would flag every negligible shift.

`count` Green at or below `green`, red at or above `red` (never when `red` is `NA`), amber in between.

`none` Reported without a light.

Every threshold is a convention of this package, documented in `note`, except where a source is cited there; adjust them to the validation policy in force.

Calibration under risk and propensity

Under `objective = "risk"` every calibration check is one-sided: over-prediction (more expected than observed events) is prudent and only under-prediction is penalized. `oe_ratio` then follows the rule `ci` with lower better, lit on the lower bound of its interval: green when it is at or below 1.10, red above 1.25, amber in between; a conservative model (O/E well below 1) stays green. `band_calibration` tests every band against under-prediction only. Under `"propensity"` both directions count: `oe_ratio` follows the two-sided rule `interval` (green when the interval meets `[0.90, 1.10]`, red when it lies outside `[0.80, 1.25]`) and the band tests are two-sided.

References

European Central Bank (2019). *Instructions for reporting the validation results of internal models: IRB Pillar I models for credit risk*. ECB Banking Supervision.

Yurdakul, B. and Naranjo, J. (2020). Statistical properties of the population stability index. *Journal of Risk Model Validation*, 14(4), 89-100.

See Also

Other score-studies: `scr_bands()`, `scr_claims()`, `scr_detection()`, `scr_maturity()`, `scr_mix_shift()`, `scr_operating()`, `scr_overlap()`, `scr_rag()`, `scr_score_cross()`, `scr_segments()`, `scr_tiers()`, `scr_uplift()`

Examples

```
plan <- scr_rag_plan()
plan[, c("family", "metric", "green", "red", "rule")]
# a stricter policy on the score PSI
plan$green[plan$metric == "score_psi"] <- 0.05
```

scr_reasons *Reason codes: the variables that took the most points from each row*

Description

For each row of newdata, the k variables whose contribution in points fell furthest below the reference. The reference is the mean points of the variable on the training population ("mean", the Regulation B safe harbor referenced to the average) or the maximum points of the variable ("max"). Only applies to the additive scorecard; a tree challenger has no reason codes.

Usage

```
scr_reasons(x, newdata, k = 4L, reference = c("mean", "max"))
```

Arguments

x	An object from scr_scorecard().
newdata	New table.
k	Number of reasons per row.
reference	"mean" (default) or "max".

Details

Under higher_is_riskier the shortfall is measured the other way round: the reasons are the variables that added the most points.

Value

A data.table with reason_1 ... reason_k (variable names) and shortfall_1 ... shortfall_k (points below the reference).

References

12 CFR 1002.9 (Regulation B), official commentary to paragraph 9(b)(2).

See Also

Other production: predict.scr_align(), scr_apply(), scr_export(), scr_monitor(), scr_monitoring_plan(), scr_sql()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_reasons(sc, head(scr_demo, 5), k = 3)
```

scr_reject *Stage 6: honest reject inference through a sensitivity band*

Description

Does not ship parceling as the default behavior: instead of inventing a single multiplier and reweighting, it declares the **population scope** of the scorecard, measures the **coverage per band** (where an observed outcome exists, and in what volume) and presents a **sensitivity band**: the event rate each band would have if the population without an outcome were 2, 4 or 8 times worse than the observed one, with the effect on the total. The analyst reads the band; no single number is fabricated.

Usage

```
scr_reject(
  x,
  population = NULL,
  accepted = NULL,
  multipliers = NULL,
  sample = "holdout"
)
```

Arguments

x	An object from scr_scorecard().
population	Optional: a table of the full population (accepted and rejected, without outcome), scored by scr_apply(). NULL restricts the scope to the population with an outcome.
accepted	Optional: a logical vector, of the length of population, marking the rows with an observed outcome. NULL treats the whole population as without an outcome beyond the development sample.
multipliers	Sensitivity band. NULL uses the configuration.
sample	Reference sample of the observed outcomes.

Value

An scr_reject object with scope, coverage (per band) and sensitivity (per band and multiplier, plus the TOTAL row).

See Also

Other stages: scr_align(), scr_bin(), scr_cutoff(), scr_model(), scr_scorecard(), scr_select(), scr_split(), scr_strategy(), scr_triage()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_reject(sc)
# with a through-the-door population: rows with an outcome are the hold-out
acc <- seq_len(nrow(scr_demo)) %in% res$split$holdout_idx
scr_reject(sc, population = scr_demo, accepted = acc)
```

---

scr_result *Result of a selection*

Description

Object returned by `scr_select()`. The methods below are the supported way of inspecting the result in the console; to extract data, use the accessors (`scr_selected()`, `scr_funnel()`, `scr_gains()`).

Usage

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

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

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

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

Arguments

`x`, `object`	An `scr_result` object.
`...`	Ignored, present for compatibility with the generic.

Value

`print()` and `plot()` return `x` invisibly; `summary()` returns an `scr_summary` object; `as.data.frame()` returns the funnel.

See Also

Other accessors: `scr_funnel()`, `scr_gains()`, `scr_leakage()`, `scr_score_gains()`, `scr_score_metrics()`, `scr_selected()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
res                     # print: the funnel in one screen
summary(res)            # full text report
head(as.data.frame(res)) # the funnel as a data.frame
plot(res)
```

`scr_run` *Run the selection for several targets straight from the database*

Description

For each target: fetches the table, runs `scr_select()` and writes the deliverables. A failure on one target is recorded and the loop continues.

Usage

```
scr_run(
  con,
  table,
  targets,
  config = scr_config(),
  drop = character(),
  date_col = config$oot_date_col,
  event_level = NULL,
  sample_frac = 1,
  max_rows = NULL,
  export = NULL
)
```

Arguments

`con`	A DBI connection, from `scr_connect()`.
`table`	Table name, with an optional `{target}`.
`targets`	Vector with the names of the target columns.
`config`	An object from `scr_config()`, used for every target.
`drop`	Columns that are never candidates.
`date_col`	Date column of the out-of-time split, passed on to `scr_select()`. Defaults to `config$oot_date_col`; an explicit `NULL` forces a random stratified split.
`event_level`	Passed on to `scr_select()`.
`sample_frac`	Sampling fraction. A scalar or a list named by target.
`max_rows`	Row cap per target. `NULL` switches it off.
`export`	Root output directory; each target writes to a subdirectory.

Value

An `scr_runset` object: a named list of `scr_result` (or, for the targets that failed, a list with `error`).

Table convention

`table` accepts the `{target}` placeholder, replaced by the lower-case target name. Without the placeholder, the same table is used for every target.

See Also

`scr_compare()` and `scr_core()` to read the run set.

Other portfolio: `scr_compare()`, `scr_core()`, `scr_runset`

Examples

```
con <- scr_connect(driver = RSQLite::SQLite(), dbname = ":memory:")
d <- scr_demo; d$ref_date <- as.character(d$ref_date)   # SQLite has no Date type
DBI::dbWriteTable(con, "dtm", d)
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
rs <- scr_run(con, "dtm", targets = c("default", "churn"), config = cfg,
              drop = c("id", "ref_date", "default", "churn"))
rs
scr_compare(rs)
DBI::dbDisconnect(con)
```

---

scr_runset *Set of runs, one per target*

Description

Object returned by `scr_run()`: a named list of `scr_result`, plus the errors of the targets that failed. Use `scr_compare()` for the comparison table and `scr_core()` for the variables that cross several targets.

Usage

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

Arguments

`x`	An `scr_runset` object.
`...`	Ignored.

Value

x, invisibly.

See Also

Other portfolio: scr_compare(), scr_core(), scr_run()

Examples

```
con <- scr_connect(driver = RSQLite::SQLite(), dbname = ":memory:")
d <- scr_demo[, c("default", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_hist_01")]
DBI::dbWriteTable(con, "dtm", d)
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
rs <- scr_run(con, "dtm", targets = "default", config = cfg)
rs
names(rs)
DBI::dbDisconnect(con)
```

---

scr_sa_rw *Standardized risk weight of an exposure*

Description

Lookup in params$sa_rw: regulatory retail ("retail_other", "qrre_*": 75 %, or the transactor weight), residential mortgages by loan-to-value band (a missing LTV takes the highest band), corporates by external rating bucket ("AAA" to "AA-", "A", "BBB", "BB", below; NA is unrated; "IG" marks an unrated investment-grade obligor where ratings are not used) or the SME weight, banks and sovereigns through the corporate rating rows, and defaulted exposures by the specific provision ratio (or the mortgage row). Arguments are recycled.

Usage

```
scr_sa_rw(
  asset_class,
  ltv = NULL,
  rating = NULL,
  transactor = NULL,
  defaulted = NULL,
  provision_ratio = NULL,
  sme = NULL,
  granular = TRUE,
  params = scr_irb_params("bcb")
)
```

Arguments

asset_class	One of "corporate", "corporate_sme", "bank", "sovereign", "hvcre", "retail_mortgage", "qrre_revolver", "qrre_transactor", "retail_other"; a scalar or a vector.
ltv	Loan-to-value at origination, decimal (mortgages).
rating	External rating string (corporates), NA when unrated.
transactor	Logical: revolving facility repaid in full every month.
defaulted	Optional 0/1 or logical vector.
provision_ratio	Specific provisions over the outstanding amount (defaulted rows).
sme	Logical: corporate small or medium enterprise (also implied by asset_class = "corporate_sme").
granular	Logical (scalar or per exposure): whether the retail exposure belongs to a granular regulatory retail pool; FALSE applies the non-granular retail weight.
params	An scr_irb_params() object.

Value

A numeric vector of standardized risk weights (decimals).

References

Basel Committee on Banking Supervision (2023). *The Basel Framework*, CRE20 (standardised approach: individual exposures).

See Also

Other irb-capital: scr_capital(), scr_ecl(), scr_el(), scr_irb_rw(), scr_pd_stress()

Examples

```
scr_sa_rw(c("retail_other", "retail_mortgage", "corporate"), ltv = c(NA, 0.55, NA),
          rating = c(NA, NA, "A+"))
scr_sa_rw("retail_other", defaulted = TRUE, provision_ratio = c(0.1, 0.3))
```

---

scr_score_cross *Two scores on the same rows*

Description

Crosses two scores read on the same rows (a credit score and a churn score, a champion and a challenger): the cross table of their bands with the event rate of one or two outcomes, the rank association of the scores, and the overlap of the rows each one selects at a few depths, with the swap-in and swap-out sets.

Usage

```
scr_score_cross(x, ...)

## S3 method for class 'data.frame'
scr_score_cross(
  x,
  score_a,
  score_b,
  y = NULL,
  y_a = NULL,
  y_b = NULL,
  objective_a = "risk",
  objective_b = NULL,
  direction_a = NULL,
  direction_b = NULL,
  n_bands = 5L,
  cuts_a = NULL,
  cuts_b = NULL,
  depths = c(0.05, 0.1, 0.2),
  weight = NULL,
  level = 0.95,
  ...
)
```

Arguments

x	A `data.frame` with both scores on every row.
...	Not used; an unknown argument is an error.
score_a, score_b	Column names of the two scores.
y	Column name of a 0/1 outcome read under both scores.
y_a, y_b	Instead of `y`: the outcome of score A and of score B (two different targets on the same rows); either may be given alone.
objective_a, objective_b	`"risk"` or `"propensity"`; `objective_b = NULL` takes `objective_a`. A study given as cuts sets the objective of its score.
direction_a, direction_b	`"higher_is_safer"` or `"higher_is_riskier"`; `NULL` derives each from its objective, or takes it from the study given as cuts.
n_bands	Bands of each score when its cuts are not given.
cuts_a, cuts_b	Optional cuts of each score: a numeric vector, or an object from `scr_bands()` or `scr_tiers()`, which also sets the objective and the direction of that score.
depths	Shares of the rows selected by each score for the overlap, in (0, 1].
weight	Optional column of non-negative case weights.
level	Confidence level of the Jeffreys intervals.

Value

An object of class `c("scr_score_cross", "list")`:

`table` The cross table: `band_a`, `label_a`, `band_b`, `label_b`, `n`, `pct` and the outcome columns (see the section Cross table).

`overlap` One row per depth: `depth`, `cut_a`, `cut_b`, `share_a`, `share_b`, `n_a`, `n_b`, `n_both`, `n_a_only`, `n_b_only` and `jaccard`.

`overlap_rates` With an outcome, one row per depth, outcome and set (`"A"`, `"B"`, `"both"`, `"A only"`, `"B only"`): `n`, `events`, `rate`, `rate_lo` and `rate_hi`.

`association` One row per method (`"spearman"`, `"kendall_tau_b"`): `estimate`, `oriented` and `n`.

`settings` A list: the score and outcome columns, objectives, directions, cuts, codes and labels of both scores, `depths`, `level`, `n` (the volume used: the sum of the weights, the number of rows without weights), `n_rows` (the rows used), `n_dropped` (rows left out) and `weighted`.

Bands

Each score is cut into `n_bands` bands of equal share, tie-safe as in `scr_bands()` (band 1 is the event-richest), unless `cuts_a` or `cuts_b` gives the cuts: a numeric vector, or an object from `scr_bands()` or `scr_tiers()`, whose cuts, numbers and labels are then used (the tier labels for tiers). Bands are left-closed, `score >= cut` being the upper side.

A study given as cuts also sets the objective and the direction of its score, so the event-rich end of the overlap is the one the study was fitted with. `objective_a`, `direction_a` (or their `_b` counterparts) need not be given then; when given and different from the study, the call is an error.

Cross table

One row per pair of bands, every pair listed (empty ones with `n = 0`), then the totals of each band of A (`band_b` missing, `label_b = "total"`), of each band of B, and the grand total. Per row: `n`, `pct` (share of all rows) and, for every outcome, `events`, `rate` with its Jeffreys interval `rate_lo`, `rate_hi` (on the Kish effective size under weights) and `lift` (the rate over the overall rate of that outcome). With `y` the outcome columns have no suffix; with `y_a` and `y_b` they end in `_a` and `_b`.

Association

Spearman's rank correlation (Pearson on mid-ranks, as `cor(method = "spearman")`) and Kendall's tau-b (pairs tied on either score count neither way, as `cor(method = "kendall")`), both on the raw scores and unweighted. The tau-b counts are exact, by Knight's (1966) algorithm in `O(n log n)`. `oriented` multiplies each estimate by the signs of the two directions, so it is positive when the two scores put the same rows at their event-rich ends.

Overlap

At each depth, each score selects the share `depth` of the rows from its event-rich end (tie-safe: the selection stops at the boundary between two distinct scores nearest to the target, so the share selected can differ from `depth` by the share of one score value; `share_a` and `share_b` report it). `overlap` counts the rows selected by A, by B, by both, by A only and by B only, and the Jaccard index `n_both / (n_a + n_b - n_both)`. With an outcome, `overlap_rates` gives the event rate of

each of the five sets with its Jeffreys interval: when B replaces A at the same depth, `"B only"` is the swap-in and `"A only"` the swap-out.

Rows with a missing or infinite value of either score, or a zero weight, are left out (`n_dropped`); rows with a missing outcome count in the volume but not in the rates of that outcome.

References

Brown, L. D., Cai, T. T. and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133. doi:10.1214/ss/1009213286

Knight, W. R. (1966). A computer method for calculating Kendall's tau with ungrouped data. *Journal of the American Statistical Association*, 61(314), 436-439. doi:10.1080/01621459.1966.10480879

See Also

`scr_bands()` and `scr_tiers()` for the cuts of each score.

Other score-studies: `scr_bands()`, `scr_claims()`, `scr_detection()`, `scr_maturity()`, `scr_mix_shift()`, `scr_operating()`, `scr_overlap()`, `scr_rag()`, `scr_rag_plan()`, `scr_segments()`, `scr_tiers()`, `scr_uplift()`

Examples

```
set.seed(1)
n <- 4000
z <- rnorm(n)
d <- data.frame(credit = round(600 + 40 * (-z + rnorm(n, sd = 0.6))),
                churn = round(450 + 30 * (0.4 * z + rnorm(n))),
                default = rbinom(n, 1, plogis(-2 + z)),
                left = rbinom(n, 1, 0.3))
cx <- scr_score_cross(d, "credit", "churn", y_a = "default", y_b = "left",
                      objective_b = "propensity", n_bands = 4)
cx
cx$association
cx$overlap_rates[cx$overlap_rates$depth == 0.1, ]
```

---

scr_score_gains *Score gains per frozen band*

Description

How the score behaves in each band: count, event rate, the event and non-event distributions, KS, lift, cumulative capture, odds and the score interval of the band, which is what lets a cut-off be read straight from the table. The bands are the deciles of the score on **train**, applied frozen to the other samples.

Usage

```
scr_score_gains(x, sample = NULL)
```

Arguments

x	An object from scr_scorecard().
sample	NULL (all), "train" or "holdout".

Details

woe is log(pct_event / pct_nonevent), event-oriented like the WOE of the variables (positive when the band event rate is above the overall rate) and equal to log_odds in scr_strategy() for the same sample and bands; when a band has no events or no non-events, 0.5 is added to the counts of every band for woe only. odds follows the odds orientation of the scale: non-events per event under higher_is_safer, events per non-event under higher_is_riskier, with 0.5 added to each count. log_odds therefore rises with the score under both directions, and its slope against mean_score can be read against log(2) / pdo.

Value

A data.table with one row per sample and band, the band richest in events first (the riskiest under objective = "risk"): sample, id, band, n, pct, events, non_events, event_rate, pct_event and pct_nonevent (the band's share of all events and of all non-events), woe, min_score, mean_score, max_score, cum_pct, cum_event_pct, cum_nonevent_pct, ks, lift, cum_lift, odds and log_odds.

See Also

Other accessors: scr_funnel(), scr_gains(), scr_leakage(), scr_result, scr_score_metrics(), scr_selected()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_score_gains(sc, "holdout")[, .(band, n, event_rate, min_score, max_score, ks)]
scr_score_metrics(sc)
```

---

scr_score_metrics *Score metrics per sample, with CI*

Description

n, events, AUC, KS and Gini of the scorecard score on train and hold-out, with a bootstrap confidence interval and the direction used: the AUC is always reported above 0.5 when the score ranks correctly in its own direction.

Usage

```
scr_score_metrics(x)
```

Arguments

x	An object from scr_scorecard().

Value

A data.table with one row per sample.

See Also

Other accessors: scr_funnel(), scr_gains(), scr_leakage(), scr_result, scr_score_gains(), scr_selected()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_score_metrics(sc)
```

---

scr_scorecard *Stages 4 and 5: points scorecard, aligned to the declared scale*

Description

Fits a logistic regression on the WOE columns of the shortlist, checks the sign of the coefficients, aligns the logit to the declared scale with scr_align() (always) and distributes the points per bin. Measures the score on train and hold-out with a bootstrap CI (always), builds the gains with bands **frozen on train**, the score PSI and the CSI per variable (fixed and n-adjusted thresholds), the calibration and the rank-order diagnostics (a one-sided Fisher exact test of each band against the previous, riskier one). Optionally fits a tree challenger on the same WOE columns, aligned to the same scale, with an explicit supports_scorecard = FALSE: it compares, it never produces points or reason codes.

Usage

```
scr_scorecard(
  x,
  features = NULL,
  base_score = NULL,
  base_odds = NULL,
  pdo = NULL,
  direction = NULL,
  align_method = NULL,
  challenger = NULL,
  points_style = NULL,
```

```
  n_boot = NULL,
  seed = NULL
)
```

Arguments

x	An object from scr_select().
features	Variables of the scorecard. Defaults to scr_selected().
base_score, base_odds, pdo, direction	The scale; NULL uses the configuration of x. See scr_config() and scr_align().
align_method	"regression" or "direct"; NULL uses the configuration.
challenger	NULL, "xgboost" or "lightgbm"; NULL uses the configuration.
points_style	"base_plus_deviation" or "distributed"; NULL uses the configuration.
n_boot	CI resamples; NULL uses the configuration.
seed	Seed; NULL uses the configuration.

Value

An scr_scorecard object. Main components: features, coef, sign_check, alignment (an scr_align object), points, base_points, samples (train and hold-out: link, prob, score, score_points, y, date), metrics, gains, stability (score and variables), calibration, rank_order, challenger, model_card and sql. Also scale (base_score, base_odds, pdo, factor, offset, direction, odds_orientation), breaks (the score bands frozen on train), monitoring_plan (see scr_monitoring_plan()), holdout_bins, fit and ledger (the frozen binning and pre-processing that scr_apply() and scr_sql() reproduce) and, after a lab commit, decisions and provenance.

Sign check

The engine's WOE is event-oriented, so every glm coefficient must be positive. A variable with a non-positive coefficient (or above max_abs_coef in absolute value) is explaining what another already explained, with the sign reversed; it is removed and the model refitted, one at a time, the most negative first, and each removal is recorded in sign_check. The last remaining variable is never removed: it is kept and flagged NON_POSITIVE_COEF_KEPT_LAST. The final shortlist of the scorecard (features) is what scr_sql() covers.

Points per bin

With score = a + b * logit and logit = alpha + sum(beta_j * woe_ij):

$$\text{points}_{ij} = b\,\beta_j\,\text{woe}_{ij}, \qquad \text{base} = a + b\,\alpha.$$

points_style = "distributed" spreads base / k over each characteristic (Siddiqi, 2006, chapter 6), leaving base_points = 0. The exact points stay in points_raw; points is the rounded version when points_round = TRUE. The exact score (score) and the whole-points score (score_points) are both returned by scr_apply() and both emitted by scr_sql(). A row that falls in no fitted bin (a category never seen on train, a missing value without a missing bin) gets a WOE of 0 from the binning engine, hence the points of a WOE of 0: 0, or base / k under "distributed".

See Also

Other stages: scr_align(), scr_bin(), scr_cutoff(), scr_model(), scr_reject(), scr_select(), scr_split(), scr_strategy(), scr_triage()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
sc
head(sc$points[, c("variable", "bin", "woe", "points")])
sc$metrics
sc$alignment
```

---

scr_segments *One score on many segments*

Description

Reads one score on the segments of a population (a product, a channel, a region) and says, for each, whether the score ranks and calibrates there as it does on the whole: discrimination with its standard error, the observed events against those expected from the pooled bands, the offset on the log-odds scale, the slope of the score relative to the pooled one and a suggested action.

Usage

```
scr_segments(x, ...)

## S3 method for class 'data.frame'
scr_segments(
  x,
  segment,
  by = NULL,
  n_bands = 10L,
  level = 0.95,
  min_events = 20L,
  n_boot = 0L,
  auc_tol = 0.03,
  offset_tol = 0.25,
  slope_tol = 0.25,
  score = "score",
  y = "y",
  objective = "risk",
  direction = NULL,
  weight = NULL,
```

```
  seed = NULL,
  max_cells = 1e+05,
  ...
)

## S3 method for class 'scr_scorecard'
scr_segments(
  x,
  newdata,
  segment,
  by = NULL,
  n_bands = 10L,
  level = NULL,
  min_events = 20L,
  n_boot = 0L,
  auc_tol = 0.03,
  offset_tol = 0.25,
  slope_tol = 0.25,
  target = NULL,
  seed = NULL,
  max_cells = 1e+05,
  ...
)
```

Arguments

x	A data.frame with one row per scored case, or an object from scr_scorecard().
...	Passed on to the methods; an unknown argument is an error.
segment	Name of the segment column. A missing segment is the segment "(missing)". The segments of a factor column are listed in the order of its levels.
by	Optional name of a column of periods or groups.
n_bands	Equal-share bands frozen on the pooled rows, for the expected events and the PSI.
level	Confidence level of the intervals; 1 - level is the significance of the tests. For a scorecard, NULL uses config$study_level (0.95).
min_events	Fewest events, and non-events, for an action other than "too few events".
n_boot	Bootstrap resamples of the AUC interval; 0 (default) keeps the DeLong interval.
auc_tol, offset_tol, slope_tol	Tolerances of the actions: on the difference of the AUC from the weighted mean, on the absolute offset and on the distance of the slope ratio from 1.
score, y	Column names of the score and of the 0/1 outcome (NA allowed).
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).
direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
weight	Optional column of non-negative case weights.

seed	Seed of the bootstrap. A number is local to the call (the user's random stream is restored on exit); NULL draws from the user's stream and advances it. For a scorecard, NULL uses config$seed.
max_cells	Largest number of distinct score values kept exactly.
newdata	For a scorecard: the rows to score with scr_apply(), holding the candidate variables, the target and the segment column.
target	For a scorecard: the outcome column of newdata; NULL uses the target of the scorecard.

Value

An object of class c("scr_segments", "list"):

table One row per group and segment: group, segment, n, events, rate, auc, auc_se, auc_lo, auc_hi, gini, ks, psi, psi_critical, expected, oe_ratio, oe_lo, oe_hi, offset, slope, slope_ratio, auc_diff, p_auc, p_auc_adj, p_slope, p_slope_adj and action.

test One row per group: group, segments (those in the test), auc_w, statistic, df and p_value.

pooled One row per group, the pooled rows: group, n, events, rate, auc, auc_se, auc_lo, auc_hi, gini, ks and slope.

cuts, segment, by, level, min_events, auc_tol, offset_tol, slope_tol, objective, direction, target, call The pooled cuts and the settings.

Statistics

The rows are counted once per segment and distinct score. Per segment:

- auc, gini and ks, with the DeLong standard error auc_se computed from the counts and the interval auc +/- z * auc_se (cut to [0, 1]); with n_boot > 0, the interval is the percentile interval of the count bootstrap of scr_bands(). A segment without a positive standard error (fewer than two events or non-events, or every score tied) has no interval.
- psi: the population stability index of the score distribution of the segment against the pooled one, over n_bands equal-share bands frozen on the pooled rows (see scr_psi()). The segment is part of the pooled rows, so the two distributions are not independent: psi_critical, the n-adjusted critical value at 1 - level, uses $(1/n_s - 1/N)\,\chi^2_{B-1}$ in place of $(1/n_s + 1/N)\,\chi^2_{B-1}$, with n_s and N the sizes of the segment and of the pooled rows (under weights, the Kish effective sizes, with the covariance term scaled by the weight share of the segment). It is NA when the segment is the whole pool.
- expected: the events expected by indirect standardization, $\sum_b n_{s,b} R_b$, with $n_{s,b}$ the rows of the segment in the pooled band b and R_b the pooled event rate of the band; oe_ratio = events / expected, with the Jeffreys interval of the observed rate (on the Kish effective size under weights) divided by the expected rate.
- offset = logit(observed rate) - logit(expected rate): what a segment intercept would add to the log-odds. NA when either rate is 0 or 1.
- slope: the coefficient of the score in a logistic regression of the outcome within the segment, fitted on the count table with the counts as weights (equal to the fit on the rows); slope_ratio is slope over the pooled slope. NA when the segment has a single score value or its classes are separated by the score. p_slope is the two-sided Wald test of the slope of the segment against

the slope fitted on the rest of its group (the pooled rows without the segment, an independent sample), $z = (b_s - b_r)/\sqrt{v_s + v_r}$ with the model-based variances of the two fits (scaled to the Kish effective size under weights); `p_slope_adj` is its Holm adjustment across the segments.

Rows with a missing outcome count in the volume and in the PSI only. With more than `max_cells` distinct scores the scores are pooled into cells, and the slope is fitted on the cell midpoints.

Test of equal discrimination

With $w_s = 1/se_s^2$, the inverse-variance weighted mean is $AUC_w = \sum_s w_s AUC_s / \sum_s w_s$, and

$$Q = \sum_s (AUC_s - AUC_w)^2 / se_s^2$$

is compared with a chi-square law with $S - 1$ degrees of freedom (`test`: `statistic`, `df`, `p_value`). The segments are disjoint, so their AUCs are independent. Per segment, `auc_diff` = $AUC_s - AUC_w$ is tested by a two-sided z test with the variance $se_s^2 - 1/\sum_s w_s$ (the mean contains the segment); `p_auc` is its p-value and `p_auc_adj` the Holm adjustment across the segments. Segments without a positive standard error (fewer than two events or non-events, or every score tied) are left out of the test. With exactly two segments the two tests are one and the same, so no adjustment is made; the same holds for the slope tests.

Action

A convention of this package, read in this order:

`"too few events"` Fewer than `min_events` events, or non-events, in the segment (unweighted counts).

`"separate model"` The score ranks differently: `abs(auc_diff) > auc_tol` with `p_auc_adj < 1 - level`, or `abs(slope_ratio - 1) > slope_tol` with `p_slope_adj < 1 - level`. A difference must be both material and significant: on a small segment, a slope ratio far from 1 is often noise.

`"offset"` The ranking holds but the level does not: `abs(offset) > offset_tol` and the interval of `oe_ratio` excludes 1.

`"shared"` None of the above: the pooled score serves the segment.

The tolerances are starting points, not rules; set them to the policy in force.

Groups

With `by` (a period, a sample label), the analysis is repeated within each group: the pooled reference of a segment is the whole of its group. The bands are frozen once, on all rows. Groups and segments are listed in the order of their labels; numeric columns in numeric order, and a factor segment column in the order of its levels.

References

Brown, L. D., Cai, T. T. and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133. doi:10.1214/ss/1009213286

DeLong, E. R., DeLong, D. M. and Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, 44(3), 837-845.

Thomas, L. C., Crook, J. and Edelman, D. (2017). *Credit Scoring and Its Applications*, 2nd edition. SIAM. doi:10.1137/1.9781611974560

See Also

`scr_mix_shift()` for the change of the event rate between samples, `scr_rag()` for lights per segment against a reference.

Other score-studies: `scr_bands()`, `scr_claims()`, `scr_detection()`, `scr_maturity()`, `scr_mix_shift()`, `scr_operating()`, `scr_overlap()`, `scr_rag()`, `scr_rag_plan()`, `scr_score_cross()`, `scr_tiers()`, `scr_uplift()`

Examples

```
local({
  set.seed(1)
  n <- 6000
  seg <- sample(c("app", "store", "web"), n, TRUE, c(0.5, 0.3, 0.2))
  x <- rnorm(n)
  # the score ranks the same everywhere; "web" defaults more at every score
  d <- data.frame(channel = seg, score = round(600 + 50 * x),
                  y = rbinom(n, 1, plogis(-2 - x + 0.6 * (seg == "web"))))
  sg <- scr_segments(d, segment = "channel")
  print(sg)
  sg$test
})
```

---

scr_select *Select variables for the scorecard*

Description

Shortcut that chains `scr_split()`, `scr_triage()`, `scr_bin()` and `scr_model()` on a table and a binary target, and returns an object with the shortlist, the complete audit funnel, the gains table and the production SQL of the approved variables. Every stage remains callable on its own for whoever wants more control (hybrid interface).

Usage

```
scr_select(
  data,
  target,
  config = scr_config(),
  drop = character(),
  date_col = config$oot_date_col,
  event_level = NULL,
```

```
  export = NULL,
  copy = TRUE
)
```

Arguments

data	A data.frame or data.table with the target, the candidates and, if any, the date column of the out-of-time cut.
target	Name of the target column (0/1, logical, or a two-level factor/character).
config	An object from scr_config().
drop	Columns that are never candidates. They stay in the funnel as 00.config.
date_col	Date column of the out-of-time cut. Defaults to config$oot_date_col.
event_level	Which target value counts as the event; see scr_split().
export	Directory to write the deliverables to. NULL (default) writes nothing; use scr_export() later.
copy	If TRUE (default), works on a copy of data.

Value

An object of class scr_result. Read it with scr_selected(), scr_funnel(), scr_gains(), scr_sql(), scr_leakage() and summary(); continue with scr_scorecard(); write it with scr_export().

Reproducibility

With the same data, the same target and the same config$seed, the result is identical with one or several nthread: the seed governs the random split, the cross-validation, the classifier subsample, the trees and the bootstrap, and the binning is deterministic per column.

See Also

scr_run() for several targets straight from the database, scr_scorecard() for the next step.

Other stages: scr_align(), scr_bin(), scr_cutoff(), scr_model(), scr_reject(), scr_scorecard(), scr_split(), scr_strategy(), scr_triage()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
res
scr_selected(res)
head(scr_funnel(res, only_selected = TRUE))
```

`scr_selected` *Variables approved for the scorecard*

Description

The final shortlist, in consensus order (the first is the strongest). It is exactly the list `scr_sql()` covers and `scr_scorecard()` fits.

Usage

```
scr_selected(x, which = c("final", "consensus", "manual"))
```

Arguments

`x`	An object from `scr_select()`.
`which`	`"final"` (default), `"consensus"` or `"manual"` (`NULL` when no manual choice was made).

Details

After `scr_classing_apply()` the result carries two lists: the automatic consensus and the analyst's final choice; `which` picks one, and the default is the final one so that every downstream function follows the analyst's decision.

Value

A character vector of column names.

See Also

Other accessors: `scr_funnel()`, `scr_gains()`, `scr_leakage()`, `scr_result`, `scr_score_gains()`, `scr_score_metrics()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
scr_selected(res)
```

scr_split *Stage 0: type the data and split train and hold-out*

Description

First stage of the pipeline, callable on its own. Converts the target to 0/1 (resolving `event_level`), types the candidates (numerics become `double`, everything else becomes `character`) and splits train and hold-out **before** any supervised fit.

Usage

```
scr_split(
  data,
  target,
  date_col = NULL,
  ratio = 0.3,
  seed = NULL,
  event_level = NULL,
  drop = character(),
  copy = TRUE
)
```

Arguments

`data`	A `data.frame` or `data.table` with the target and the candidates.
`target`	Name of the target column. Binary: 0/1, logical, or a two-level factor/character.
`date_col`	Date column of the out-of-time cut. `NULL` uses a random stratified split.
`ratio`	Target hold-out fraction.
`seed`	Seed of the random split. `NULL` draws from the session's random stream (reproducible only through a `set.seed()` of your own; `scr_select()` passes the `seed` of `scr_config()`). A seed is applied locally: the session's random stream is restored on exit.
`event_level`	Which target value counts as the event. `NULL` uses the convention (`1`, or the second alphabetical level).
`drop`	Columns that are never candidates (identifiers, sibling targets, free text). They stay in the funnel as `00.config`.
`copy`	If `TRUE` (default), works on a copy of `data`. `FALSE` modifies a `data.table` by reference (target and typing), saving memory; a `data.frame` is always converted, hence copied.

Details

The split prefers out-of-time by `date_col`: it is the only one that tests generalization to a future period. The cut is made on the **distinct** date values, not by row quantile: it picks the smallest set of most recent periods that already reaches `ratio` of the population. Without a date column, or

with a single period, it falls back to random stratified by the target. The date column is never a candidate: it is the key of the split and leaves the contest. A text date column is read as an ISO date (`YYYY-MM-DD`, `YYYY/MM/DD`, `YYYY-MM`) or as all-digit periods (`YYYYMM`); rows with a missing date belong to no period and are left out of both train and hold-out, with a warning in the log.

Value

An `scr_split` object with `data` (typed), `target`, `train_idx`, `holdout_idx`, `method`, `cutoff`, `date_col` and `cols` (`features`, `var_num`, `var_cat`, `dropped`, `event`).

Event orientation

`event_level` changes **what is modeled**. Passing `0` makes class 0 the event: the sign of every WOE flips, the emitted SQL changes, the points change. For a text target, the second level in alphabetical order is the event by default, and the choice is always reported. Not to be confused with `config$objective`, which only orients the reading and the points scale.

See Also

Other stages: `scr_align()`, `scr_bin()`, `scr_cutoff()`, `scr_model()`, `scr_reject()`, `scr_scorecard()`, `scr_select()`, `scr_strategy()`, `scr_triage()`

Examples

```
sp <- scr_split(scr_demo, "default", date_col = "ref_date", drop = "id")
sp
length(sp$train_idx); length(sp$holdout_idx)
```

---

scr_sql *Production SQL*

Description

Code ready to run in the database, covering **exactly** the approved variables (or those of the scorecard), in blocks in this order:

Usage

```
scr_sql(x, table = NULL, dialect = NULL, file = NULL, ...)

## S3 method for class 'scr_capital'
scr_sql(
  x,
  table = NULL,
  dialect = NULL,
  file = NULL,
  level = c("exposure", "portfolio"),
  ...
```

```
)

## S3 method for class 'scr_ead'
scr_sql(x, table = NULL, dialect = NULL, file = NULL, ...)

## S3 method for class 'scr_lgd'
scr_sql(x, table = NULL, dialect = NULL, file = NULL, ...)

## S3 method for class 'scr_pd'
scr_sql(x, table = NULL, dialect = NULL, file = NULL, ...)

## S3 method for class 'scr_result'
scr_sql(x, table = NULL, dialect = NULL, file = NULL, output = NULL, ...)

## S3 method for class 'scr_scorecard'
scr_sql(
  x,
  table = NULL,
  dialect = NULL,
  file = NULL,
  what = c("score", "woe", "all"),
  keep_columns = NULL,
  ...
)

## S3 method for class 'scr_study'
scr_sql(
  x,
  table = NULL,
  dialect = NULL,
  file = NULL,
  score = "score",
  numbered = TRUE,
  ...
)
```

Arguments

x	An object from scr_select(), scr_scorecard(), scr_pd(), scr_lgd(), scr_ead(), scr_capital(), scr_bands() or scr_tiers().
table	Source table name, written verbatim (it may be qualified, schema.table, and is never quoted: pass only a trusted name). NULL uses config$sql_table.
dialect	Dialect ("ansi", "databricks", "spark", "hive", "mysql", "mariadb", "sqlserver", "bigquery", "postgres", "oracle", "snowflake", "redshift", "duckdb", "sqlite"). NULL uses config$sql_dialect.
file	Path to write to. NULL (default) returns the lines.
...	Passed on to the methods.

level	For scr_capital: "exposure" (default, one row per exposure with el, k, rw, rwa) or "portfolio" (the aggregate by segment).
output	For scr_result: "woe", "bin" or "both". NULL uses config$sql_output.
what	For scr_scorecard: "score" (default: the three blocks, with the points per variable, the exact score and the whole-points score), "woe" (the WOE/BIN SQL of the scorecard variables only) or "all" (the three blocks plus, for every variable, its bin label, WOE and points side by side: the deployment layout that reports the band of each variable next to the score).
keep_columns	For scr_scorecard: key columns carried untransformed into the output (for example the customer identifier and the reference date); NULL uses config$sql_keep_columns.
score	For scr_study: name of the score column of table.
numbered	For a tiers study: TRUE (default) emits the tier labels with their order in front, FALSE the plain labels.

Details

1. CTE base_scr: reproduces the Stage 1 pre-processing - imputation of missing and sentinel by the **training** median, special-population flags, COALESCE of the categorical missing.
2. The WOE/BIN transformation, emitted by OptimalBinningWoE::obwoe_sql() from the authoritative cut points with full precision.
3. (Scorecard) CTE woe_scr with WOE and bin index, followed by the final SELECT with score (exact, a + b * logit), <f>_points per variable and score_points (whole points).

The order matters: without the first block, the WOE would be applied to data different from what was binned. Column names are quoted with the dialect's delimiters only when they are not plain identifiers or are reserved words, the same rule OptimalBinningWoE::obwoe_sql() applies, so every block names a column the same way. A row whose value falls in no fitted bin (a category never seen on train) takes a WOE of 0 and the points of a WOE of 0, in the SQL as in scr_apply(). The score computed by the SQL matches scr_apply() numerically, by an automated test that runs both paths.

Value

A character vector with the SQL (invisibly, when file is given).

IRB models

scr_pd wraps the scorecard SQL in a common table expression and adds a CASE on the score cut points that yields grade and pd_final. scr_lgd chains the driver bins of both stages, the logits, the pool CASE and the floored result. scr_ead computes the utilization and the undrawn amount, assigns the pool from the frozen cut points and applies the greatest of the model, the drawn amount and the standardized floor. scr_capital carries the constants of every pool (PD, LGD, k, risk weight) in a pool_params table joined on segment and grade, so no normal quantile is evaluated at run time; level chooses the exposure or the portfolio output.

Score studies

For a score study (scr_bands(), scr_tiers()), the SQL reads the score column of table and adds tier and tier_label with a CASE on the frozen cuts (score >= cut is the upper side; a NULL score gives a NULL tier). table and dialect default to the configuration of the scorecard the study came from, else to "your_table" and "ansi". The tiers computed by the SQL match scr_apply(), by an automated test.

The labels of a tiers study carry their order, '01.very high' for the tier with the highest event rate down to the tier with the lowest, as in scr_apply(), so ORDER BY tier_label lists the event-richest tier first; numbered = FALSE emits the plain labels. Band labels are intervals and never get a prefix.

See Also

Other production: predict.scr_align(), scr_apply(), scr_export(), scr_monitor(), scr_monitoring_plan(), scr_reasons()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
cat(head(scr_sql(res, table = "prd.customers", dialect = "databricks"), 20), sep = "\n")
sc <- scr_scorecard(res)
cat(tail(scr_sql(sc), 12), sep = "\n")
```

---

scr_strategy *Stage 6: strategy table per band, with marginal expected profit*

Description

Score bands (by default the deciles frozen on train) with volume, event rate, the event and non-event distributions, decision and the expected result per account. The good case is the non-event under objective = "risk" (credit, fraud) and the event under "propensity"; the bad case is the other one. With p the rate of the bad case in the band (the event rate under risk, one minus it under propensity):

$$EP = (1 - p)\,\text{revenue_good} - p\,\text{loss_bad},$$

which makes visible the band that is profitable **at the margin** even with a high rate of the bad case. EP = 0 at the break-even rate of the bad case, revenue_good / (revenue_good + loss_bad). The object stores it as an event rate (breakeven): the same value under risk, and loss_bad / (revenue_good + loss_bad) under propensity, where a band is targeted at or above it.

Usage

```
scr_strategy(
  x,
  breaks = NULL,
  decisions = NULL,
  revenue_good = 1,
  loss_bad = 1,
  sample = "holdout",
  rule = c("breakeven", "crossing")
)
```

Arguments

x	An object from scr_scorecard().
breaks	Band cut points. NULL uses the deciles frozen on train.
decisions	Vector of decisions, one per band (from the first row of the table to the last). NULL derives them from rule; when given, it overrides rule.
revenue_good	Expected revenue per account of the good case (the non-event under risk, the event under propensity; default 1).
loss_bad	Expected loss per account of the bad case (default 1; with both defaults the break-even is 50%). revenue_good and loss_bad cannot both be 0.
sample	"holdout" (default) or "train".
rule	"breakeven" (default) or "crossing"; see the section Decision rules.

Details

The table runs from the band richest in the good case to the poorest: the safest band first under risk, the most likely first under propensity.

Value

An scr_strategy object with

table One row per band: id, band, min_score, max_score, n, pct, events, event_rate, pct_event, pct_nonevent, odds_event, log_odds, decision, ep_per_account, band_profit, cum_pct, cum_event_rate and cum_profit.

breakeven The break-even event rate.

crossing A list: cut, the score where the upper side of the crossing starts (score >= cut, the convention of scr_cutoff()), frozen on the training scores like the bands: midway between the largest training score at or below the band edge of the crossing and the smallest training score above it. score >= cut then reproduces the split of the bands on train and on any score seen in training; a score of another sample strictly between those two training scores can fall on the other side. When breaks is a number of intervals (whose edges come from sample), or no training score lies on one side of the edge, the cut is the midpoint between the bands on sample; ks, the distance D_k at it; after_band, the last band on the good side; single_crossing, whether log_odds changes sign exactly once along the table. All NA when undefined.

`objective`, `rule` The objective of the scorecard and the rule used.

`revenue_good`, `loss_bad`, `sample`, `direction`, `target` The parameters and the scorecard's direction and target.

Event and non-event distributions

With e_k events and m_k non-events in band k, and E and M their totals over the sample:

$$\text{pct_event}_k = e_k/E, \qquad \text{pct_nonevent}_k = m_k/M,$$

$$\text{odds_event}_k = \text{pct_event}_k/\text{pct_nonevent}_k, \qquad \text{log_odds}_k = \ln \text{odds_event}_k.$$

`log_odds` is the WOE of the band, event-oriented like the WOE of the variables: `log_odds > 0` if and only if the band event rate is above the overall event rate, that is, the lift of the band is above 1 (exact when every band has both classes; under the smoothing below, a band at the overall rate can fall on either side). When a band has no events or no non-events, 0.5 is added to the counts of every band for `odds_event` and `log_odds`; the shares stay exact. With a single class in the sample, the shares of the missing class and every ratio are `NA`. This `log_odds` is the `woe` column of `scr_score_gains()`, not its `log_odds`, which is the log of the band odds in the orientation of the scale.

Decision rules

`rule = "breakeven"` (default) gives the good label (`"approve"` under risk, `"target"` under propensity) to a band whose rate of the bad case is at or below break-even, `"review"` to one up to 25% above it, and the bad label (`"decline"` or `"skip"`) to the rest.

`rule = "crossing"` cuts where the event and non-event distributions are furthest apart. With

$$D_k = \left| \sum_{j \le k} \text{pct_event}_j - \sum_{j \le k} \text{pct_nonevent}_j \right|$$

over the first k rows of the table, the first maximum of D_k over the boundaries between rows is the KS of the table; the rows up to it get the good label and the rest the bad label, with no review band. When `log_odds` is monotone along the table this is where it changes sign, the band event rate crossing the overall rate; when it is not, the cut still gives a contiguous set of bands. The boundary is always computed and stored in `crossing`. It is undefined with fewer than two bands or a single class in the sample, and `rule = "crossing"` is then an error. Scores outside `breaks` form a last row with a missing `band`, which gets no decision (`NA`) under the crossing rule; the shares, and hence `ks`, stay relative to the whole sample, that row included.

`decisions`, when given, overrides either rule.

See Also

Other stages: `scr_align()`, `scr_bin()`, `scr_cutoff()`, `scr_model()`, `scr_reject()`, `scr_scorecard()`, `scr_select()`, `scr_split()`, `scr_triage()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_strategy(sc, revenue_good = 1080, loss_bad = 4500)
# approve down to where the event and non-event distributions cross
st <- scr_strategy(sc, rule = "crossing")
st$crossing
st$table[, .(band, event_rate, log_odds, decision)]
```

`scr_tiers` *Tiers of a score: a few labeled levels of risk or propensity*

Description

Groups the score into a small number of contiguous tiers (typically 3, 5 or 7, labeled from "very low" to "very high") **fitted on the reference sample** and read on the reference and the study samples. Tiers are a policy and communication device; they are not a rating scale in the sense of the internal ratings-based approach (see `scr_grades()` for that).

Usage

```
scr_tiers(x, ...)

## S3 method for class 'scr_scorecard'
scr_tiers(
  x,
  n_tiers = 5L,
  method = c("optimal", "anchored", "quantile"),
  criterion = c("deviance", "iv"),
  anchors = NULL,
  conservative = FALSE,
  level = NULL,
  min_pct = NULL,
  min_events = NULL,
  alpha = 0.05,
  max_bins = NULL,
  labels = NULL,
  round_to = NULL,
  n_boot = 0L,
  seed = NULL,
  sample = "holdout",
  reference = "train",
  max_cells = 1e+05,
  ...
```

```
)

## S3 method for class 'data.frame'
scr_tiers(
  x,
  score = "score",
  y = "y",
  objective = "risk",
  direction = NULL,
  weight = NULL,
  sample = NULL,
  reference = NULL,
  study = NULL,
  counts = FALSE,
  n = "n",
  events = "events",
  n_tiers = 5L,
  method = c("optimal", "anchored", "quantile"),
  criterion = c("deviance", "iv"),
  anchors = NULL,
  conservative = FALSE,
  level = 0.95,
  min_pct = 0.05,
  min_events = 20L,
  alpha = 0.05,
  max_bins = 100L,
  labels = NULL,
  round_to = NULL,
  n_boot = 0L,
  seed = NULL,
  max_cells = 1e+05,
  ...
)
```

Arguments

x	An object from scr_scorecard(), or a data.frame with one row per scored case (or one row per score value with counts = TRUE).
...	Passed on to the methods; an unknown argument is an error.
n_tiers	Number of tiers requested (2 to 9).
method	"optimal", "anchored" or "quantile".
criterion	For "optimal": "deviance" (binomial log-likelihood) or "iv" (information value).
anchors	For "anchored": event-rate thresholds, as numbers in (0, 1) or the string "overall" (the reference event rate); a character vector may mix both.
conservative	For "anchored": compare the anchors with the lower Jeffreys bound of the block rate instead of the rate.

level	Confidence level of the Jeffreys intervals. For a scorecard, NULL uses config$study_level (0.95).
min_pct	Smallest volume share of a tier. For a scorecard, NULL uses config$tier_min_pct (0.05).
min_events	Fewest events, and fewest non-events, of a tier. For a scorecard, NULL uses config$tier_min_events (20).
alpha	Significance level of the adjacency test (divided by n_tiers - 1) and of all_distinct.
max_bins	Pre-bins of the search (2 to 500). For a scorecard, NULL uses config$tier_max_bins (100).
labels	Optional labels, one per tier achieved, in ascending order of the event rate. They get the order prefix of tier_label too.
round_to	Optional positive number: cuts are rounded to its multiples.
n_boot	Bootstrap resamples of the stability study (0, the default, skips it).
seed	Seed of the stability bootstrap. A number is local to the call (the user's random stream is restored on exit); NULL draws from the user's stream and advances it. For a scorecard, NULL uses config$seed.
sample	For a scorecard: the study sample(s), "holdout" (default) and/or "train". For a data.frame: the name of a column with sample labels, or NULL (all rows are one sample, reference and study at once).
reference	For a scorecard: the sample the bands are frozen on ("train"). For a data.frame: the label of the reference sample; NULL takes the first level of the sample column (the levels of a factor in their order, numbers in numeric order, text sorted).
max_cells	Largest number of distinct score values kept exactly.
score, y	Column names of the score and of the 0/1 outcome (NA allowed).
objective	"risk" (the event is the bad case) or "propensity" (the event is the good case).
direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
weight	Optional column of non-negative case weights.
study	Labels of the study samples; NULL takes every level other than the reference.
counts	TRUE when x is pre-aggregated: one row per score value with the columns score, n and events (and, optionally, value and value_events).
n, events	Column names of the counts when counts = TRUE.

Value

An object of class c("scr_study_tiers", "scr_study", "list"):

table One row per sample and tier, event-richest tier first: sample, tier, label, tier_label (the label with its order in front, "01." for the event-richest tier; see the section Labels), score_lo, score_hi, n, pct, events, rate, rate_lo, rate_hi, lift, pct_event, pct_nonevent, woe, p_adjacent (one-sided Fisher exact test that the tier has a higher event rate than the next lower tier) and p_adjacent_adj (Holm).

summary One row per sample: sample, n, events, rate, n_tiers, monotone (the rates rise with the tier), all_distinct (every p_adjacent_adj below alpha), iv and psi (of the tier mix against the reference).

`cuts`, `cuts_raw` The cuts in use (rounded when `round_to` is given) and the fitted ones.

`labels`, `tier_labels` The tier labels, plain and numbered, lowest rate first.

`codes`, `code_labels` Tier number and plain label of every interval in ascending score order, used by `scr_apply()` and `scr_sql()`.

`method`, `criterion`, `measure`, `n_tiers_requested`, `n_tiers` The fit.

`ledger` One row per step: `step`, `n_tiers`, `status` and `detail`.

`stability` `NULL`, or with `n_boot > 0` a list: `cuts` (per cut: `score`, `median`, `q25`, `q75`, `iqr` and `n_same`, the resamples that gave the same number of tiers), `agreement`, `same_count` and `n_boot`.

`objective`, `direction`, `level`, `alpha`, `min_pct`, `min_events`, `reference`, `samples`, `target`, `hist`, `call` The settings and the count table.

Method

The reference scores are cut into at most `max_bins` pre-bins of equal share (tie-safe, as in `scr_bands()`); adjacent pre-bins are then pooled (pool adjacent violators) until the event rate rises monotonically toward the event-rich side of the score. Every tier is a run of these blocks, so the tier rates are monotone on the reference.

`method = "optimal"` searches every segmentation of the blocks into `n_tiers` contiguous tiers with an exact dynamic program and keeps the one that maximizes the binomial log-likelihood

$$\sum_t \left[e_t \ln p_t + (n_t - e_t) \ln(1 - p_t) \right], \qquad p_t = e_t / n_t$$

(`criterion = "deviance"`, the same as minimizing the deviance) or the information value (`criterion = "iv"`), subject to: every tier holds at least `min_pct` of the volume; at least `min_events` events **and** `min_events` non-events; and every pair of adjacent tiers is distinct by a one-sided Fisher exact test at `alpha / (L - 1)`, where L is the tier count being tried. The rates and the objective use the weighted counts; the event constraints and the Fisher test use the unweighted counts. When no segmentation into `n_tiers` tiers meets the constraints, `n_tiers - 2`, `n_tiers - 4`, ... down to 2 are tried (an odd count stays odd while possible); when even 2 tiers are infeasible, the last resort is a single tier. Every attempt is recorded in `ledger` and a warning is raised. The tiers are never relabeled silently: the labels follow the number of tiers achieved.

The previous tier enters the state of the program, so its cost is $O(LM^3)$ time and LM^2 memory for L tiers and M blocks (compiled code); the Fisher tests are cached when $M \leq 200$ and recomputed above. `n_tiers` is capped at 9 and `max_bins` (hence M) at 500, and the stability study refits once per resample, so `n_boot` multiplies the cost.

`method = "anchored"` places one cut per value of `anchors` (event-rate thresholds; `"overall"` stands for the reference event rate): the cut sits before the first block, from the low-rate side, whose event rate (its lower Jeffreys bound with `conservative = TRUE`) reaches the anchor. `n_tiers` is then `length(anchors) + 1` and the argument is ignored. `method = "quantile"` cuts the reference into `n_tiers` tiers of equal share, the baseline.

`round_to` rounds every cut to the nearest multiple (a policy-friendly cut-off, such as 500 or 520 points); the tiers are then re-evaluated on every sample with the rounded cuts, and both sets of cuts are kept. When the scores were pooled into cells (`max_cells`), the count table is rebuilt with the rounded cuts as forced cell edges, so the reported tiers agree exactly with `scr_apply()` and `scr_sql()`. Tiers are left-closed: `score >= cut` is the upper side, the convention of `scr_cutoff()`.

With n_boot > 0, the stability of the fit is measured by redrawing the event and non-event counts of the reference pre-bins from a multinomial law and refitting: per cut, the median and interquartile range of the refitted cut in score units, and the agreement rate (the share of the reference volume that keeps its tier).

Labels

Tiers are numbered by event rate, lowest first: 2 tiers are labeled "low", "high"; 3 tiers "low", "medium", "high"; 4 tiers "low", "medium low", "medium high", "high"; 5 tiers "very low", "low", "medium", "high", "very high"; 6 tiers "very low", "low", "medium low", "medium high", "high", "very high"; 7 tiers add "extremely low" and "extremely high" to the five; 1, 8 and 9 tiers are numbered "T1", "T2", ... The labels describe the event rate, so under objective = "risk" they read as risk and under "propensity" as propensity (measure). The table lists the event-richest tier first.

For production, every label also exists with its order in front (tier_label): "01." for the event-richest tier, the first row of the table, then "02.", ... down to the tier with the lowest event rate, under every objective and direction, and for labels given in labels too. With five tiers of a credit score, tier 5 is "01. very high" and tier 1 is "05. very low": the number tier rises with the event rate, the prefix sorts from the highest rate down. The prefix is zero-padded to two digits. scr_apply() and scr_sql() return these numbered labels (numbered = FALSE gives the plain ones), so their output joins to the table by tier_label.

References

Brown, L. D., Cai, T. T. and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133. doi:10.1214/ss/1009213286

Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.

Yurdakul, B. and Naranjo, J. (2020). Statistical properties of the population stability index. *Journal of Risk Model Validation*, 14(4), 89-100.

See Also

scr_bands(), scr_rag(); scr_apply() and scr_sql() assign the tiers in production.

Other score-studies: scr_bands(), scr_claims(), scr_detection(), scr_maturity(), scr_mix_shift(), scr_operating(), scr_overlap(), scr_rag(), scr_rag_plan(), scr_score_cross(), scr_segments(), scr_uplift()

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
tr <- scr_tiers(sc, n_tiers = 5)
tr
tr$table[sample == "holdout", .(tier, label, score_lo, score_hi, pct, rate)]
```

```
# policy cuts on round numbers, and the tiers assigned to new scores
tr10 <- scr_tiers(sc, n_tiers = 3, round_to = 10)
tr10$cuts
head(scr_apply(tr10, c(480, 530, 600)))

# anchors on the event rate: below, around and above the overall rate
scr_tiers(sc, method = "anchored", anchors = c(0.08, "overall", 0.25))$summary
```

`scr_triage` *Stage 1: descriptive triage and sentinel resolution*

Description

Profiles every candidate **on the training rows only**, decides its fate and materializes the clean data for train and hold-out with the same values (training median, `"MISSING"` level). A sentinel or missing mass with weight (`special_min_share`) and signal (`special_min_woe`) becomes a categorical flag `<column><flag_suffix>`, which the engine bins and emits in SQL natively.

Usage

```
scr_triage(split, config = scr_config())
```

Arguments

`split`	An object from `scr_split()`.
`config`	An object from `scr_config()`.

Details

Failures at this stage: `CONSTANT`, `NEAR_CONSTANT`, `TOO_MANY_MISSING`, `HIGH_CARDINALITY`, `NO_SIGNAL` (coarse IV below `min_iv_quick`) and `DUPLICATE_OF:<column>`. A failed numeric whose special-population flag survives carries the suffix `;RESCUED_AS_FLAG`.

Value

An `scr_triage` object with `profile` (one row per candidate and derived flag), `ledger` (the source of truth of the pre-processing the SQL reproduces), `keep`, `derived`, `clean` (target + survivors + flags, with no `NA`) and the originating `split`.

See Also

Other stages: `scr_align()`, `scr_bin()`, `scr_cutoff()`, `scr_model()`, `scr_reject()`, `scr_scorecard()`, `scr_select()`, `scr_split()`, `scr_strategy()`

Examples

```
sp <- scr_split(scr_demo, "default", date_col = "ref_date", drop = "id")
tr <- scr_triage(sp, scr_config(verbose = FALSE))
tr
table(tr$profile$triage_reason)
```

`scr_uplift` *Uplift of a treatment along a score*

Description

Reads a score on a treated and a control group (a campaign with a hold-out): the difference of the event rates per score band with its interval, the incremental events, the Qini and uplift curves with their areas, and a check that the two groups have the same score distribution.

Usage

```
scr_uplift(x, ...)

## S3 method for class 'data.frame'
scr_uplift(
  x,
  score = "score",
  y = "y",
  treat = "treat",
  n_bands = 10L,
  cuts = NULL,
  level = 0.95,
  n_boot = 200L,
  seed = NULL,
  objective = "propensity",
  direction = NULL,
  weight = NULL,
  max_cells = 1e+05,
  boot_cells = 10000,
  ...
)
```

Arguments

`x`	A `data.frame` with one row per case.
`...`	Passed on to the methods; an unknown argument is an error.
`score`, `y`, `treat`	Column names of the score, of the 0/1 outcome (`NA` allowed) and of the 0/1 or logical treatment indicator (1 = treated).
`n_bands`	Equal-share bands of the score when `cuts` is not given.
`cuts`	Optional ascending cuts of the score, or an object from `scr_bands()` or `scr_tiers()`, whose cuts, numbers and labels are then used, with its objective and direction.
`level`	Confidence level of the intervals.
`n_boot`	Bootstrap resamples of the Qini and AUUC intervals (`0` skips them).
`seed`	Seed of the bootstrap. A number is local to the call (the user's random stream is restored on exit); `NULL` draws from the user's stream and advances it. For a scorecard, `NULL` uses `config$seed`.

objective	"propensity" (default: the event is the outcome sought, and a higher score means a higher propensity) or "risk".
direction	"higher_is_safer" or "higher_is_riskier"; NULL derives it from objective.
weight	Optional column of non-negative case weights.
max_cells	Largest number of distinct score values kept exactly.
boot_cells	Largest number of score cells resampled exactly by the bootstrap (default 10,000; Inf for no pooling). See the section Method.

Value

An object of class c("scr_uplift", "list"):

table One row per band, event-richest first: band, label, score_lo, score_hi, n_t, n_c, events_t, events_c, rate_t, rate_c, uplift, uplift_lo, uplift_hi, incremental, cum_incremental, cum_pct_treated and type.

summary One row: n_t, n_c, rate_t, rate_c, uplift, uplift_lo, uplift_hi, qini, qini_lo, qini_hi, auuc, auuc_lo, auuc_hi, psi, psi_critical and psi_flag.

curve The curves, at most about 1,000 rows spread evenly in the treated share (the areas use every cell): cut (the score boundary), pct_treated, pct_all, n_t, n_c, qini, qini_random (the straight line) and uplift.

randomization A list: psi, critical, flag and note.

cuts, codes, labels, level, n_boot, objective, direction, score, target, treat, weighted, call The cuts and the settings.

Bands

The bands are cut on the score of both arms together, tie-safe as in scr_bands(), the event-richest end first (band = 1). Per band, with n_t, n_c the rows with a known outcome and r_t, r_c the event rates of the treated and of the control:

- uplift = $r_t - r_c$, with the Newcombe hybrid score interval: from the Wilson intervals $[l_t, u_t]$ and $[l_c, u_c]$ of the two rates,

$$lo = r_t - r_c - \sqrt{(r_t - l_t)^2 + (u_c - r_c)^2}, \qquad hi = r_t - r_c + \sqrt{(u_t - r_t)^2 + (r_c - l_c)^2}.$$

  Under weights the Wilson intervals use the Kish effective sizes.

- incremental = uplift $\times n_t$: the events of the treated that the treatment added; cum_incremental is its running sum from the first band (a band with an empty arm adds nothing) and cum_pct_treated the cumulative share of the treated.

- type, a convention of this package: "persuadable" when the interval lies above 0, "negative" when it lies below, "no effect" when it covers 0.

Qini and AUUC

The score cells are accumulated from the event-rich end. With $E_t(k)$, $E_c(k)$ the cumulative events and $N_t(k)$, $N_c(k)$ the cumulative rows of each arm after k cells:

- the Qini curve is $Q(k) = E_t(k) - E_c(k)N_t(k)/N_c(k)$ (the second term is 0 while no control row is in), read against $N_t(k)$;
- `qini` is the area between the Qini curve and the straight line from the origin to its end point, by trapezoids from the origin, divided by the square of the treated volume:

$$qini = \frac{1}{N_t^2} \sum_k \frac{Q(k-1) + Q(k)}{2} (N_t(k) - N_t(k-1)) - \frac{Q(K)}{2N_t}.$$

  It is 0 for a score that orders at random and positive when the incremental events come first;
- the uplift curve is $U(k) = (E_t(k)/N_t(k) - E_c(k)/N_c(k))(N_t(k) + N_c(k))/N$ (an arm with no row yet has rate 0), read against the cumulative share of all rows $(N_t(k)+N_c(k))/N$; `auuc` is the area under it by trapezoids from the origin. A score that orders at random gives about half the overall uplift.

Rows with a missing outcome are left out of the rates and of the curves.

`qini` is not Radcliffe's Q, which divides the same area by that of an ideal curve; here the area is divided by N_t^2 only, so it reads as incremental events per treated row, averaged over the depth.

The intervals of `qini` and `auuc` are percentile intervals of a bootstrap on the counts, stratified by arm: each resample draws, for the treated and for the control separately, the counts per score cell and outcome from one multinomial law with the observed shares and the unweighted size of the arm, the law of a row bootstrap within each arm. The event totals of the arms vary from one resample to the next, so the intervals carry the uncertainty of the overall uplift as well as that of the ordering. Above `boot_cells` cells the resamples run on pooled cells and are shifted to the point estimates, as in `scr_bands()`. A given `seed` is local to the call.

Randomization check

Under a randomized assignment the score has the same distribution in both arms. `randomization` holds the PSI of the treated against the control over the bands, with the n-adjusted critical value at `1 - level` (see `scr_psi()`); when the PSI exceeds it, `flag` is `TRUE` and `note` says so: the arms then differ along the score and the uplift may reflect the assignment, not the treatment.

References

Newcombe, R. G. (1998). Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in Medicine*, 17(8), 873-890.

Radcliffe, N. J. (2007). Using control groups to target on predicted lift: building and assessing uplift models. *Direct Marketing Analytics Journal*, 1, 14-21.

Yurdakul, B. and Naranjo, J. (2020). Statistical properties of the population stability index. *Journal of Risk Model Validation*, 14(4), 89-100.

See Also

`scr_bands()` for the bands, `scr_operating()` for the cut of a campaign under a budget.

Other score-studies: `scr_bands()`, `scr_claims()`, `scr_detection()`, `scr_maturity()`, `scr_mix_shift()`, `scr_operating()`, `scr_overlap()`, `scr_rag()`, `scr_rag_plan()`, `scr_score_cross()`, `scr_segments()`, `scr_tiers()`

Examples

```
local({
  set.seed(1)
  n <- 8000
  x <- rnorm(n)
  treat <- rbinom(n, 1, 0.5)
  # the offer works on the customers with a high score only
  d <- data.frame(score = round(500 + 50 * x), treat = treat,
                  y = rbinom(n, 1, plogis(-1.5 + 0.5 * x + treat * pmax(x, 0))))
  up <- scr_uplift(d, n_bands = 5, n_boot = 50, seed = 1)
  print(up)
  up$summary[, c("uplift", "qini", "qini_lo", "qini_hi", "auuc")]
})
```

`scr_verbose` *Switch progress messages on or off*

Description

Large tables take tens of minutes, and the pipeline reports every stage as it runs: stage name, input and output counts, elapsed time. Messages go through `message()` as single lines, with no progress bar that redraws itself, so that a scheduled job (`Rscript` in batch) produces a readable log. `suppressMessages()` works too; this function exists to switch them off persistently, without wrapping every call. The `verbose` key of `scr_config()` has the same effect per run.

Usage

```
scr_verbose(on = NULL)
```

Arguments

`on`	`TRUE` to switch on, `FALSE` to switch off, `NULL` (default) to query the current state only.

Value

The verbosity state in force *before* the call, invisibly, so that `old <- scr_verbose(FALSE); ...; scr_verbose(old)` restores it.

See Also

Other configuration: `scr_config()`, `scr_config_keys()`, `scr_presets()`

Examples

```
old <- scr_verbose(FALSE)   # silence, keeping the previous state
scr_verbose(old)            # restore
scr_verbose()               # query
```

scr_workout *Workout LGD: the reference data set from default events and cash flows*

Description

Builds the reference data set (RDS) of realized loss given default, one row per default event, from a table of default events and the long table of their post-default cash flows. Every cash flow is discounted to the default date at the reference rate in force at that date plus `lgd_discount_add_on`, with monthly compounding over whole months:

$$\mathrm{PV} = \frac{A}{(1 + r/12)^{t}}$$

where `t` is the number of whole months between the default date and the cash-flow date. The realized LGD is the economic loss

$$\mathrm{LGD} = \frac{E - \mathrm{PV}(R) + \mathrm{PV}(C) + \mathrm{PV}(D) + C^{\mathrm{ind}}}{E}$$

with `E` the exposure at default, `R` recoveries, `C` direct costs, `D` drawings after default and `C^ind` the indirect costs allocated by `lgd_cost_allocation`.

Usage

```
scr_workout(
  defaults,
  cashflows,
  rates = NULL,
  config = scr_config(),
  indirect_costs = 0,
  obs_date = NULL,
  keep_rows = FALSE
)
```

Arguments

`defaults`	A `data.frame` or `data.table` with one row per default event: `default_id`, `facility_id`, `default_date`, `ead`, `product`, `status` (`"closed"`, `"cured"` or `"open"`), optionally `close_date`, plus any driver columns, which are carried into the RDS.
`cashflows`	Long table: `default_id`, `date`, `amount`, `type` (`"recovery"`, `"direct_cost"` or `"drawing"`).
`rates`	Optional table `(date, rate)` of the annual reference rate; the rate in force at the default date is used. `NULL` uses the flat `lgd_discount_rate` of the configuration.
`config`	A `scr_config()`; keys `lgd_*`.
`indirect_costs`	Total indirect workout cost to allocate: a single number, or a table `(product, amount)` allocated within product.

obs_date	Observation date; NULL uses the latest date seen.
keep_rows	Keep the cash-flow table with its present values.

Value

An object of class scr_workout: rds (one row per default event: identifiers, default_date, ead, product, drivers, status, months_in_default, discount_rate, pv_recovery, pv_cost, pv_drawing, cost_indirect, recovery_extrapolated, lgd_raw, lgd_real, is_cure, is_incomplete, merged_n), recovery_profile (product x month: cum_recovery), extrapolation (per open event), funnel (rule, n, action), summary (n, cure_rate, lra_default_weighted, lra_exposure_weighted, share_incomplete, discount_rate_mean, by_product, by_year), ledger, config, obs_date, and cashflows with keep_rows. rds also carries year, close_date, recovery_nominal, recovery_artificial, cost_nominal, drawing_nominal, closed_at_t_max, absorbed, n_cashflows and last_month; funnel has kept; summary has n_cure, lra_raw, ead_total and years.

Rules

- **Cures.** An event with status == "cured" returns to performing: the balance outstanding at the cure date (ead plus the drawings after default, net of the cash recovered) enters as an artificial recovery on the cure date, so the cure carries its costs and the discount effect, never a zero loss by decree.
- **Multiple defaults.** Two defaults of one facility separated by fewer than lgd_cure_window months (from the close of the first to the start of the second), or a new default while the first is still open, are one event: the first default date and exposure are kept, the cash flows of both spells are pooled and the status of the last spell rules.
- **Incomplete workouts.** An open event younger than lgd_t_max months receives the expected further recovery read from the recovery profile of the closed defaults of the same product (cumulative discounted recovery rate by month in default); an open event at or beyond lgd_t_max is treated as closed with no further recovery.
- **Bounds.** With lgd_floor_at_zero the realized LGD used in the averages is floored at zero and with lgd_cap_at_one capped at one; lgd_raw always keeps the unbounded value.

The long-run average is reported default-weighted (the arithmetic mean over events) and exposure-weighted, overall, by product and by calendar year of default.

See Also

Other irb-lgd: scr_elbe(), scr_lgd(), scr_lgd_downturn(), scr_lgd_floor(), scr_lgd_pools(), scr_lgd_validate()

Examples

```
cfg <- scr_config(verbose = FALSE)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
wo
wo$funnel
head(wo$rds[, c("default_id", "product", "status", "lgd_raw", "lgd_real", "is_cure")])
```

Index

∗ **accessors**
scr_funnel, 69
scr_gains, 70
scr_leakage, 77
scr_result, 122
scr_score_gains, 129
scr_score_metrics, 130
scr_selected, 139
∗ **classing**
scr_classing_accept, 27
scr_classing_apply, 29
scr_classing_choose, 30
scr_classing_propose, 31
scr_classing_spec, 32
scr_classing_view, 33
scr_coarse_classing, 34
scr_decisions, 42
∗ **configuration**
scr_config, 37
scr_config_keys, 39
scr_presets, 112
scr_verbose, 156
∗ **database**
scr_connect, 40
scr_fetch, 68
∗ **datasets**
scr_demo, 46
scr_demo_ead, 47
scr_demo_lgd, 48
scr_demo_lgd_cashflows, 49
scr_demo_panel, 50
scr_demo_portfolio, 51
scr_demo_rates, 52
∗ **data**
scr_demo, 46
scr_demo_ead, 47
scr_demo_lgd, 48
scr_demo_lgd_cashflows, 49
scr_demo_panel, 50
scr_demo_portfolio, 51
scr_demo_rates, 52
∗ **irb-capital**
scr_capital, 21
scr_ecl, 61
scr_el, 63
scr_irb_rw, 74
scr_pd_stress, 109
scr_sa_rw, 125
∗ **irb-ead**
scr_bin_continuous, 17
scr_ead, 55
scr_ead_data, 57
scr_ead_downturn, 59
scr_ead_validate, 60
∗ **irb-lgd**
scr_elbe, 64
scr_lgd, 78
scr_lgd_downturn, 80
scr_lgd_floor, 81
scr_lgd_pools, 82
scr_lgd_validate, 83
scr_workout, 157
∗ **irb-parameters**
scr_default, 43
scr_default_rate, 45
scr_irb_params, 73
∗ **irb-pd**
predict.scr_grades, 5
predict.scr_pd, 6
scr_calibrate, 19
scr_grades, 71
scr_master_scale, 84
scr_migration, 90
scr_moc, 94
scr_pd, 106
scr_pd_pit_ttc, 108
scr_pd_validate, 110
∗ **metrics**

scr_iv, 76
scr_metrics, 88
scr_psi, 112
∗ **portfolio**
scr_compare, 36
scr_core, 41
scr_run, 123
scr_runset, 124
∗ **production**
predict.scr_align, 4
scr_apply, 9
scr_export, 65
scr_monitor, 96
scr_monitoring_plan, 98
scr_reasons, 120
scr_sql, 141
∗ **score-studies**
scr_bands, 11
scr_claims, 23
scr_detection, 53
scr_maturity, 85
scr_mix_shift, 91
scr_operating, 98
scr_overlap, 103
scr_rag, 114
scr_rag_plan, 118
scr_score_cross, 126
scr_segments, 133
scr_tiers, 147
scr_uplift, 153
∗ **stages**
scr_align, 7
scr_bin, 16
scr_cutoff, 41
scr_model, 95
scr_reject, 121
scr_scorecard, 131
scr_select, 137
scr_split, 140
scr_strategy, 144
scr_triage, 152

as.data.frame.scr_result *(*scr_result*)*, 122
as.Date(), *103*

DBI::dbConnect(), *40*
DBI::dbDisconnect(), *40*

message(), *156*

odbc::odbc(), *40*
OptimalBinningWoE::obwoe_algorithms(), *38*
OptimalBinningWoE::obwoe_sql(), *143*

plot.scr_result *(*scr_result*)*, 122
predict.scr_align, 4
predict.scr_align(), *8*, *11*, *67*, *97*, *98*, *120*, *144*
predict.scr_grades, 5
predict.scr_grades(), *7*, *20*, *72*, *85*, *90*, *95*, *107*, *108*, *111*
predict.scr_pd, 6
predict.scr_pd(), *6*, *20*, *72*, *85*, *90*, *95*, *107*, *108*, *111*
print.scr_result *(*scr_result*)*, 122
print.scr_runset *(*scr_runset*)*, 124

scr_align, 7
scr_align(), *5*, *17*, *19*, *42*, *96*, *121*, *131–133*, *138*, *141*, *146*, *152*
scr_apply, 9
scr_apply(), *5*, *6*, *14*, *15*, *17*, *22*, *34*, *56*, *67*, *97*, *98*, *120*, *121*, *132*, *135*, *143*, *144*, *150*, *151*
scr_bands, 11
scr_bands(), *10*, *11*, *24–27*, *55*, *66*, *67*, *86*, *87*, *92–94*, *100*, *101*, *103*, *105*, *106*, *115–117*, *119*, *127–129*, *135*, *137*, *142*, *144*, *150*, *151*, *153–155*
scr_bin, 16
scr_bin(), *9*, *42*, *96*, *121*, *133*, *137*, *138*, *141*, *146*, *152*
scr_bin_continuous, 17
scr_bin_continuous(), *55*, *56*, *58*, *59*, *61*, *78*
scr_calibrate, 19
scr_calibrate(), *6*, *7*, *72*, *85*, *90*, *95*, *107*, *108*, *111*
scr_capital, 21
scr_capital(), *11*, *51*, *63*, *64*, *66*, *76*, *109*, *126*, *142*
scr_claims, 23
scr_claims(), *15*, *55*, *66*, *67*, *87*, *94*, *103*, *106*, *117*, *119*, *129*, *137*, *151*, *155*
scr_classing_accept, 27
scr_classing_accept(), *29–35*, *43*
scr_classing_apply, 29

scr_classing_apply(), *28*, *30*, *32–35*, *43*, *139*
scr_classing_choose, 30
scr_classing_choose(), *28*, *29*, *32–35*, *43*, *70*
scr_classing_discard *(*scr_classing_accept*)*, 27
scr_classing_discard(), *34*
scr_classing_import *(*scr_classing_spec*)*, 32
scr_classing_import(), *32*
scr_classing_propose, 31
scr_classing_propose(), *28–30*, *33–35*, *43*
scr_classing_read *(*scr_classing_spec*)*, 32
scr_classing_read(), *32*
scr_classing_spec, 32
scr_classing_spec(), *28–30*, *32*, *34*, *35*, *43*
scr_classing_view, 33
scr_classing_view(), *28–30*, *32–35*, *43*
scr_coarse_classing, 34
scr_coarse_classing(), *28–34*, *43*, *66*
scr_compare, 36
scr_compare(), *41*, *124*, *125*
scr_config, 37
scr_config(), *16*, *22*, *39*, *44*, *46*, *56*, *58*, *62*, *79*, *96*, *98*, *112*, *123*, *132*, *138*, *140*, *152*, *156*, *157*
scr_config_keys, 39
scr_config_keys(), *37*, *38*, *112*, *156*
scr_connect, 40
scr_connect(), *68*, *69*, *123*
scr_core, 41
scr_core(), *36*, *124*, *125*
scr_cutoff, 41
scr_cutoff(), *9*, *14*, *17*, *96*, *101*, *103*, *105*, *121*, *133*, *138*, *141*, *145*, *146*, *150*, *152*
scr_decisions, 42
scr_decisions(), *28–30*, *32–35*
scr_default, 43
scr_default(), *45*, *46*, *50*, *74*, *87*
scr_default_rate, 45
scr_default_rate(), *19*, *44*, *50*, *72*, *74*, *87*, *110*
scr_demo, 46, *48–53*
scr_demo_ead, 47, *47*, *49–53*
scr_demo_lgd, *47*, 48, *48–53*
scr_demo_lgd_cashflows, *47*, *48*, 49, *49*, *51–53*
scr_demo_panel, *47–49*, 50, *50*, *52*, *53*, *86*
scr_demo_portfolio, *47–50*, 51, *51*, *53*
scr_demo_rates, *47–51*, 52, *52*
scr_detection, 53
scr_detection(), *15*, *27*, *66*, *67*, *87*, *94*, *103*, *106*, *117*, *119*, *129*, *137*, *151*, *155*
scr_ead, 55
scr_ead(), *18*, *47*, *58–61*, *66*, *142*
scr_ead_data, 57
scr_ead_data(), *18*, *47*, *56*, *59–61*
scr_ead_downturn, 59
scr_ead_downturn(), *18*, *56*, *58*, *61*
scr_ead_validate, 60
scr_ead_validate(), *18*, *47*, *56*, *58*, *59*, *66*
scr_ecl, 61
scr_ecl(), *23*, *51*, *64*, *76*, *109*, *126*
scr_el, 63
scr_el(), *23*, *63*, *76*, *109*, *126*
scr_elbe, 64
scr_elbe(), *67*, *79*, *80*, *82*, *84*, *158*
scr_export, 65
scr_export(), *5*, *11*, *22*, *29*, *34*, *37*, *97*, *98*, *107*, *120*, *138*, *144*
scr_fetch, 68
scr_fetch(), *40*
scr_funnel, 69
scr_funnel(), *71*, *78*, *122*, *130*, *131*, *138*, *139*
scr_gains, 70
scr_gains(), *70*, *78*, *122*, *130*, *131*, *138*, *139*
scr_grades, 71
scr_grades(), *5–7*, *20*, *72*, *85*, *90*, *94*, *95*, *106–108*, *111*, *147*
scr_irb_params, 73
scr_irb_params(), *22*, *44*, *46*, *56*, *75*, *81*, *107*, *126*
scr_irb_rw, 74
scr_irb_rw(), *21–23*, *63*, *64*, *109*, *126*
scr_iv, 76
scr_iv(), *89*, *113*
scr_leakage, 77
scr_leakage(), *70*, *71*, *122*, *130*, *131*, *138*, *139*
scr_lgd, 78
scr_lgd(), *48*, *64*, *66*, *80–84*, *142*, *158*
scr_lgd_downturn, 80
scr_lgd_downturn(), *64*, *79*, *82*, *84*, *158*

scr_lgd_floor, 81
scr_lgd_floor(), *64*, *79, 80*, *82*, *84*, *158*
scr_lgd_pools, 82
scr_lgd_pools(), *64*, *79, 80*, *82*, *84*, *158*
scr_lgd_validate, 83
scr_lgd_validate(), *64*, *66*, *79, 80*, *82*, *158*
scr_master_scale, 84
scr_master_scale(), *6, 7*, *20*, *71, 72*, *90*, *95*, *107, 108*, *111*
scr_maturity, 85
scr_maturity(), *15*, *27*, *55*, *66, 67*, *94*, *103*, *106*, *117*, *119*, *129*, *137*, *151*, *155*
scr_metrics, 88
scr_metrics(), *77*, *111*, *113*
scr_migration, 90
scr_migration(), *6, 7*, *20*, *72*, *85*, *95*, *107, 108*, *111*
scr_mix_shift, 91
scr_mix_shift(), *15*, *27*, *55*, *66, 67*, *87*, *103*, *106*, *117*, *119*, *129*, *137*, *151*, *155*
scr_moc, 94
scr_moc(), *6, 7*, *20*, *72*, *85*, *90*, *107, 108*, *111*
scr_model, 95
scr_model(), *9*, *17*, *42*, *121*, *133*, *137, 138*, *141*, *146*, *152*
scr_monitor, 96
scr_monitor(), *5*, *11*, *67*, *98*, *120*, *144*
scr_monitoring_plan, 98
scr_monitoring_plan(), *5*, *11*, *67*, *97*, *120*, *132*, *144*
scr_operating, 98
scr_operating(), *15*, *27*, *55*, *66, 67*, *87*, *94*, *106*, *117*, *119*, *129*, *137*, *151*, *155*
scr_overlap, 103
scr_overlap(), *15*, *27*, *55*, *66, 67*, *87*, *94*, *103*, *117*, *119*, *129*, *137*, *151*, *155*
scr_pd, 106
scr_pd(), *6, 7*, *20*, *66*, *72*, *85*, *90*, *95*, *108*, *110, 111*, *142*
scr_pd_pit_ttc, 108
scr_pd_pit_ttc(), *6, 7*, *20*, *72*, *85*, *90*, *95*, *106, 107*, *109*, *111*
scr_pd_stress, 109
scr_pd_stress(), *22, 23*, *63, 64*, *76*, *108*, *126*
scr_pd_validate, 110
scr_pd_validate(), *6, 7*, *20*, *66*, *72*, *85*, *90*, *95*, *107, 108*
scr_presets, 112
scr_presets(), *37–39*, *156*
scr_psi, 112
scr_psi(), *15*, *60*, *77*, *89*, *93*, *111*, *135*, *155*
scr_rag, 114
scr_rag(), *15*, *27*, *55*, *66, 67*, *87*, *93, 94*, *103*, *106*, *118, 119*, *129*, *137*, *151*, *155*
scr_rag_plan, 118
scr_rag_plan(), *15*, *27*, *55*, *87*, *94*, *103*, *106*, *115*, *117*, *129*, *137*, *151*, *155*
scr_reasons, 120
scr_reasons(), *5*, *11*, *67*, *97, 98*, *144*
scr_reject, 121
scr_reject(), *9*, *17*, *42*, *96*, *133*, *138*, *141*, *146*, *152*
scr_result, *70, 71*, *78*, 122, *130, 131*, *139*
scr_run, 123
scr_run(), *36*, *41*, *124, 125*, *138*
scr_runset, *36*, *41*, 124, *124*
scr_sa_rw, 125
scr_sa_rw(), *23*, *63, 64*, *76*, *109*
scr_score_cross, 126
scr_score_cross(), *15*, *27*, *55*, *66, 67*, *87*, *94*, *103*, *106*, *117*, *119*, *137*, *151*, *155*
scr_score_gains, 129
scr_score_gains(), *14, 15*, *70, 71*, *78*, *122*, *131*, *139*, *146*
scr_score_metrics, 130
scr_score_metrics(), *70, 71*, *78*, *122*, *130*, *139*
scr_scorecard, 131
scr_scorecard(), *7*, *9, 10*, *13*, *17*, *19*, *24*, *29*, *34*, *42*, *66*, *72*, *92*, *96–98*, *100*, *115*, *120, 121*, *130, 131*, *134*, *138, 139*, *141, 142*, *145, 146*, *148*, *152*
scr_segments, 133
scr_segments(), *15*, *27*, *55*, *66, 67*, *87*, *93, 94*, *103*, *106*, *117*, *119*, *129*, *151*, *155*
scr_select, 137
scr_select(), *9, 10*, *17*, *34, 35*, *38*, *42*, *66*, *69, 70*, *78*, *96*, *121–123*, *132, 133*, *139–142*, *146*, *152*
scr_selected, 139
scr_selected(), *70, 71*, *78*, *122*, *130–132*, *138*
scr_split, 140
scr_split(), *9*, *17*, *37, 38*, *42*, *96*, *121*, *133*, *137, 138*, *146*, *152*
scr_sql, 141

scr_sql(), *5*, *11*, *14, 15*, *17*, *34*, *67*, *97, 98*, *120*, *132*, *138, 139*, *150, 151*
scr_strategy, 144
scr_strategy(), *9*, *15*, *17*, *42*, *96*, *102, 103*, *121*, *130*, *133*, *138*, *141*, *152*
scr_tiers, 147
scr_tiers(), *10, 11*, *15*, *24*, *26, 27*, *55*, *66, 67*, *86, 87*, *92–94*, *103*, *106*, *115*, *117*, *119*, *127–129*, *137*, *142*, *144*, *153*, *155*
scr_triage, 152
scr_triage(), *9*, *16–18*, *42*, *96*, *121*, *133*, *137, 138*, *141*, *146*
scr_uplift, 153
scr_uplift(), *15*, *27*, *55*, *66, 67*, *87*, *94*, *103*, *106*, *117*, *119*, *129*, *137*, *151*
scr_verbose, 156
scr_verbose(), *38, 39*, *68*, *112*
scr_workout, 157
scr_workout(), *48*, *52*, *64*, *78–80*, *82–84*
stats::quantile(), *103*
summary(), *138*
summary.scr_result *(*scr_result*)*, 122
suppressMessages(), *156*

tabulate(), *77*