Package 'winning'

September 9, 2026

Type Package

Title Contest Win Probabilities and Ability Calibration

Version 0.3.0

Description Solves the horse race problem in both directions. Infers relative ability from win probabilities (or betting dividends) for contests whose entrants share a common performance distribution, with dead heats handled exactly, by the lattice fixed-point algorithm of Cotton (2021) <doi:10.1137/19M1276261>. Also computes all N win probabilities of a factor-structured Gaussian race (multinomial probit with low-rank-plus-diagonal covariance) in one shared-lattice pass of O(Q*N*L) operations, and inverts observed shares to abilities by a damped Newton method with analytic slopes. A dependency-free base-R port of the reference python package 'winning'. Performances are times: lowest wins.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/microprediction/winning

BugReports https://github.com/microprediction/winning/issues

Suggests mvtnorm, testthat (>= 3.0.0)

Config/testthat/edition 3

RoxygenNote 7.3.0

NeedsCompilation no

Author Peter Cotton [aut, cre]

Maintainer Peter Cotton <peter.cotton@microprediction.com>

Repository CRAN

Date/Publication 2026-09-09 16:30:02 UTC

Contents

`abilities_from_probabilities_factor`

Calibrate abilities from observed shares

Description

Inverts the factor-race share map by damped coordinate Newton against the frozen shared field, with analytic slopes and an independent-race warm start. Shares determine abilities uniquely up to a common shift.

Usage

```
abilities_from_probabilities_factor(p, V, D, nodes = NULL,
  n_iter = 50, tol = 1e-6, points = 501)
```

Arguments

`p`	vector of positive target shares (normalized internally).
`V`	N by k matrix of factor loadings.
`D`	vector of idiosyncratic variances.
`nodes`	optional list(F, W) of factor nodes.
`n_iter`	maximum Newton iterations.
`tol`	tolerance on the max log-share residual over identified alternatives.
`points`	lattice size.

Value

Mean-zero ability vector (min-wins convention).

`block_race_probabilities` *Block, nested and tree races*

Description

Clustered covariance grammars of the general race: independent cluster effects (block), block plus one global factor (nested, interpolated by gamma), and a hierarchy of uniform shared effects (tree, two message passes). Across-cluster independence factorizes the survival field, so cost is O(n x lattice x nodes) regardless of the number of blocks. Base-R ports of the python reference, parity-locked.

Usage

```
block_race_probabilities(mu, cluster, loading, D, points = 257, qa = 9)

nested_race_probabilities(mu, cluster, loading, D, coupling = NULL,
  gamma = 1, points = 257, qa = 9, qf = 15)

tree_race_probabilities(mu, cluster, loading, D, parent, strength,
  points = 257, qa = 9)

block_race_jacobian(mu, cluster, loading, D, points = 257, qa = 9)

nested_race_jacobian(mu, cluster, loading, D, coupling = NULL, gamma = 1,
  points = 257, qa = 9, qf = 15)

abilities_from_block_race(p, cluster, loading, D, points = 257, qa = 9,
  tol = 1e-10, max_iter = 25)

tree_race_jacobian(mu, cluster, loading, D, parent, strength,
  points = 257, qa = 9)
```

Arguments

`mu`	abilities (min-wins), length n
`p`	positive target probabilities
`cluster`	cluster labels, length n
`loading`	numeric length n (rank 1) or n x r matrix (r <= 2 in pure R; supply quadrature nodes for higher rank)
`D`	idiosyncratic variances
`coupling`	per-runner loading on the global factor
`gamma`	coupling strength: 0 = independent blocks, 1 = full
`parent`	1-based parent index per tree node; root marked 0 or NA
`strength`	per-node uniform shared-effect strength

points	lattice size
qa, qf	quadrature orders (cluster effect, global factor)
tol	inversion tolerance
max_iter	Newton iterations after the fixed-point globalizer

Value

Probability vectors summing to one; Jacobians as n x n matrices with zero row sums (off-diagonals are symmetric tie densities); abilities_from_block_race a list(mu, residual, iterations).

---

dividend_implied_ability

The horse race problem: ability from win probabilities, and back

---

Description

Given win probabilities for a multi-entrant contest in which every contestant's performance is a translated copy of one density, infer the translations (relative ability), or price the contest in the forward direction, with dead heats handled exactly. Implements the lattice fixed-point algorithm of Cotton (2021), doi:10.1137/19M1276261; this is a base-R port of the reference python package winning.

Usage

```
dividend_implied_ability(dividends, density, nan_value = 2000, unit = 1.0)

state_price_implied_ability(prices, density, unit = 1.0)

ability_implied_state_prices(ability, density, unit = 1.0)

ability_implied_dividends(ability, density, unit = 1.0)

solve_for_implied_offsets(prices, density, offset_samples = NULL,
  implied_offsets_guess = NULL, n_iter = 3)
```

Arguments

dividends	numeric decimal prices (Australian style), NA allowed
prices	numeric state prices (win probabilities), positive
ability	numeric relative abilities; performances are times, so lower is better
density	numeric performance density on the symmetric lattice -L..L (length 2L+1), e.g. from skew_normal_density
nan_value	dividend assigned to NA entries
unit	lattice spacing assumed when the density was constructed

offset_samples	descending offsets for the interpolation table (default: the reference's half-lattice grid)
implied_offsets_guess	starting offsets
n_iter	fixed-point iterations (default 3)

Value

The *_implied_ability functions return numeric abilities (lower is better) in units of unit; ability_implied_state_prices returns state prices; ability_implied_dividends their inverses; solve_for_implied_offsets raw lattice offsets.

Examples

```
d <- skew_normal_density(L = 200, unit = 0.02, a = 1.5)
ability <- dividend_implied_ability(c(2, 6, 6, 20), d)
p <- ability_implied_state_prices(ability, d)
```

---

hermite_nodes *Pruned product Gauss-Hermite nodes*

Description

Golub-Welsch nodes of the probabilists' Hermite rule for expectations over a standard k-variate normal, as a pruned product grid.

Usage

```
hermite_nodes(k, order = 15, prune = 1e-7)
```

Arguments

k	factor dimension.
order	univariate quadrature order.
prune	drop nodes with weight below prune times the maximum.

Value

A list with matrix F (nodes by k) and weight vector W.

`polish_race` *Polish a race onto linear constraints*

Description

Weights ARE race probabilities, and finance imposes linear constraints on them; clipping and renormalizing breaks model-consistency. The right object is the nearest race satisfying the constraints, found by an augmented-Lagrangian on the exact race Jacobian (the python reference uses SLSQP; the two agree to optimizer tolerance).

Usage

```
polish_race(p0 = NULL, mu0 = NULL, V = NULL, D = NULL, F = NULL,
  W = NULL, base = "normal", points = 257, name_caps = NULL,
  groups = NULL, A = NULL, b = NULL, tol = 1e-09, max_iter = 60,
  structure = NULL)

race_jacobian(mu, V = NULL, D = NULL, F = NULL, W = NULL,
  base = "normal", points = 501, structure = NULL, qa = 9, qf = 15)

concentration_matrix(n, name_caps = NULL, groups = NULL)
```

Arguments

`p0`	current weights (inverted to abilities internally)
`mu0`	abilities directly (min-wins, mean-zero)
`mu`	abilities at which to evaluate the Jacobian
`V`, `D`, `F`, `W`, `base`, `points`, `qa`, `qf`	as in `race_probabilities`
`name_caps`	scalar or per-name weight caps (NA skipped)
`groups`	list of list(indices, cap) group caps
`A`, `b`	explicit constraint rows, A p <= b
`tol`	optimizer tolerance
`max_iter`	outer iterations
`structure`	covariance grammar (Tree not yet supported here)
`n`	number of contestants

Value

`polish_race`: list(p, mu, info) – a probability vector that IS the race at mu, satisfying the caps, with mu as close to mu0 as the constraints allow. `race_jacobian`: the exact n x n d p / d mu. `concentration_matrix`: list(A, b).

Examples

```
p0 <- c(0.4, 0.25, 0.2, 0.15)
out <- polish_race(p0 = p0, D = rep(1, 4), name_caps = 0.3)
max(out$p)  # <= 0.3
```

`prices_from_dividends` *Convert between dividends and probabilities*

Description

Naive renormalization between Australian-style dividends (decimal prices) and risk-neutral win probabilities.

Usage

```
prices_from_dividends(dividends, nan_value = 2000)

dividends_from_prices(prices, multiplicity = 1.0)
```

Arguments

`dividends`	numeric decimal prices, `NA` allowed
`prices`	numeric win probabilities
`nan_value`	dividend assigned to `NA` entries
`multiplicity`	dead-heat multiplicity divisor

Value

Numeric vector: normalized probabilities, or dividends.

`race_probabilities` *The general race: one API, distributions and correlation as parameters*

Description

Win probabilities of the general Gaussian (or gumbel, or custom-base) race, all N in one shared-field lattice pass, and the inverse map from probabilities to abilities. With `base = "gumbel"` and D = pi^2/6 the race is exactly softmax. Base-R port of the python reference; parity/vectors.json pins the two to machine precision.

Usage

```
race_probabilities(mu, V = NULL, D = NULL, F = NULL, W = NULL,
  base = "normal", points = 257, return_slopes = FALSE, structure = NULL,
  window = "bulk", delta = 1e-12, qa = 9, qf = 15, nodes = NULL)

abilities_from_race(p, V = NULL, D = NULL, F = NULL, W = NULL,
  base = "normal", points = 257, n_iter = 60, tol = 1e-08,
  structure = NULL, qa = 9, qf = 15)

calibrate_abilities(p, V = NULL, D = NULL, F = NULL, W = NULL,
  base = "normal", points = 257, n_iter = 60, tol = 1e-08,
  structure = NULL, qa = 9, qf = 15)
```

Arguments

mu	numeric abilities; performances are times, so lower is better (min-wins convention)
p	positive target probabilities (normalized internally)
V	optional N x k factor loading matrix
D	idiosyncratic variances (default 1)
F, W	optional factor quadrature nodes and weights, e.g. from hermite_nodes
base	"normal", "gumbel", or a function z -> list(S, f, fp) of a mean-zero unit-variance law
points	lattice size
return_slopes	also return the inversion preconditioner d p_i / d mu_i
structure	a covariance grammar (Independent, Factor, Blocks, Nested, Tree); overrides V/D
window	"bulk" (winner-bulk lattice, default) or "span"
delta	omitted winner mass bound for the bulk window
qa, qf	quadrature orders used by structure dispatch
nodes	deprecated alias for list(F, W)
n_iter	maximum damped-Newton iterations
tol	convergence tolerance on max |log p - log target|

Value

race_probabilities: probabilities summing to one (or list(p, slopes)). abilities_from_race and its alias calibrate_abilities: the mean-zero ability vector reproducing p.

Examples

```
p <- race_probabilities(c(-0.5, 0, 0.8))
mu <- calibrate_abilities(c(0.5, 0.3, 0.2))
s <- Blocks(cluster = c(1, 1, 2, 2), loading = rep(0.4, 4), D = rep(1, 4))
pb <- race_probabilities(c(-0.3, 0, 0.1, 0.2), structure = s)
```

skew_normal_density *Skew-normal performance density on a symmetric lattice*

Description

The reference performance distribution of the horse race problem: proportional to `2/scale * dnorm(t) * pnorm(a t)`, normalized and centered on the lattice.

Usage

```
skew_normal_density(L, unit, loc = 0, scale = 1.0, a = 2.0)
```

Arguments

`L`	half-width: the density has length 2L+1 on lattice -L..L
`unit`	lattice spacing (performance units per lattice step)
`loc`	location shift, in performance units
`scale`	scale, in performance units
`a`	skew (a > 0 puts the fat tail on the right: slow stragglers, the usual racing shape)

Value

Numeric density of length 2L+1.

state_prices_from_offsets *Lattice race primitives*

Description

The two primitives of the lattice race: the distribution of the winning (minimum) time of a field, with exact dead-heat multiplicity tracking, and the state prices of translated copies of a common density.

Usage

```
state_prices_from_offsets(density, offsets)

winner_of_many(densities)
```

Arguments

`density`	numeric density on the symmetric lattice (length 2L+1)
`offsets`	numeric translations in lattice units (lower is better)
`densities`	list of numeric densities, all the same length

Value

`state_prices_from_offsets` returns the expected payoff of each contestant against the field (1 if strictly first, split on dead heats). `winner_of_many` returns a list with the `density` of the minimum of the field and the dead-heat `multiplicity` lattice.

`structures` *One race, five covariance grammars*

Description

Every model in this package is the SAME Gaussian min-race, Y = mu + noise; these constructors are declarative descriptions of the noise covariance admitting O(N)-per-lattice-point evaluation: Sigma = diag(D); V V' + diag(D); block-diagonal rank-1 + diag; blocks plus a global factor dialed by gamma; a hierarchy of uniform shared effects. Independent = Blocks with zero loadings; Blocks = Tree of depth 1.

Usage

```
Independent(D)

Factor(V, D)

Blocks(cluster, loading, D)

Nested(cluster, loading, D, coupling, gamma = 1)

Tree(cluster, loading, D, parent, strength)
```

Arguments

`D`	idiosyncratic variances (always the VARIANCE)
`V`	N x k loading matrix
`cluster`	cluster labels
`loading`	within-cluster loadings
`coupling`	loadings on the global factor
`gamma`	coupling strength in [0, 1]
`parent`	1-based parent index per node (root: 0 or NA)
`strength`	per-node uniform shared-effect strength

Value

A structure object for the structure= argument of `race_probabilities`, `abilities_from_race`, `race_jacobian` and `polish_race`.

tree_from_linkage *The tree race implied by a hierarchical clustering*

Description

Builds the tree race whose implied correlation is EXACTLY the cophenetic correlation matrix 1 - 2 d^2 of the clustering (HRP's implicit covariance): each merge at height h contributes lam^2 = rho - rho_parent with rho = 1 - 2 h^2. Pass the result as structure= to `race_probabilities` or `polish_race` to price or polish along the dendrogram.

Usage

```
tree_from_linkage(Z)

tree_from_hclust(hc)
```

Arguments

Z	scipy-style linkage matrix (n-1 rows: merged node ids 0-based, cophenetic distance, size)
hc	an `hclust` object

Value

A `Tree` structure with unit total variance per runner.

Examples

```
hc <- hclust(dist(matrix(rnorm(40), 8, 5)) / 10, method = "average")
tr <- tree_from_hclust(hc)
p <- race_probabilities(rnorm(8), structure = tr)
```

win_probabilities_factor *All win probabilities of a factor Gaussian race*

Description

Computes every P(X_i = min_j X_j) for X = mu + V f + sqrt(D) eps in one shared-lattice pass per factor node: conditional on f the alternatives are independent, the product of all survival functions is built once, and each alternative's own factor is divided back out.

Usage

```
win_probabilities_factor(mu, V, D, nodes = NULL, points = 501)
```

Arguments

mu	vector of locations (length N); the minimum wins. Negate for argmax (utility) races.
V	N by k matrix of factor loadings.
D	vector of idiosyncratic variances.
nodes	optional list(F, W) of factor nodes; defaults to `hermite_nodes(ncol(V))`.
points	lattice size.

Value

Vector of win probabilities summing to one.

Examples

```
p <- win_probabilities_factor(c(0, .5, 1), matrix(rnorm(6, 0, .3), 3), rep(1, 3))
```

Index