Package: hpsreg (via r-universe)

October 20, 2026

Type Package

Title Regression for Hierarchically Ordered Clinical Outcomes via Pairwise Comparisons (Bradley-Terry-Davidson Model)

Version 1.0.0

Description Provides regression for hierarchically ordered clinical outcomes via pairwise comparisons, using a covariate-adjusted Davidson (Bradley-Terry with ties) model with U-statistic sandwich standard errors. A hierarchy of outcomes (binary, ordinal, continuous, and time-to-event, with eligibility guards and tie margins) is defined; all pairs of subjects are scored lexicographically; and the resulting win / loss / tie data are analysed. Unadjusted win statistics (net benefit, win ratio, win odds), diagnostics, and a worked example based on outcomes after allogeneic stem cell transplant are also provided. The computational engine is written in 'C++' using 'Rcpp' and 'RcppEigen'.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.0.0)

Imports Rcpp (>= 1.0.0), stats, graphics

LinkingTo Rcpp, RcppEigen

Suggests knitr, rmarkdown, testthat (>= 3.0.0), survival, gnm

VignetteBuilder knitr

SystemRequirements OpenMP (optional)

Config/testthat/edition 3

RoxygenNote 7.3.3

NeedsCompilation yes

Author Ravi Varadhan [aut, cre]

Maintainer Ravi Varadhan <ravi.varadhan@jhu.edu>

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

Date/Publication 2026-10-20 17:50:14 UTC

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

RemoteRef HEAD

RemoteSha 5d1a6e325d2ba1b25c1a3f96299fd002ff60dc3d

Contents

btd 2
btd_calibration 5
btd_control 7
hb_levels 7
hierarchy 9
hps_beta 10
hps_cox_grfs 11
hps_hierarchy 11
lvl_group 12
openmp_threads 12
plot.btd_calibration 13
predict.btd 14
resolution 14
sample_pairs 15
score_pairs 16
simulate_hps 16
win_stats 17

Index **19**

`btd` *Covariate-adjusted analysis of hierarchical pairwise outcomes*

Description

Fits the Bradley-Terry-Davidson model to the win / loss / tie outcomes produced by `score_pairs()`. For a pair of subjects with covariate difference $x_i - x_j$ and $d = (x_i - x_j)^\top \beta/2$,

$$P(i \text{ wins}) = e^d/q, \quad P(j \text{ wins}) = e^{-d}/q, \quad P(\text{tie}) = \delta/q,$$

with $q = e^d + e^{-d} + \delta$. A coefficient is the log of the multiplicative effect of a one-unit increase of the covariate on a subject's "worth"; positive values mean a better position in the hierarchy. If no pair is tied the model reduces to Bradley-Terry ($\delta = 0$).

Usage

```
btd(
  formula,
  data,
  pairs = NULL,
  hierarchy = NULL,
  scale = TRUE,
  engine = c("cpp", "r"),
  storage = c("auto", "stored", "compact", "stream"),
  control = btd_control()
)

## S3 method for class 'btd'
summary(object, exponentiate = FALSE, ...)
```

Arguments

`formula` One-sided formula (`~ x1 + x2`) for the covariates. The intercept is not identifiable and is dropped.

`data` Data frame with one row per subject, the same rows used in `score_pairs()`.

`pairs` An `hb_pairs` object from `score_pairs()`. If `NULL`, supply `hierarchy` and the pairs are scored first.

`hierarchy` A `hierarchy()`, used only if `pairs` is `NULL`.

`scale` Standardise covariates internally (recommended).

`engine` `"cpp"` (default) or `"r"` (slow reference implementation).

`storage` How pairs are handled when `hierarchy` is given.

- `"stored"`: score and keep all pairs (about 20 bytes per pair, more at peak), available afterwards as `$pairs`.
- `"compact"`: score every pair once and keep one byte per pair (outcome and resolving level; indices are implicit). Fit and variance passes read this array instead of re-scoring. About 50 MB at N = 10^4.
- `"stream"`: never store anything proportional to the number of pairs; pairs are re-scored in every pass. Memory is O(N p), at the cost of extra compute.
- `"auto"` (default): `"stored"` up to `getOption("hpsreg.stream_min_pairs", 2.5e6)` pairs (about N = 2200), then `"compact"` up to `getOption("hpsreg.compact_max_pairs", 1e9)` pairs (about N = 44,000), then `"stream"`; for hierarchies with `lvl_custom()` levels, always `"stored"`.

`"compact"` and `"stream"` require `engine = "cpp"` and a `hierarchy` (not `pairs`), support at most 84 comparisons for `"compact"`, and give a fit without a `$pairs` component. Estimates and standard errors agree with the stored engine to numerical precision, and `"compact"` and `"stream"` agree bitwise.

`control` See `btd_control()`.

`object` A fitted `btd` object.

exponentiate	If TRUE, the Estimate column reports exp(beta) (the preference odds ratio for covariate terms, and delta for the log_delta row). Std. Error, z value and Pr(>|z|) remain on the log scale.
...	Unused.

Details

Estimation is by Levenberg-Marquardt on standardised covariates (C++ engine, or a pure R reference implementation with engine = "r"). Standard errors are sandwich (robust) estimates based on the projection of the pairwise estimating equation onto subjects, which accounts for the dependence between pairs that share a subject (Hoeffding / U-statistic variance). They assume the complete set of N(N-1)/2 pairs; for a sampled or user-supplied subset the normalisation uses the average number of pairs per subject and should be treated as approximate. The model-based inverse information, which ignores that dependence, is returned as vcov_model for reference only.

Value

An object of class btd with coefficients, log_delta, delta, vcov (sandwich, on the original covariate scale, including log_delta), vcov_model, loglik, convergence information, the latent scores eta and the pairs used.

The object supports the usual generics: summary() (coefficient table with z-tests), coef(), vcov() (model = TRUE for the model-based information), confint() (Wald intervals), logLik(), nobs() (number of pairs; the subject count is fit$n_subjects), print(), and predict() (latent score for the training rows, or for a newdata frame).

Pair-level predicted probabilities are provided by btd_pair_probs(); calibration tables by btd_calibration().

Two workflows

btd() needs the scored pairs of a hierarchy. You can supply them or let btd() compute them:

1. **Pre-computed pairs** (pairs = pw). Score once with score_pairs(), inspect with resolution() or win_stats(), then fit several covariate models on the same pairs without rescoring.
2. **Inline scoring** (hierarchy = h). btd() calls score_pairs() for you; the pairs are kept in the fitted object (fit$pairs) for later use.

In both cases data must be the same data frame (same rows, same order) that is scored.

References

Varadhan R. "Hierarchical Pairwise Scoring: A Regression Framework for Composite Outcomes, with Application to Hematopoietic Stem Cell Transplantation." Manuscript in preparation, 2026.

Examples

```
## From scratch: define a hierarchy from data-frame columns, score, fit.
dat <- simulate_hps(N = 150, seed = 11)

h <- hierarchy(
  lvl_tte("death_day", better = "later", name = "survival"),
```

```
  lvl_tte("relapse_day", better = "later", name = "relapse"),
  lvl_ordinal("gvhd_grade", better = "lower", name = "GVHD grade"),
  lvl_binary("gvhd_resolved", favorable = 1, when = "gvhd_grade > 0",
             name = "GVHD resolution")
)

## Workflow 1: pre-computed pairs (reusable across models)
pw <- score_pairs(h, dat)
resolution(pw)
fit1 <- btd(~ donor_type + age + dri + hct_ci, data = dat, pairs = pw)
summary(fit1)
fit2 <- btd(~ donor_type + age, data = dat, pairs = pw)   # no rescoring

## Workflow 2: let btd() score the pairs
fit3 <- btd(~ donor_type + age, data = dat, hierarchy = h)
all.equal(coef(fit2), coef(fit3))

## The packaged transplant hierarchy is a shortcut for the one above
fit4 <- btd(~ donor_type + age, data = dat, hierarchy = hps_hierarchy())

## How many threads can this build use? 0 means no OpenMP, in which
## case the engine runs serially regardless of the `threads` argument.
openmp_threads()


## Multiple threads: results are bitwise identical to a single-threaded
## fit, so this only changes the runtime, not the estimates.
n <- max(1L, min(4L, openmp_threads()))
fit_mt <- btd(~ donor_type + age, data = dat, hierarchy = h,
              control = btd_control(threads = n))
all.equal(coef(fit_mt), coef(fit3))

## The same setting can be supplied through an option, which is
## convenient when many calls are made in a script.
op <- options(hpsreg.threads = n)
fit_mt2 <- btd(~ donor_type + age, data = dat, hierarchy = h)
options(op)
all.equal(coef(fit_mt), coef(fit_mt2))
```

`btd_calibration` *Calibration of the pairwise model*

Description

Compares observed outcomes of the scored pairs with the model's predicted probabilities.

Usage

```
btd_calibration(object, by = c("level", "bin"), n_bins = 10L)
```

Arguments

`object`	A `btd` fit.
`by`	`"level"` or `"bin"`.
`n_bins`	Number of bins when `by = "bin"`.

Details

- `by = "level"`: among pairs *decided* at each hierarchy level, the observed share won by the first-listed subject versus the model's $P(i)/(P(i) + P(j))$. Systematic differences between levels suggest that a single latent scale does not describe all levels.
- `by = "bin"`: pairs are grouped by quantiles of the predicted latent difference; observed and predicted win / tie / loss fractions are compared. For fits made with streaming (no stored pairs) the bins are approximate quantile bins built from a fine histogram of the latent difference.

The returned object is plotted by `plot.btd_calibration`, which draws a dot plot for `by = "level"` and a three-panel observed-vs-predicted line plot for `by = "bin"`.

Value

A `btd_calibration` object (a data frame with additional attributes) with columns

- `by = "level"`: `level`, `n`, `obs_win_i`, `pred_win_i`.
- `by = "bin"`: `bin`, `n`, `obs_win_i`, `pred_win_i`, `obs_tie`, `pred_tie`, `obs_win_j`, `pred_win_j`.

See Also

`plot.btd_calibration` for the graphical counterpart; `btd()` to fit the model; `score_pairs()` to score the pairs.

Examples

```
dat <- simulate_hps(N = 120, seed = 5)
h   <- hps_hierarchy()
fit <- btd(~ donor_type + dri, dat, hierarchy = h)

## Calibration by hierarchy level: for each level, the observed and
## predicted share of pairs won by the first subject.
cal_level <- btd_calibration(fit, by = "level")
cal_level
plot(cal_level)

## Calibration by quantile bins of the latent difference d:
## observed vs predicted win / tie / loss fractions.
cal_bin <- btd_calibration(fit, by = "bin", n_bins = 8)
cal_bin
plot(cal_bin)
```

btd_control — *Control parameters for the Bradley-Terry-Davidson fit*

Description

Control parameters for the Bradley-Terry-Davidson fit

Usage

```
btd_control(
  max_iter = 200L,
  tol = 1e-06,
  lambda0 = 1e-06,
  verbose = FALSE,
  threads = NULL
)
```

Arguments

`max_iter`	Maximum Levenberg-Marquardt iterations.
`tol`	Convergence tolerance on the scaled gradient, `max|g| / (1 + |loglik|)`.
`lambda0`	Initial damping.
`verbose`	Print iteration progress (C++ engine only).
`threads`	Number of OpenMP threads for the streaming engine. `NULL` (default) uses `getOption("hpsreg.threads", 1)`. Ignored if the package was built without OpenMP (see `openmp_threads()`). Results do not depend on it.

Value

A list.

hb_levels — *Outcome levels for a hierarchy*

Description

Constructors for the building blocks of a `hierarchy()`. Each level compares two subjects on one outcome and returns "first wins", "second wins" or "neutral" (the pair is passed to the next level).

Usage

```
lvl_binary(var, favorable = 1, when = NULL, name = var, na = "tie")

lvl_ordinal(
  var,
  better,
  order = NULL,
  threshold = 0,
  when = NULL,
  name = var,
  na = "tie"
)

lvl_continuous(var, better, threshold = 0, when = NULL, name = var, na = "tie")

lvl_tte(
  time,
  status = NULL,
  better,
  horizon = Inf,
  threshold = 0,
  when = NULL,
  name = time
)

lvl_custom(fun, name = "custom", when = NULL)
```

Arguments

`var`	Name of the outcome column in the data.
`favorable`	Value of `var` that is the favourable outcome (binary levels). Logical, numeric, character and factor columns are supported.
`when`	Optional eligibility guard, an expression, formula or string.
`name`	Label used in printed output and resolution tables.
`na`	How to treat a missing value in exactly one subject (see below).
`better`	Direction that is favourable: `"higher"` or `"lower"` (ordinal, continuous); `"later"` or `"earlier"` (time-to-event). Must be supplied.
`order`	Optional character vector giving the levels of an ordinal variable from lowest to highest. Defaults to the factor levels (or the sorted unique values).
`threshold`	Tie margin (>= 0): differences up to this size are treated as ties (clinically meaningful difference).
`time`, `status`	Names of the time and event-status columns (`lvl_tte()`); see Details.
`horizon`	Administrative horizon for time-to-event levels.
`fun`	For `lvl_custom()`: a function `function(data, i, j)` returning, for the pairs `(i[k], j[k])`, a vector in `{-1, 0, 1}` (1 = subject `i` wins) or `NA` for an uninformative pair.

Value

An object of class `hb_level`.

Eligibility guards

`when` is an expression evaluated in the data (e.g. `when = gvhd_grade > 0`). Inside package code, where an unquoted column name would trigger an "undefined global variable" note in `R CMD check`, pass it as a string (`when = "gvhd_grade > 0"`); both forms are equivalent. If either subject of a pair is ineligible (guard `FALSE` or `NA`) the level is neutral for that pair. This expresses conditional comparisons such as "compare resolution of GVHD only for patients who had GVHD".

Missing values

For `lvl_binary()`, `lvl_ordinal()` and `lvl_continuous()` the argument `na` controls pairs in which exactly one subject has a missing value: `"tie"` (default) makes the level neutral, `"best"` / `"worst"` treat a missing value as the best / worst possible outcome. Pairs in which both are missing are always neutral.

Time-to-event levels

With `status = NULL` the time variable is read as "time of the event, `NA` if the event did not occur" and follow-up is assumed complete (no censoring); non-events are placed at `horizon` (or infinity). With a `status` column (1 = event, 0 = censored) pairs are compared with the Gehan rule: a pair is decided only if the ordering is certain under right censoring, otherwise it is *uninformative* and passed to the next level (and flagged, see `score_pairs()`). Follow-up beyond `horizon` is truncated and treated as complete at `horizon`. This rule is simple and biased toward the null under heavy censoring; inverse-probability-of-censoring scoring is not yet implemented.

Examples

```
lvl_tte("death_day", better = "later", name = "survival")
lvl_ordinal("gvhd_grade", better = "lower")
lvl_binary("resolved", favorable = 1, when = gvhd_grade > 0)
lvl_continuous("pain_score", better = "lower", threshold = 1)
```

`hierarchy` *Define a hierarchy of outcomes*

Description

Levels are listed from highest to lowest priority. When two subjects are compared, the first level on which they differ decides who wins; if no level separates them the pair is a tie.

Usage

```
hierarchy(...)
```

Arguments

...	Levels (`lvl_*()`) and/or groups (`lvl_group()`).

Value

An object of class `hb_hierarchy`.

Examples

```
h <- hierarchy(
  lvl_tte("death_day", better = "later", name = "survival"),
  lvl_group("GVHD",
    lvl_ordinal("gvhd_grade", better = "lower"),
    lvl_binary("gvhd_resolved", favorable = 1, when = gvhd_grade > 0)
  )
)
h
```

---

hps_beta — *Data-generating coefficients for the transplant example*

Description

Starting values for `simulate_hps()`: the negative log hazard ratios of published m-GRFS effects (uncalibrated), with fixed age and comorbidity effects. Use these as a convenient default; effects should be re-calibrated to a target if exact hazard ratios matter.

Usage

```
hps_beta()
```

Value

A named numeric vector.

Examples

```
hps_beta()
```

hps_cox_grfs — *Composite m-GRFS Cox model (comparator for the example)*

Description

Fits a Cox model for modified GVHD- and relapse-free survival: the first of death, relapse, grade >= 2 GVHD or grade >= 2 infection within the horizon. This is the conventional composite analysis that the hierarchical analysis is compared with in the example. Requires the **survival** package.

Usage

```
hps_cox_grfs(dat, covars, horizon = 180)
```

Arguments

dat	Data from simulate_hps().
covars	Character vector of covariate names.
horizon	Follow-up horizon in days.

Value

A list with fit, coef and se.

Examples

```
dat <- simulate_hps(N = 200, seed = 4)
hps_cox_grfs(dat, c("donor_type", "age"))$coef
```

hps_hierarchy — *Hierarchy for the stem cell transplant example*

Description

Four levels, evaluated in this order: survival (later death better, non-death best), relapse (later relapse better, non-relapse best), GVHD (lower grade; then resolution; then later onset) and infection (lower grade; then resolution). Resolution and onset are only compared between patients who had the event.

Usage

```
hps_hierarchy()
```

Value

A hierarchy().

Examples

```
hps_hierarchy()
```

---

lvl_group — *Group several levels into one reporting level*

Description

A group is evaluated as consecutive levels (in the order given) but is reported as a single level, e.g. GVHD grade, then resolution, then onset time are three comparisons forming one "GVHD" level of the hierarchy.

Usage

```
lvl_group(name, ..., when = NULL)
```

Arguments

`name`	Label of the group.
`...`	Levels created by the `lvl_*()` constructors.
`when`	Optional eligibility guard applied to every level in the group.

Value

An object of class `hb_group`.

---

openmp_threads — *OpenMP availability*

Description

Number of threads OpenMP can use in this build of the package, or `0` if the package was compiled without OpenMP. The streaming engine uses one thread by default; set `options(hpsreg.threads = n)` (or `btd_control(threads = n)`) to use more. Results are bitwise identical for any number of threads.

Usage

```
openmp_threads()
```

Value

An integer.

Examples

```
openmp_threads()
```

plot.btd_calibration *Plot a calibration table*

Description

Two graphical summaries are produced depending on the by argument of `btd_calibration()`:

Usage

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

Arguments

`x`	A `btd_calibration` object from `btd_calibration()`.
`...`	Additional graphical parameters passed to the main `plot()` call of each panel. User-supplied `main`, `xlab`, `ylab` and `ylim` override the defaults.

Details

- by = `"level"`: a dot plot with one row per hierarchy level, showing the observed and predicted win share for pairs resolved at that level. Lines of agreement between the two are what a well-calibrated hierarchy should look like.
- by = `"bin"`: three line plots (win, tie, loss) of observed against predicted probabilities across quantile bins of the latent difference `d`.

The bin plot shows observed and predicted probabilities only; no binomial error bars are drawn. Pairs share subjects, so treating them as independent Bernoulli draws would understate the sampling variability. For a variability estimate use the subject-clustered standard errors from `btd()` rather than a pair-level binomial interval.

Value

`x`, invisibly.

See Also

`btd_calibration()`

Examples

```
dat <- simulate_hps(N = 120, seed = 5)
fit <- btd(~ donor_type + dri, dat, hierarchy = hps_hierarchy())

plot(btd_calibration(fit, by = "level"))
plot(btd_calibration(fit, by = "bin", n_bins = 8))
```

predict.btd — *Predicted latent score and pair probabilities*

Description

`predict()` returns the latent score $(x - \bar{x})^\top \beta$ (centred at the training covariate means; only differences matter). `btd_pair_probs()` returns the model probabilities that subject A wins, that the pair is tied, and that subject B wins, for rows of `a` compared with the same rows of `b`.

Usage

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

btd_pair_probs(object, a, b)
```

Arguments

`object`	A `btd` fit.
`newdata`	Data frame of covariates.
`...`	Unused.
`a`, `b`	Data frames of equal length: first and second subject of each pair.

Value

A numeric vector, or a matrix with columns `win_a`, `tie`, `win_b`.

Examples

```
dat <- simulate_hps(N = 100, seed = 3)
fit <- btd(~ donor_type + age, dat, pairs = score_pairs(hps_hierarchy(), dat))
btd_pair_probs(fit, dat[1:3, ], dat[4:6, ])
```

resolution — *Where are pairs decided?*

Description

Number and share of pairs resolved at each level of the hierarchy, and the share left tied. Pass either scored pairs, or a hierarchy together with the data: in that case the pairs are generated and counted on the fly and are never stored, so this works for very large N.

Usage

```
resolution(pairs, by = c("level", "leaf"), data = NULL)
```

Arguments

`pairs`	An `hb_pairs` object from `score_pairs()`, or a `hierarchy()`.
`by`	`"level"` (top-level levels, default) or `"leaf"` (each comparison inside groups).
`data`	The data frame, when `pairs` is a hierarchy.

Value

A data frame with columns `level`, `n` (pairs decided there) and `prop` (share of all pairs). The last row, `(tied)`, counts pairs that no level separates.

Examples

```
dat <- simulate_hps(N = 80, seed = 2)
resolution(score_pairs(hps_hierarchy(), dat))
resolution(hps_hierarchy(), data = dat)      # streaming, nothing stored
```

`sample_pairs` *Sample a random set of pairs*

Description

Draws `n` distinct pairs uniformly from the N(N-1)/2 possible pairs without enumerating them. Useful when N is too large for all pairs.

Usage

```
sample_pairs(N, n)
```

Arguments

`N`	Number of subjects.
`n`	Number of pairs to draw.

Value

A two-column integer matrix `(i, j)` with `i < j`.

Examples

```
set.seed(1); head(sample_pairs(1000, 5))
```

score_pairs *Score all pairs of subjects under a hierarchy*

Description

Applies a `hierarchy()` to every pair of subjects (or to a supplied set of pairs). Built-in levels are evaluated by a compiled kernel that stops at the first level that separates a pair; `lvl_custom()` levels are evaluated in R on the pairs still unresolved.

Usage

```
score_pairs(hierarchy, data, pairs = NULL, max_pairs = 5e+07)
```

Arguments

`hierarchy`	A `hierarchy()`.
`data`	A data frame with one row per subject.
`pairs`	`NULL` (all N(N-1)/2 pairs, the default) or a two-column integer matrix of row indices `(i, j)`, e.g. from `sample_pairs()`. See the note on variance in `btd()` for non-complete pair sets.
`max_pairs`	Safety limit on the number of pairs held in memory.

Value

An object of class `hb_pairs` with components `id_i`, `id_j` (row indices), `W` (1 if `i` wins, -1 if `j` wins, 0 if tied on every level), `level` (index of the resolving top-level level, `K + 1` if unresolved), `leaf` (finer index when the hierarchy contains groups, else `NULL`), `uninf` (1 if an earlier level was uninformative because of censoring), `N`, `complete` and the `hierarchy`.

Examples

```
dat <- simulate_hps(N = 100, seed = 1)
pw <- score_pairs(hps_hierarchy(), dat)
pw
resolution(pw)
```

simulate_hps *Simulate a transplant cohort*

Description

Simulates covariates and 180-day outcomes (death, relapse, graft-versus-host disease, infection) for the worked example. Outcomes share a latent frailty-like score driven by the covariates in `beta`.

Usage

```
simulate_hps(beta = hps_beta(), N = 613, seed = NULL)
```

Arguments

beta	Named coefficient vector, see hps_beta().
N	Number of patients.
seed	Optional random seed.

Value

A data frame with covariates (donor_type, age, sex, dri, hct_ci, cell_source, conditioning, year, sex_mismatch, diagnosis), the latent score, and outcome columns (death_day, relapse_day, gvhd_grade, gvhd_resolved, gvhd_onset_day, infection_grade, infection_resolved, infection_onset_day). NA in a *_day column means the event did not occur by day 180.

Examples

```
dat <- simulate_hps(N = 200, seed = 1)
head(dat[, c("donor_type", "age", "death_day", "gvhd_grade")])
```

---

win_stats *Win statistics for two groups*

Description

Summarises the cross-group pairs of a scored hierarchy: the proportion of pairs won by each group, and the derived net benefit, win ratio and win odds of treatment versus the reference group, with U-statistic (Hoeffding) variances that account for each subject appearing in many pairs.

Usage

```
win_stats(pairs, group, treatment = NULL, conf.level = 0.95, data = NULL)
```

Arguments

pairs	An hb_pairs object from score_pairs() (complete pair set), or a hierarchy() (then data is required).
group	A vector with exactly two distinct values, one per subject.
treatment	Value of group defining the treatment group; defaults to the second level (for a factor) or the larger value.
conf.level	Confidence level.
data	The data frame, when pairs is a hierarchy.

Details

With p_w and p_l the proportions of cross-group pairs won by the treatment and reference groups and $p_t = 1 - p_w - p_l$: net benefit $p_w - p_l$; win ratio p_w/p_l; win odds $(p_w + p_t/2)/(p_l + p_t/2)$. Confidence intervals for the two ratios are computed on the log scale (delta method).

Requires the complete set of pairs. Pass a hierarchy together with `data` instead of scored pairs to compute everything on the fly without storing pairs (memory O(N)); this is the way to analyse very large N.

Value

An object of class `hb_win` with the summary `table`, the `by_level` contribution of each hierarchy level to wins and losses, and the counts.

Examples

```
dat <- simulate_hps(N = 150, seed = 7)
pw <- score_pairs(hps_hierarchy(), dat)
win_stats(pw, dat$year)
win_stats(hps_hierarchy(), dat$year, data = dat)   # streaming, same result
```

Index

btd, 2
btd(), *6*, *13*, *16*
btd_calibration, 5
btd_calibration(), *4*, *13*
btd_control, 7
btd_control(), *3*
btd_pair_probs *(*predict.btd*)*, 14
btd_pair_probs(), *4*

hb_levels, 7
hierarchy, 9
hierarchy(), *3*, *7*, *11*, *15–17*
hps_beta, 10
hps_beta(), *17*
hps_cox_grfs, 11
hps_hierarchy, 11

lvl_binary *(*hb_levels*)*, 7
lvl_continuous *(*hb_levels*)*, 7
lvl_custom *(*hb_levels*)*, 7
lvl_custom(), *3*
lvl_group, 12
lvl_group(), *10*
lvl_ordinal *(*hb_levels*)*, 7
lvl_tte *(*hb_levels*)*, 7

openmp_threads, 12
openmp_threads(), *7*

plot.btd_calibration, *6*, 13
predict.btd, 14

resolution, 14
resolution(), *4*

sample_pairs, 15
sample_pairs(), *16*
score_pairs, 16
score_pairs(), *2–4*, *6*, *9*, *15*, *17*
simulate_hps, 16
simulate_hps(), *10, 11*
summary.btd *(*btd*)*, 2

win_stats, 17
win_stats(), *4*