Package: gosset (via r-universe)

November 23, 2026

Type Package

Title Tools for Data Analysis in Experimental Agriculture

Version 2.0.0

URL https://agrdatasci.github.io/gosset/

BugReports https://github.com/agrdatasci/gosset/issues

Description Methods to analyse experimental agriculture data, from data synthesis to model selection and visualisation, with a focus on ranking data from decentralised on-farm trials with the 'tricot' approach (van Etten et al. 2019 <doi:10.1017/S0014479716000739>). Includes tools to build rankings, fit and cross-validate Plackett-Luce models and trees, compare traits with Kendall's tau, and summarise results as worth, regret and preference indices. The package is named after W.S. Gosset aka 'Student', a pioneer of modern statistics in small sample experimental design and analysis.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0), methods

Imports ggplot2 (>= 3.4.0), partykit, PlackettLuce, psychotools, qvcalc, stats, utils, patchwork, parallel

Suggests BradleyTerry2, ggrepel, climatrends, ClimMobTools, chirps, gnm, knitr, psychotree, rmarkdown, testthat (>= 3.0.0)

Language en-GB

RoxygenNote 7.3.2

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Kauê de Sousa [aut, cre] (ORCID:
<https://orcid.org/0000-0002-7571-7845>), Jacob van Etten [aut]
(ORCID: <https://orcid.org/0000-0001-7554-2558>), David Brown
[aut] (ORCID: <https://orcid.org/0000-0003-2859-1618>),
Jonathan Steinke [aut] (ORCID:
<https://orcid.org/0000-0001-5742-702X>), Joost van Heerwaarden
[aut] (ORCID: <https://orcid.org/0000-0002-4959-3914>)

Maintainer Kauê de Sousa <desousa.kaue@gmail.com>

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

Date/Publication 2026-11-23 07:50:10 UTC

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

RemoteRef HEAD

RemoteSha 1ffc328130c9371855f8a18a8c3c52ee866458fe

Contents

akaike_weights . . . 3
anova.PlackettLuce . . . 4
breadwheat . . . 5
btpermute . . . 6
cassava . . . 8
compare . . . 9
crossvalidation . . . 10
gosset . . . 12
kendallTau . . . 13
kendallW . . . 17
kenyachoice . . . 18
nicabean . . . 19
pairwise_probs . . . 20
plot.pltree . . . 21
preference_index . . . 23
pseudo_rank . . . 24
pseudoR2 . . . 25
rank_numeric . . . 27
rank_tricot . . . 28
rank_tricot2 . . . 30
regret . . . 32
reliability . . . 34
resample . . . 36
rowbind . . . 37
set_binomialfreq . . . 38
set_paircomp . . . 39
worth_map . . . 40

Index **43**

akaike_weights *Akaike weights*

Description

Akaike weights represent the relative likelihood of a model. It can be used in model averaging and selection.

Usage

```
akaike_weights(object)
```

Arguments

object	a numerical vector with models goodness of fit coefficients

Value

A data frame containing the coefficients:

delta	the delta overall change in the coefficients
relative_logLik	the relative log-likelihood
akaike_weights	the Akaike weights

Author(s)

Kauê de Sousa and Jacob van Etten

References

Wagenmakers E. J. & Farrell S. (2004). Psychonomic Bulletin and Review, 11(1), 192-196. doi:10.3758/BF03206482

Examples

```
data("airquality")

# try three model approaches
mod1 = glm(Temp ~ 1,
           data = airquality,
           family = poisson())

mod2 = glm(Temp ~ Ozone,
           data = airquality,
           family = poisson())

mod3 = glm(Temp ~ Ozone + Solar.R,
           data = airquality,
```

```
            family = poisson())

# models AICs together in a single vector
models = c(mod1 = AIC(mod1),
            mod2 = AIC(mod2),
            mod3 = AIC(mod3))

# calculate akaike weights
aw = akaike_weights(models)

# the higher the better
names(models[which.max(aw$akaike_weights)])
```

`anova.PlackettLuce` *Likelihood-ratio test*

Description

Assesses the goodness of fit of competing statistical models

Usage

```
## S3 method for class 'PlackettLuce'
anova(object, ...)

likelihood_ratio(x, split, ...)
```

Arguments

`object`	an object of class PlackettLuce
`...`	additional arguments passed to methods
`x`	an object of class rankings or grouped_rankings
`split`	a vector indicating the splitting rule for the test

Details

The test compares a Plackett-Luce model fitted to all data with separate models fitted within each group defined by `split`. The degrees of freedom are the sum of the ranks (number of estimated parameters) of the group models minus the rank of the global model. When a group has no ties, its tie parameter is fixed at zero, at the boundary of the parameter space, and the chi-square p-value is approximate.

Value

A data.frame with the deviance, the degrees of freedom (`DF_delta`) and the p-value of the chi-square test

Author(s)

Joost van Heerwaarden and Kauê de Sousa

Examples

```
library("PlackettLuce")
example("beans", package = "PlackettLuce")
G = group(R, rep(seq_len(nrow(beans)), 4))
d = cbind(G, beans)

split = ifelse(d$maxTN < 18.7175, TRUE, FALSE)

likelihood_ratio(G, split)

mod = PlackettLuce(G)

anova(mod)
```

---

breadwheat *Preferred bread wheat varieties*

Description

Data from decentralized on-farm trials of bread wheat (*Triticum aestivum* L.) varieties in Vaishali, India over the 2014 Rabi season. Farmers were asked to test three varieties of bread wheat randomly assigned as incomplete blocks of three varieties (out of 16 varieties) and assess which one had the best and worst performance considering four traits, germination, grain quality, yield and overall performance.

Usage

```
breadwheat
```

Format

A data frame with 493 records and 19 variables:

`variety_a` The name of variety A in the comparison.

`variety_b` The name of variety B in the comparison.

`variety_c` The name of variety C in the comparison.

`district` The administrative region where the experiment was stablished.

`village` The village within the administrative region where the experiment was stablished.

`participant_name` The participant name (omitted for protection and privacy).

`age` The participant age.

`gender` The participant gender M = Male; F = Female.

`planting_date` The date which the experiment started.

`lon` The longitude in which the experiment was stablished.

`lat` The latitude in which the experiment was stablished.

`germination_best` The variety ranked as best for germination ("A", "B" or "C").

`germination_worst` The variety ranked as worst for germination ("A", "B" or "C").

`grainquality_best` The variety ranked as best for grain quality ("A", "B" or "C").

`grainquality_worst` The variety ranked as worst for grain quality ("A", "B" or "C").

`yield_best` The variety ranked as best for yield ("A", "B" or "C").

`yield_worst` The variety ranked as worst for yield ("A", "B" or "C").

`overall_best` The variety ranked as best for overall performance ("A", "B" or "C").

`overall_worst` The variety ranked as worst for overall performance ("A", "B" or "C").

Source

van Etten, J., et. al. (2016). Experimental Agriculture, 55, 275-296. doi:10.1017/S0014479716000739

van Etten, J., et. al. (2019). PNAS 116(10) 4194-4199 doi:10.1073/pnas.1813720116

`btpermute` *Variable selection with Permuted Inclusion Criterion*

Description

Method of forward variable selection based on deviance for Bradley-Terry models using pairwise ranking data. The selection procedure consists of two steps, first, permuting the variables from the original `predictors` with `n.iterations`, then performing a forward selection to retain the `predictors` with highest contribution to the model, see details.

Usage

```
btpermute(
  contests = NULL,
  predictors = NULL,
  n.iterations = 15,
  seed = NULL,
  ...
)
```

Arguments

`contests`	a data frame with pairwise binary contests with these variables 'id','player1','player2','win1','win2'; in that order. The id should be equivalent to the index of each row in `predictors`
`predictors`	a data frame with player-specific variables with row indices that should match with the ids in `contests`. An id is not required, only the predictor variables, the ids are the index for each row

n.iterations	integer, number of iterations to compute
seed	integer, the seed for random number generation. If NULL (the default), **gosset** will set the seed randomly
...	additional arguments passed to **BradleyTerry2** methods

Details

The selection procedure consists of two steps. In the first step, btpermute adds to the set of original (candidate) predictors variables an additional set of 'fake', permuted variables. This set of permuted predictors is created by assigning to each ranking the variables from another, randomly selected ranking. The permuted variables are not expected to have any predictive power for pairwise rankings. In the second step, btpermute adds predictors to the Bradley-Terry model in a forward selection procedure. Each predictors (real and permuted) is added to the null model individually, and btpermute retains which variable reduces model deviance most strongly. The two-step process is replicated n times with argument n.iterations. At each iteration, a new random permutation is generated and all variables are tested. Replicability can be controlled using argument seed. Across the n n.iterations, the function identifies the predictor that appeared most often as the most deviance-reducing one. When this is a real variable, it is constantly added to the model and the forward selection procedure moves on - again creating new permutations, adding real and fake variables individually, and examining model deviance. Variable selection stops when a permuted variable is found to be most frequently the most deviance-reducing predictors across n.iterations. In turn, variable selection continuous as long as any real variable has stronger explanatory power for pairwise rankings than the random variables.

Value

an object of class gosset_btpermute with the final BTm() model, selected variables, seeds (random numbers) used for permutations and deviances

Author(s)

Jonathan Steinke and Kauê de Sousa

References

Lysen, S. (2009) Permuted inclusion criterion: A variable selection technique. University of Pennsylvania

See Also

set_binomialfreq, BTm

Other model selection functions: crossvalidation()

Examples

```
require("BradleyTerry2")

data("kenyachoice", package = "gosset")
```

```
mod <- btpermute(contests = kenyachoice$contests,
                 predictors = kenyachoice$predictors,
                 n.iterations = 10,
                 seed = 1)

mod
```

---

cassava *Gari-Eba Consumer Acceptability in Cameroon and Nigeria*

Description

Data from decentralized consumer trials of gari-eba, a sub-product of cassava (*Manihot esculenta* Crantz). Consumer testing was carried out in 2022 in Cameroon and Nigeria using the tricot approach. Diverse consumers in villages, towns and cities evaluated the overall acceptability of gari-eba made from 13 cassava genotypes. Apart from overall preference of the samples, the following traits were evaluated for eba based on the triangulated insights obtained by earlier survey and participatory work in the three areas. Nigeria (Osun and Benue States): colour, smoothness, mouldability, stretchability, and taste. Cameroon (Littoral zone): colour, odour, taste, firmness and stretchability. Traits in common are: colour, taste and stretchability.

Usage

```
cassava
```

Format

A data frame with 1000 records and 27 variables

`id` The tricot package id.

`option_a` The name of genotype A in the comparison.

`option_b` The name of genotype B in the comparison.

`option_c` The name of genotype C in the comparison.

`country` The country where the experiment was conducted.

`gender` The participant gender.

`age` The participant age.

`consumption` Indicates how often the participant consumes eba.

`consumptionform` Indicates in which meal the participant consumes eba.

`colour_pos` The sample ranked as best for colour ("A", "B" or "C").

`colour_neg` The sample ranked as worst for colour ("A", "B" or "C").

`smoothness_pos` The sample ranked as best for smoothness ("A", "B" or "C").

`smoothness_neg` The sample ranked as worst for smoothness ("A", "B" or "C").

`mouldability_pos` The sample ranked as best for mouldability ("A", "B" or "C").

`mouldability_neg` The sample ranked as worst for mouldability ("A", "B" or "C").

`stretchability_pos` The sample ranked as best for stretchability ("A", "B" or "C").

`stretchability_neg` The sample ranked as worst for stretchability ("A", "B" or "C").

`taste_pos` The sample ranked as best for taste ("A", "B" or "C").

`taste_neg` The sample ranked as worst for taste ("A", "B" or "C").

`odour_pos` The sample ranked as best for odour ("A", "B" or "C").

`odour_neg` The sample ranked as worst for odour ("A", "B" or "C").

`firmness_pos` The sample ranked as best for firmness ("A", "B" or "C").

`firmness_neg` The sample ranked as worst for firmness ("A", "B" or "C").

`overall_pos` The sample ranked as best overall ("A", "B" or "C").

`overall_neg` The sample ranked as worst overall ("A", "B" or "C").

`reason_like` Open question for the reason why the participant ranked the best sample as "best".

`reason_dislike` Open question for the reason why the participant ranked the worst sample as "worst".

Source

Olaosebikan, et. al. (2023). Journal of the Science of Food and Agriculture. doi:10.1002/jsfa.12867

`compare` *Compare agreement between two methods*

Description

Measures the agreement between two methods

Usage

```
compare(x, y, ...)

## Default S3 method:
compare(x, y, labels = NULL, ...)

## S3 method for class 'PlackettLuce'
compare(x, y, ...)
```

Arguments

`x`	a numeric vector, or an object of class `PlackettLuce`
`y`	a numeric vector, or an object of class `PlackettLuce`
`...`	additional arguments passed to methods
`labels`	optional, a vector with the same length *x* to plot values

Value

a ggplot with the agreement

References

Bland, M. J., and Altman, D. G. (1986). Lancet (8476):307-10.

Examples

```
set.seed(1)
x = runif(10, -1, 2)

set.seed(2)
y = runif(10, -1, 2)

compare(x, y)
```

`crossvalidation` *Cross-validation*

Description

Methods for measuring the performance of a predictive model on sets of test data in Bradley-Terry model from **psychotree**, Generalized Linear and Generalized Nonlinear models from **gnm**, and Plackett-Luce model from **PlackettLuce**. The package that provides the model (**gnm**, **PlackettLuce** or **psychotree**) must be attached with `library()` before calling `crossvalidation()`.

Usage

```
crossvalidation(formula, data, k = 10, folds = NULL, seed = NULL, ...)

## S3 method for class 'bttree'
AIC(object, newdata = NULL, ...)

## S3 method for class 'bttree'
deviance(object, newdata = NULL, ...)

## S3 method for class 'pltree'
deviance(object, newdata = NULL, ...)

## S3 method for class 'gnm'
AIC(object, newdata = NULL, ...)

## S3 method for class 'gnm'
deviance(object, newdata = NULL, ...)
```

Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted, of the form $y\ x1 + ... + xn$
data	a data frame (or object coercible by as.data.frame to a data frame) containing the variables in the model
k	an integer for the number of bins in the cross-validation
folds	an optional vector or list specifying the k-folds in the cross-validation. A vector gives, for each row of data, the fold (1 to k) in which that row is held out for testing. A list gives, for each fold, the row positions of data used for training, as returned by caret::createFolds(..., returnTrain = TRUE). Rows dropped for missing values are removed from the folds
seed	integer, the seed used to create the folds. If NULL (the default), folds are drawn from the current random number stream. When a seed is given, the caller's random number state is restored on exit
...	additional arguments passed to the methods of the chosen model
object	a model object
newdata	a data.frame with test data

Value

an object of class gosset_cv with the cross-validation goodness-of-fit estimates, which are:

AIC	Akaike Information Criterion
deviance	Model deviance
logLik	Log-Likelihood
MaxLik	Maximum likelihood pseudo R-squared
CraggUhler	Cragg and Uhler's pseudo R-squared
McFadden	McFadden pseudo R-squared
kendallTau	the Kendall correlation coefficient

Author(s)

Kauê de Sousa, Jacob van Etten and David Brown

References

Elder J. F. (2003). Journal of Computational and Graphical Statistics, 12(4), 853-864. doi:10.1198/1061860032733

James G., et al. (2013). doi:10.1007/9781461471387

Whitlock M. C. (2005). Journal of Evolutionary Biology, 18(5), 1368-1373. doi:10.1111/j.1420-9101.2005.00917.x

See Also

bttree, gnm, pltree

Other model selection functions: btpermute()

Examples

```
# Generalized Linear Models
if (require("gnm")) {
data("airquality")

cv = crossvalidation(Temp ~ Wind + Solar.R,
                     data = airquality,
                     k = 3,
                     seed = 999,
                     family = poisson())
}
# Plackett-Luce Model
if(require("PlackettLuce")) {
# beans data from PlackettLuce
data("beans", package = "PlackettLuce")

G = rank_tricot(data = beans,
                items = c(1:3),
                input = c(4:5),
                additional.rank = beans[c(6:8)],
                group = TRUE)

beans = cbind(G, beans)

# take seasons as bins
k = length(unique(beans$season))
folds = as.integer(as.factor(beans$season))

cv = crossvalidation(G ~ maxTN,
                     data = beans,
                     k = k,
                     folds = folds,
                     minsize = 100)
}
```

`gosset` *Tools for Data Analysis in Experimental Agriculture*

Description

Methods to analyse experimental agriculture data, from data synthesis to model selection and visualisation, with a focus on ranking data from decentralised on-farm trials with the 'tricot' approach (van Etten et al. 2019 doi:10.1017/S0014479716000739). Includes tools to build rankings, fit and cross-validate Plackett-Luce models and trees, compare traits with Kendall's tau, and summarise results as worth, regret and preference indices. The package is named after W.S. Gosset aka 'Student', a pioneer of modern statistics in small sample experimental design and analysis.

Author(s)

Kauê de Sousa and Jacob van Etten and David Brown and Jonathan Steinke

See Also

Useful links:

- gosset paper: doi:10.1016/j.softx.2023.101402
- Development repository: https://github.com/AgrDataSci/gosset
- Static documentation: https://AgrDataSci.github.io/gosset/
- Report bugs: https://github.com/AgrDataSci/gosset/issues

`kendallTau` *Kendall rank correlation coefficient*

Description

Compute Kendall rank correlation coefficient between two objects. Kendall is a coefficient used in statistics to measure the ordinal association between two measured quantities. A tau test is a non-parametric hypothesis test for statistical dependence based on the tau coefficient. The 'kendallTau' function applies the "kendall" method from 'stats::cor' with some previous treatment in the data, such as converting floating numbers into ranks (from the higher being the first and negative being the last) and the possibility to remove zeros from incomplete ranks.

Perform a pairwise permutation test to assess statistical differences in Kendall's Tau correlation between two or more groups.

Usage

```
kendallTau(x, y, null.rm = TRUE, average = TRUE, na.omit = FALSE, ...)

## Default S3 method:
kendallTau(
  x,
  y,
  null.rm = TRUE,
  ...,
  alternative = c("two.sided", "greater", "less")
)

## S3 method for class 'matrix'
kendallTau(
  x,
  y,
  null.rm = TRUE,
  average = TRUE,
  na.omit = FALSE,
```

```
  ...,
  alternative = c("two.sided", "greater", "less")
)

## S3 method for class 'rankings'
kendallTau(x, y, ...)

## S3 method for class 'grouped_rankings'
kendallTau(x, y, ...)

## S3 method for class 'paircomp'
kendallTau(x, y, ...)

kendallTau_bootstrap(x, y, nboot = 100, seed = NULL, ...)

kendallTau_permute(x, y, split, n.permutations = 500)
```

Arguments

x	a numeric vector, matrix or data frame
y	a vector, matrix or data frame with compatible dimensions to x
null.rm	logical, to remove zeros from x and y
average	logical, if FALSE returns the kendall and N-effective for each entry
na.omit	logical, if TRUE ignores entries with kendall = NA when computing the average
...	further arguments affecting the Kendall tau produced. See details
alternative	the alternative hypothesis, one of "two.sided" (the default), "greater" (positive association) or "less" (negative association). The p-value is returned in the column Pr(>|z|) whatever the alternative
nboot	integer, the size of the bootstrap sample
seed	integer, the seed for random number generation. If NULL (the default), the current random number stream is used
split	a vector indicating the splitting rule for the test
n.permutations	an integer, the number of permutations to perform

Details

kendallTau_bootstrap() resamples the rows (participants) with replacement nboot times and returns the pair-weighted average tau of each resample. The spread of these values estimates the sampling variability of the average tau, for example to draw boxplots by trait. Rows with fewer than two ranked items, or with tau = NA, are dropped first. When seed is given, the resamples are reproducible and the caller's random number stream is left unchanged.

Value

The Kendall correlation coefficient, the Effective N (the equivalent N if all items were compared to all items), the z value and its p-value. For matrices, each row is treated as an independent ranking:

the average tau weights rows by their number of pairs, and the z value uses the null variance of that weighted average. The null variance assumes no ties; use `kendallTau_permute` for an exact test with small or tied data.

A data.frame containing:

`observed_diff`	observed absolute differences in Kendall's tau for all group pairs.
`p_values`	p-values from the permutation test for all group pairs.

Author(s)

Kauê de Sousa and Jacob van Etten

Kauê de Sousa

References

Kendall M. G. (1938). Biometrika, 30(1-2), 81-93. doi:10.1093/biomet/30.12.81

See Also

`cor`

`kendallTau`

Other goodness-of-fit functions: `kendallW()`, `pseudoR2()`

Examples

```
# Vector based example same as stats::cor(x, y, method = "kendall")
# but showing N-effective
x = c(1, 2, 3, 4, 5)

y = c(1, 1, 3, 2, NA)

w = c(1, 1, 3, 2, 5)

kendallTau(x, y)

kendallTau(x, w)

# Matrix and PlacketLuce ranking example

library("PlackettLuce")

R = matrix(c(1, 2, 4, 3,
             1, 4, 2, 3,
             1, 2, NA, 3,
             1, 2, 4, 3,
             1, 3, 4, 2,
             1, 4, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) = LETTERS[1:4]

G = group(as.rankings(R), 1:6)
```

```
mod = pltree(G ~ 1, data = G)

preds = predict(mod)

kendallTau(R, preds)

# Also returns raw values (no average)

kendallTau(R, preds, average = FALSE)

# Choose to ignore entries with NA
R2 = matrix(c(1, 2, 4, 3,
              1, 4, 2, 3,
              NA, NA, NA, NA,
              1, 2, 4, 3,
              1, 3, 4, 2,
              1, 4, 3, 2), nrow = 6, byrow = TRUE)

kendallTau(R, R2, average = FALSE)

kendallTau(R, R2, average = TRUE)

kendallTau(R, R2, average = TRUE, na.omit = TRUE)

# Variability of the average tau across participants
boot = kendallTau_bootstrap(R, preds, nboot = 200, seed = 1)
quantile(boot, c(0.025, 0.5, 0.975))


set.seed(42)
x = rnorm(100)
y = rnorm(100)
split = rep(c("Group1", "Group2", "Group3"), length.out = 100)
kendallTau_permute(x, y, split)

data("breadwheat", package = "gosset")

x = rank_tricot(breadwheat,
                items = paste0("variety_", letters[1:3]),
                input = c("yield_best", "yield_worst"),
                validate.rankings = TRUE)

y = rank_tricot(breadwheat,
                items = paste0("variety_", letters[1:3]),
                input = c("overall_best", "overall_worst"),
                validate.rankings = TRUE)

kendallTau_permute(x, y,
             split = rep(c("Group1", "Group2", "Group3"), length.out = nrow(breadwheat)),
                n.permutations = 100)
```

`kendallW` *Kendall's W (coefficient of concordance)*

Description

Compute Kendall's W, also known as coefficient of concordance among observed rankings and those predicted by the Plackett-Luce model.

Usage

```
kendallW(x, ...)

## Default S3 method:
kendallW(x, y, ...)

## S3 method for class 'pltree'
kendallW(x, newdata = NULL, ...)
```

Arguments

`x`	an object of class `pltree`, or a matrix of predicted rankings for the default method
`...`	additional arguments passed to methods
`y`	a matrix of observed rankings, for the default method
`newdata`	data for predictions

Details

Kendall's W is computed for each ranking (row) between the observed and the predicted ranks, with the correction for ties, and then averaged across rankings. Only the items that the participant ranked are used. Items with a 0 (not ranked) or NA in the observed rankings, or NA in the predicted rankings, are dropped from that row, and rows with fewer than two items left are skipped.

Kendall's W values range from 0 (no agreement) to 1 (full agreement)

Value

Kendall's W (coefficient of concordance)

Author(s)

David Brown and Jacob van Etten

See Also

Other goodness-of-fit functions: `kendallTau()`, `pseudoR2()`

Examples

```
pred = matrix(c(1, 2, 3, 4,
                2, 1, 3, 4,
                1, 3, 2, 4), nrow = 3, byrow = TRUE)

# 0 marks items the participant did not rank
obs = matrix(c(1, 2, 3, 0,
               1, 2, 0, 3,
               0, 2, 1, 3), nrow = 3, byrow = TRUE)

kendallW(pred, obs)
```

---

kenyachoice *Kenyan farmers' preferences for agricultural and livelihood practices*

Description

Data from a preference experiment in Makueni County, Kenya. Twenty-six smallholder farmers ordered 9 different livelihood improvement practices by their personal preference. Full rankings were broken down into multiple pairwise rankings (`kenyachoice[["contests"]]`). To each respondent, a set of nine socio-economic variables is available (`kenyachoice[["predictors"]]`).

Usage

```
kenyachoice
```

Format

A list with two dataframes. `kenyachoice[["contests"]]` contains the pairwise rankings from farmers choices. `kenyachoice[["predictors"]]` contains the socioeconomic data for each farmer. Codes for contests are described:

- `B` Opening a business
- `D` Dry planting
- `G` Collective crop marketing
- `J` Finding off-farm job
- `M` Machine tillage
- `O` Renting out traction animals
- `R` Mulching
- `T` Terracing
- `Z` Zai pits

Source

Steinke, J., et. al. (2019). Computers and Electronics in Agriculture, 162, 991-1000. doi:10.1016/j.compag.2019.05.026

`nicabean` *Common bean on-farm trial in Nicaragua*

Description

Data from decentralized on-farm trials of common bean (*Phaseolus vulgaris* L.) varieties in Nicaragua over five seasons between 2015 and 2016. Following the tricot approach, farmers were asked to test three varieties of common bean randomly assigned as incomplete blocks of three varieties (out of 10 varieties) and assess which of those three had the best and worst performance in nine traits (Vigor, Architecture, Resistance to Pests, Resistance to Diseases, Tolerance to Drought, Yield, Marketability, Taste, and Overall Appreciation).

Usage

```
nicabean
```

Format

A list with two data frames, `nicabean[["trial"]]` contains the trial data:

`id` the plot id

`item` the variety name

`trait` the trait for the given variety and plot id

`rank` the rank for the given variety and trait, with 1 being higher and 3 the lowest

`nicabean[["bean_covar"]]` contains the covariates associated with the data:

`id` the plot id

`adm0` the country name where trials were set

`longitude` the longitude of the trial plot

`latitude` the latitude of the trial plot

`trial` the trial name as registered on ClimMob

`variety_a` the variety assigned as label A in the incomplete block

`variety_b` the variety assigned as label B in the incomplete block

`variety_c` the variety assigned as label C in the incomplete block

`planting_date` the planting date

`gender` the farmer gender

`age` the farmer age

Source

van Etten, J., et. al. (2016). Experimental Agriculture, 55, 275-296. doi:10.1017/S0014479716000739

van Etten, J., et. al. (2019). PNAS 116(10) 4194-4199 doi:10.1073/pnas.1813720116

`pairwise_probs` *Compute pairwise probabilities*

Description

Implements Luce's Choice Axiom to calculate pairwise probabilities in a set of choice probabilities. The Luce's Choice Axiom states that the probability of selecting one item over another from a pool of many items is not affected by the presence or absence of other items in the pool.

Usage

```
pairwise_probs(object, relative.probs = TRUE, ...)
```

Arguments

`object`	a named numeric vector with probabilities
`relative.probs`	logical, TRUE to return matrix with relative probs (prob - 0.5), otherwise true values are returned
`...`	additional arguments passed to methods

Value

a matrix with pairwise probabilities

Examples

```
library("PlackettLuce")
library("ggplot2")

R = matrix(c(1, 2, 3, 0,
             4, 1, 2, 3,
             2, 1, 3, 4,
             1, 2, 3, 0,
             2, 1, 3, 0,
             1, 0, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) = c("apple", "banana", "grape", "pear")

mod = PlackettLuce(R)

coefs = coefficients(mod, log = FALSE)

pair_worth = pairwise_probs(coefs)

pair_worth

# plot the results
lvls = dimnames(pair_worth)[[1]]

pair_dat = data.frame(player1 = rep(lvls, times = length(lvls)),
```

```
                      player2 = rep(lvls, each = length(lvls)),
                      worth = as.vector(pair_worth))

pair_dat

pair_dat$player1 = factor(pair_dat$player1, levels = lvls)

pair_dat$player2 = factor(pair_dat$player2, levels = rev(lvls))

pair_dat$worth = round(pair_dat$worth, 2)

ggplot(pair_dat,
       aes(x = player2,
           y = player1,
           fill = worth,
           label = worth)) +
  geom_tile(show.legend = FALSE) +
  geom_text() +
  scale_fill_gradient2(low = "#b2182b",
                       high = "#2166ac",
                       na.value = "white") +
  scale_x_discrete(position = "top") +
  theme_bw() +
  theme(axis.text = element_text(color = "grey10"),
        strip.text.x = element_text(color = "grey10"),
        axis.text.x = element_text(angle = 90, hjust = 0),
        panel.grid = element_blank()) +
  labs(x = "",
       y = "",
       fill = "")
```

`plot.pltree` *Get node labels and rules used in a party tree*

Description

Returns the covariates used to split a recursive partitioning tree and the rules that were applied to build the tree

Usage

```
## S3 method for class 'pltree'
plot(x, log = TRUE, ref = NULL, ...)

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

node_labels(x)
```

```
node_rules(x)

top_items(x, top = 5)
```

Arguments

x	an object of class party or PlackettLuce
log	logical, if TRUE log-worth coefficients are displayed instead of worth
ref	optional, character or integer for the reference item when *log* = TRUE
...	additional arguments passed to methods. See details
top	an integer for the number of items to return

Details

multcomp = TRUE adds multi-comparison letters from multcompView ci.level = numeric for the confidence interval levels

Value

a vector with the node labels, a data.frame with node rules, a ggplot object

Author(s)

Kauê de Sousa

Examples

```
library("PlackettLuce")
library("ggplot2")

data("beans", package = "PlackettLuce")

G = rank_tricot(data = beans,
                items = c(1:3),
                input = c(4:5),
                group = TRUE,
                additional.rank = beans[c(6:8)])

pld = cbind(G, beans[,c("maxTN", "season", "lon")])

tree = pltree(G ~ maxTN + season + lon, data = pld)

node_labels(tree)

node_rules(tree)

top_items(tree)

plot(tree)
```

```
plot(tree, log = FALSE)

################################

# Plot PlackettLuce
R = matrix(c(1, 2, 4, 3,
             4, 1, 2, 3,
             2, 4, 1, 3,
             1, 2, 3, 0,
             2, 1, 4, 3,
             1, 4, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) = c("apple", "banana", "orange", "pear")
R = as.rankings(R)

mod = PlackettLuce(R)

plot(mod)
```

`preference_index` *Preference index from rankings*

Description

Computes a preference index for each item in a ranking using Plackett-Luce worth parameters. For each item, the function estimates the probability of being ranked at the top or bottom of the sets in which it appears, and calculates a net winning probability as the difference between top and bottom probabilities.

Usage

```
preference_index(object)
```

Arguments

`object`	A `rankings` object (from **PlackettLuce**) or a matrix that can be coerced to rankings with `as.rankings`.

Details

The preference index provides a model-based summary of performance across all ranking sets. Probabilities are derived from the fitted Plackett-Luce worth parameters, normalized to sum to one.

For each item:

- `top_probs` is the expected probability (in percent) of being ranked first in its sets.
- `bottom_probs` is the expected probability (in percent) of being ranked last in its sets.

- `net_winning_probs` is the difference `top_probs - bottom_probs`, a relative preference score.
- `worth` is the normalized worth parameter from the Plackett-Luce model.

The output is ordered by descending net winning probability, providing a leaderboard-style summary of items.

Value

A `gosset_df` (data.frame subclass) with columns:

item Item identifier (name).

n Number of sets with two or more ranked items in which the item appears.

top Expected top probability (%).

bottom Expected bottom probability (%).

net_winning Net winning probability (%).

worth Normalized worth parameter.

See Also

`PlackettLuce`, `as.rankings`, `worth_map`

Examples

```
library(PlackettLuce)

R = matrix(c(1, 2, 0, 0,
             4, 1, 2, 3,
             2, 1, 3, 0,
             1, 2, 3, 0,
             2, 1, 3, 0,
             1, 0, 3, 2),
           nrow = 6, byrow = TRUE)
colnames(R) = c("apple", "banana", "orange", "pear")
R = as.rankings(R)

preference_index(R)
```

`pseudo_rank` *Add pseudo-rank to missing values*

Description

Add pseudo-rank to missing values

Usage

```
pseudo_rank(object, ...)
```

Arguments

object	a matrix or PlackettLuce rank
...	additional arguments passed to methods

Value

a matrix or PlackettLuce rank

Examples

```
library("PlackettLuce")
R = matrix(c(1, 2, 0, 0,
             4, 1, 0, 3,
             2, 1, 0, 3,
             1, 2, 0, 0,
             2, 1, 0, 0,
             1, 0, 0, 2), nrow = 6, byrow = TRUE)
colnames(R) = c("apple", "banana", "orange", "pear")

# summary(PlackettLuce(R))

R = pseudo_rank(R)

summary(PlackettLuce(R))
```

---

pseudoR2 *Pseudo R-squared*

Description

Regression coefficient to evaluate goodness-of-fit in a given model when ordinary least squares (OLS) are not available. The algorithm computes estimates from the maximum likelihood through an iterative process. These estimates are called 'pseudo R-squared' because they look like 'R-squared' in the sense that they are on a similar scale (from 0 to 1), with higher values indicating better model fit.

Usage

```
pseudoR2(object, ...)

## Default S3 method:
pseudoR2(object, ...)

## S3 method for class 'pltree'
pseudoR2(object, newdata = NULL, ...)

## S3 method for class 'bttree'
pseudoR2(object, ...)
```

Arguments

object	a model object of class glm, gnm, lm, pltree or bttree
...	additional arguments affecting the R-squared produced
newdata	a data.frame with testing data

Value

A data frame containing the pseudo R-squared coefficients:

logLik	log-likelihood
logLikNull	Null log-likelihood
MaxLik	maximum likelihood pseudo R-squared
CraggUhler	Cragg and Uhler's pseudo R-squared
McFadden	McFadden pseudo R-squared

Author(s)

Kauê de Sousa and Jacob van Etten

References

Agresti A. (2002). Categorical Data Analysis. John Wiley & Sons, Inc., Hoboken, NJ, USA. doi:10.1002/0471249688

Hunter D. R. (2004). The Annals of Statistics, 32(1), 384-406. http://www.jstor.org/stable/3448514

Cragg, J. G., & Uhler, R. S. (1970). The Canadian Journal of Economics 3(3), 386-406. doi:10.2307/133656

McFadden, D. (1973). Conditional logit analysis of qualitative choice behavior.

See Also

Other goodness-of-fit functions: kendallTau(), kendallW()

Examples

```
data("airquality")

mod = glm(Temp ~ Wind + Solar.R,
          data = airquality,
          family = poisson())

pseudoR2(mod)
```

rank_numeric *Plackett-Luce rankings from numeric values*

Description

Group and coerce numeric values into Plackett-Luce rankings.

Usage

```
rank_numeric(
  data,
  items,
  input,
  id = NULL,
  group = FALSE,
  ascending = FALSE,
  ...
)
```

Arguments

data	a data.frame with columns specified by items and input values
items	a character or numerical vector for indexing the column(s) containing the item names in data
input	a character or numerical vector for indexing the column(s) containing the values in data to be ranked
id	an index of data indicating the ids for "long" data
group	logical, if TRUE return an object of class "grouped_rankings"
ascending	logical, only for floating point numbers, to compute rankings from lower to higher values
...	additional arguments passed to methods

Value

a PlackettLuce "rankings" object, which is a matrix of dense rankings

Author(s)

Kauê de Sousa

See Also

rankings

Other rank functions: rank_tricot(), rank_tricot2(), set_binomialfreq(), set_paircomp()

Examples

```
# A matrix with 10 rankings of 5 items (A, B, C, D, E)
# with numeric values as "rank"
set.seed(123)
df = cbind(id = rep(1:10, each = 5),
            items = rep(LETTERS[1:5], times = 10),
            input = runif(50, 1, 3))

# return an object of class 'rankings'
R = rank_numeric(df,
                  items = 2,
                  input = 3,
                  id = 1)

# rankings can be computed in ascending order
R = rank_numeric(df,
                  items = 2,
                  input = 3,
                  id = 1,
                  ascending = TRUE)


# return an object of class 'grouped_rankings'
R = rank_numeric(df,
                  items = 2,
                  input = 3,
                  id = 1,
                  group = TRUE)
```

`rank_tricot` *Build Plackett-Luce rankings from tricot dataset*

Description

Create an object of class "rankings" from tricot data. Tricot stands for "triadic comparison of technology options". It is an approach to carry out large decentralized agronomic field experiments as incomplete blocks. Each incomplete block contains a set of three randomised technologies out of a larger set.

Usage

```
rank_tricot(
  data,
  items,
  input,
  group = FALSE,
  validate.rankings = FALSE,
  additional.rank = NULL,
```

```
  ...
)
```

Arguments

`data`	a data.frame with columns specified by items and input values
`items`	a character or numerical vector for indexing the column(s) containing the item names in `data`
`input`	a character or numerical vector for indexing the column(s) containing the values in `data` to be ranked
`group`	logical, if `TRUE` return an object of class "grouped_rankings"
`validate.rankings`	logical, if `TRUE` implements a check on ranking consistency looking for possible ties, NA or letters other than A, B, C. These entries are set to 0
`additional.rank`	optional, a data frame for the comparisons between tricot items and the local item
`...`	additional arguments passed to methods. See details

Details

full.output: logical, to return a list with a "rankings", a "grouped_rankings" and the ordered items

Value

a PlackettLuce "rankings" object, which is a matrix of dense rankings

Author(s)

Kauê de Sousa and Jacob van Etten, with ideas from Heather Turner

References

van Etten J., et al. (2016). Experimental Agriculture, 55(S1), 275-296. doi:10.1017/S0014479716000739

See Also

`rankings`, `breadwheat`

Other rank functions: `rank_numeric()`, `rank_tricot2()`, `set_binomialfreq()`, `set_paircomp()`

Examples

```
library("PlackettLuce")
data("beans", package = "PlackettLuce")

# Using a subset of the bean data
beans = beans[1:5, 1:5]
beans[1, 1] = NA
beans[3, 4:5] = NA
```

```
beans[5, 5] = "Tie"

# The default approach do not validate rankings
# accepting any entry used in the argument input
R1 = rank_tricot(beans,
                 items = c(1:3),
                 input = c(4:5),
                 group = FALSE)

# Using validate.rankings = TRUE, the rankings
# are only considered for those entries without
# NAs, Ties and with any of the letters A, B, C
# this does not affect the length of the final ranking
R2 = rank_tricot(beans,
                 items = c(1:3),
                 input = c(4:5),
                 validate.rankings = TRUE,
                 group = FALSE)

coef(PlackettLuce(R1))

coef(PlackettLuce(R2))

############################################################

# pass the comparison with local item as an additional rankings, then
# each of the 3 varieties are compared separately with the local item
# and return an object of class grouped_rankings

data("beans", package = "PlackettLuce")

G = rank_tricot(data = beans,
                 items = c(1:3),
                 input = c(4:5),
                 group = TRUE,
                 additional.rank = beans[c(6:8)])

head(G)
```

`rank_tricot2` *Build Plackett-Luce rankings from tricot ranking dataset*

Description

Create an object of class "rankings" from tricot ranking data.

Usage

```
rank_tricot2(data, id, items, input, strict = FALSE, ...)
```

Arguments

data	a data.frame with columns specified by items and input values
id	an index of data indicating the id column
items	a character or numerical vector for indexing the column(s) containing the item names in data
input	a character or numerical vector for indexing the column(s) containing the values in data to be ranked
strict	logical. If TRUE, reject missing or duplicate id/item/rank records and require positive integer ranks with no ties within an id. Defaults to FALSE to preserve the legacy warning-and-skip behaviour
...	additional arguments passed to methods. See details

Value

a PlackettLuce "rankings" object, which is a matrix of dense rankings

Author(s)

Kauê de Sousa

References

van Etten J., et al. (2016). Experimental Agriculture, 55(S1), 275-296. doi:10.1017/S0014479716000739

See Also

rankings, nicabean

Other rank functions: rank_numeric(), rank_tricot(), set_binomialfreq(), set_paircomp()

Examples

```
library("gosset")

R = nicabean$trial

R = R[R$trait == "Vigor", ]

names(R)

R = rank_tricot2(R, id = "id", items = "item", input = "rank")
```

regret *Regret-based values for risk assessment*

Description

Regret is an important heuristic in the behavioural sciences. Minimizing worst regret (the loss under the worst possible outcome) is a criterion that takes a conservative approach to risk analysis in diversification strategies.

Usage

```
regret(object, ..., bootstrap = TRUE, normalize = TRUE)

## Default S3 method:
regret(
  object,
  ...,
  values,
  items,
  group,
  node.weights = NULL,
  bootstrap = TRUE,
  normalize = TRUE
)

## S3 method for class 'pltree'
regret(object, bootstrap = TRUE, normalize = TRUE, ...)

## S3 method for class 'list'
regret(object, bootstrap = TRUE, normalize = TRUE, ...)
```

Arguments

object	a data.frame, an object of class pltree, or a list with PlackettLuce models
...	further arguments passed to methods
bootstrap	logical, to run a Bayesian bootstrap on *object*
normalize	logical, to normalize values to sum to 1
values	an index in *object* with the values to compute regret
items	an index in *object* for the different items
group	an index in *object* for the different scenarios
node.weights	optional named numeric vector with one weight per scenario in *group*, used for worth, regret and their standard errors. For pltree objects the weights are the share of observations in each terminal node, and for lists of models the share of rankings in each model. Worst regret is not weighted

Details

Additional details for Bayesian bootstrap: `statistic` A function that accepts data as its first argument and possibly, the weights as its second, if use_weights is TRUE; `n1` The size of the bootstrap sample; `n2` The sample size used to calculate the statistic each bootstrap draw

Value

A data frame with regret estimates

`items`	the item names
`worth`	the worth parameters
`regret`	the squared regret
`worst_regret`	the worst regret

Author(s)

Jacob van Etten and Kauê de Sousa

References

Loomes G. & Sugden R. (1982). The Economic Journal, 92(368), 805. doi:10.2307/2232669

Bleichrodt H. & Wakker P. P. (2015). The Economic Journal, 125(583), 493-532. doi:10.1111/ecoj.12200

Examples

```
# Case 1 ####
library("PlackettLuce")
data("breadwheat", package = "gosset")

# convert the tricot rankings from breadwheat data
# into a object of class 'grouped_rankings'

G = rank_tricot(breadwheat,
                items = c("variety_a","variety_b","variety_c"),
                input = c("overall_best","overall_worst"),
                group = TRUE)


# combine grouped rankings with temperature indices
mydata = cbind(G, breadwheat[c("lon","lat")])

# fit a pltree model using geographic data
mod = pltree(G ~ ., data = mydata)

regret(mod)

# Case 2 ####
# list of PlackettLuce models
R = matrix(c(1, 2, 3, 0,
```

```
              4, 1, 2, 3,
              2, 1, 3, 4,
              1, 2, 3, 0,
              2, 1, 3, 0,
              1, 0, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) = c("apple", "banana", "orange", "pear")

mod1 = PlackettLuce(R)

R2 = matrix(c(1, 2, 0, 3,
               2, 1, 0, 3,
               2, 1, 0, 3,
               1, 2, 0, 3,
               2, 1, 0, 3,
               1, 3, 4, 2), nrow = 6, byrow = TRUE)
colnames(R2) = c("apple", "banana", "orange", "pear")

mod2 = PlackettLuce(R2)

mod = list(mod1, mod2)

regret(mod, n1 = 500)
```

`reliability` *Probability of outperforming a check*

Description

Measures the precision of estimated values, and the potential response to selection on those estimated values compared to a check

Usage

```
reliability(x, ...)

## Default S3 method:
reliability(x, y = NULL, ...)

## S3 method for class 'PlackettLuce'
reliability(x, ref, ...)

## S3 method for class 'pltree'
reliability(x, ref, ...)
```

Arguments

`x`	a numeric vector, or an object of class `PlackettLuce` or `pltree`
`...`	additional arguments passed to methods

y	numeric, the reference value
ref	a character or integer for indexing the element containing reference values in *x*

Value

the reliability based on the worth parameters

Author(s)

Kauê de Sousa, David Brown, Jacob van Etten

References

Eskridge and Mumm (1992). Theoret. Appl. Genetics 84, 494-500 doi:10.1007/BF00229512.

Examples

```
# Case 1. vector example

x = c(9.5, 12, 12.3, 17)

y = 11.2

reliability(x, y)

# Case 2. PlackettLuce model

library("PlackettLuce")

R = matrix(c(1, 2, 4, 3,
             4, 1, 2, 3,
             2, 3, 1, 4,
             4, 2, 3, 1,
             2, 1, 4, 3,
             1, 4, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) = c("apple", "banana", "orange", "pear")

mod = PlackettLuce(R)

reliability(mod, ref = "orange")


# Case 3. PlackettLuce tree

data("beans", package = "PlackettLuce")

G = rank_tricot(data = beans,
                items = c(1:3),
                input = c(4:5),
                group = TRUE,
                additional.rank = beans[c(6:8)])
```

```
pld = cbind(G, beans[,c("maxTN", "season", "lon")])

tree = pltree(G ~ maxTN + season + lon, data = pld)

reliability(tree, ref = "Local")
```

`resample` *Re-sample model estimates*

Description

Applies a k-fold approach to re-sample estimates from PlackettLuce model. The function will subset the data into 'k' folds and re-calculate the model estimates. Optionally, a Bayesian bootstrapping technique can be used to increase output size and normalize the distribution of estimates

Usage

```
resample(object, k = 5, bootstrap = FALSE, seed = NULL, ...)
```

Arguments

`object`	a PlackettLuce model object
`k`	an integer for the number of bins to subset the data
`bootstrap`	logical, to run a Bayesian bootstrapping on object
`seed`	integer, the seed for random number generation. If NULL (the default), gosset will set the seed randomly
`...`	additional arguments passed to methods, see details

Details

Additional details for Bayesian bootstrapping: `statistic` A function that accepts data as its first argument and possibly, the weights as its second, if use_weights is TRUE; `n1` The size of the bootstrap sample; `n2` The sample size used to calculate the statistic each bootstrap draw

Value

A data frame with re-sampled estimates

Author(s)

Kauê de Sousa

Examples

```
library("PlackettLuce")

data("breadwheat", package = "gosset")

G = rank_tricot(breadwheat,
                items = c("variety_a","variety_b","variety_c"),
                input = c("overall_best","overall_worst"),
                group = FALSE)

mod = PlackettLuce(G)

# default method, no bootstrapping and 5 folds
resample(mod)

resample(mod, log = FALSE)

# the argument 'seed' will make sure that the function
# always return the same results
resample(mod, log = FALSE, seed = 1526)

# add bootstrapping
resample(mod, bootstrap = TRUE, log = FALSE, n1 = 100)
```

---

rowbind *Combine R objects by rows*

Description

Combine R objects when number and names of columns do not match

Usage

```
rowbind(x, ...)

## Default S3 method:
rowbind(x, y, ...)

## S3 method for class 'list'
rowbind(x, ...)
```

Arguments

x	an R object, typically a data.frame, matrix or list
...	additional arguments passed to methods
y	a matrix, a data.frame (or any other object that can be coerced to data.frame)

Value

a data.frame with the combined data

Examples

```
# two data frames
rowbind(airquality, mtcars)

# a list of data frames
l = list(airquality, mtcars)
rowbind(l)
```

`set_binomialfreq` *Binomial frequency rankings from pairwise contests*

Description

Binary comparisons from a ranking object. Ties are not taken into account and are set to NA.

Usage

```
set_binomialfreq(object, drop.null = FALSE, disaggregate = FALSE)
```

Arguments

`object`	an object of class `rankings`, `grouped_rankings` or `paircomp`
`drop.null`	logical, an optional argument to remove null contests
`disaggregate`	logical, if `TRUE` binaries are disaggregated by individual contests

Value

A data.frame with binary rank of pairwise contests:

`player1`	a factor with n levels for the first player in the contests
`player2`	a factor with n levels (same as player1) for the second player in the contests
`win1`	number of times player1 wins against player2
`win2`	number of times player2 wins against player1

Author(s)

Kauê de Sousa

References

Turner H. & Firth D. (2012). Journal of Statistical Software, 48(9), 1-21. doi:10.18637/jss.v048.i09

See Also

Other rank functions: `rank_numeric()`, `rank_tricot()`, `rank_tricot2()`, `set_paircomp()`

Examples

```
library("PlackettLuce")

R = matrix(c(1, 2, 0, 0,
             4, 1, 2, 3,
             2, 4, 3, 1,
             1, 2, 3, 0,
             2, 1, 1, 0,
             1, 0, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) = c("apple", "banana", "orange", "pear")

R = as.rankings(R)

set_binomialfreq(R)

set_binomialfreq(R, disaggregate = TRUE)
```

---

set_paircomp *Pairwise comparison rankings*

Description

Pairwise comparisons from a PlackettLuce ranking object. Ties are not taken into account and are set to NA.

Usage

```
set_paircomp(object)
```

Arguments

`object`	an object of class "rankings" or "grouped_rankings" which is a matrix of dense rankings.

Value

an object of class "paircomp" which is a matrix of pairwise comparisons

Author(s)

Kauê de Sousa and Jacob van Etten

References

Strobl C., Wickelmaier F. & Zeileis A. (2011). Journal of Educational and Behavioral Statistics, 36(2), 135-153. doi:10.3102/1076998609359791

See Also

rankings, paircomp

Other rank functions: rank_numeric(), rank_tricot(), rank_tricot2(), set_binomialfreq()

Examples

```
library("PlackettLuce")

R = matrix(c(1, 2, 0, 0,
             4, 1, 2, 3,
             2, 4, 3, 1,
             1, 2, 3, 0,
             2, 1, 1, 0,
             1, 0, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) = c("apple", "banana", "orange", "pear")
R = as.rankings(R)

PC = set_paircomp(R)

############################################

# using breadwheat data
data("breadwheat", package = "gosset")

# convert the tricot rankings from breadwheat data
# into a object of class 'rankings' from PlackettLuce
R = rank_tricot(breadwheat,
                items = c("variety_a","variety_b","variety_c"),
                input = c("overall_best","overall_worst"))


PC = set_paircomp(R)
```

---

worth_map *Plot worth parameters*

Description

Produces plots to highlight worth coefficients of items in a party tree or a list of PlackettLuce models

Usage

```
worth_map(object, ...)

## Default S3 method:
worth_map(object, ...)

## S3 method for class 'list'
worth_map(object, labels, labels.order = NULL, items.order = NULL, ...)

worth_bar(object, ...)
```

Arguments

object	an object of class party or an object of class PlackettLuce or a list objects of class PlackettLuce
...	additional arguments passed to methods
labels	a vector with the name of models in *object*
labels.order	optional, a vector to determine the order of labels
items.order	optional, a vector to determine the order of items

Examples

```
library("psychotree")
library("PlackettLuce")
data("Topmodel2007", package = "psychotree")
R = as.grouped_rankings(Topmodel2007$preference)

tm_tree = pltree(R ~ ., data = Topmodel2007[, -1],
                 minsize = 5,
                 npseudo = 0)

worth_map(tm_tree)

########################################

# Ranking of preference on four fruits
# based on traits taste, texture,
# price and storability

# taste
R1 = matrix(c(1, 2, 3, 4,
              4, 1, 3, 2,
              4, 1, 2, 3,
              1, 2, 0, 3), nrow = 4, byrow = TRUE)
colnames(R1) = c("apple", "banana", "orange", "pear")
mod1 = PlackettLuce(R1)

# texture
R2 = matrix(c(1, 4, 2, 3,
```

```
               1, 4, 3, 2,
               1, 4, 2, 3,
               1, 4, 2, 3), nrow = 4, byrow = TRUE)
colnames(R2) = c("apple", "banana", "orange", "pear")
mod2 = PlackettLuce(R2)

# price
R3 = matrix(c(2, 4, 3, 1,
               4, 1, 2, 3,
               3, 4, 2, 1,
               4, 3, 1, 2), nrow = 4, byrow = TRUE)
colnames(R3) = c("apple", "banana", "orange", "pear")
mod3 = PlackettLuce(R3)

# storability
R4 = matrix(c(1, 4, 3, 2,
               3, 4, 1, 2,
               1, 3, 2, 4,
               2, 3, 4, 1), nrow = 4, byrow = TRUE)
colnames(R4) = c("apple", "banana", "orange", "pear")
mod4 = PlackettLuce(R4)

# models in a list
mods = list(mod1, mod2, mod3, mod4)

# name for each model
labels = c("Taste", "Texture", "Price", "Storability")

worth_map(mods, labels)

# plot only one model as bar
worth_bar(mod1)
```

Index

∗ **dataset**
  breadwheat, 5
  cassava, 8
  kenyachoice, 18
  nicabean, 19
∗ **goodness-of-fit functions**
  kendallTau, 13
  kendallW, 17
  pseudoR2, 25
∗ **model selection functions**
  btpermute, 6
  crossvalidation, 10
∗ **rank functions**
  rank_numeric, 27
  rank_tricot, 28
  rank_tricot2, 30
  set_binomialfreq, 38
  set_paircomp, 39
∗ **utility functions**
  rowbind, 37

AIC.bttree *(*crossvalidation*)*, 10
AIC.gnm *(*crossvalidation*)*, 10
akaike_weights, 3
anova.PlackettLuce, 4
as.rankings, *23, 24*

breadwheat, 5, *29*
BTm, *7*
btpermute, 6, *11*
bttree, *11*

cassava, 8
compare, 9
cor, *15*
crossvalidation, *7*, 10

deviance.bttree *(*crossvalidation*)*, 10
deviance.gnm *(*crossvalidation*)*, 10
deviance.pltree *(*crossvalidation*)*, 10

gnm, *11*
gosset, 12
gosset-package *(*gosset*)*, 12

kendallTau, 13, *15*, *17*, *26*
kendallTau.default *(*kendallTau*)*, 13
kendallTau.grouped_rankings *(*kendallTau*)*, 13
kendallTau.matrix *(*kendallTau*)*, 13
kendallTau.paircomp *(*kendallTau*)*, 13
kendallTau.rankings *(*kendallTau*)*, 13
kendallTau_bootstrap *(*kendallTau*)*, 13
kendallTau_permute, *15*
kendallTau_permute *(*kendallTau*)*, 13
kendallW, *15*, 17, *26*
kenyachoice, 18

likelihood_ratio *(*anova.PlackettLuce*)*, 4

nicabean, 19, *31*
node_labels *(*plot.pltree*)*, 21
node_rules *(*plot.pltree*)*, 21

paircomp, *40*
pairwise_probs, 20
PlackettLuce, *24*
plot.PlackettLuce *(*plot.pltree*)*, 21
plot.pltree, 21
pltree, *11*
preference_index, 23
pseudo_rank, 24
pseudoR2, *15*, *17*, 25

rank_numeric, 27, *29*, *31*, *39, 40*
rank_tricot, *27*, 28, *31*, *39, 40*
rank_tricot2, *27*, *29*, 30, *39, 40*
rankings, *27*, *29*, *31*, *40*
regret, 32
reliability, 34
resample, 36
rowbind, 37

set_binomialfreq, *7*, *27*, *29*, *31*, 38, *40*
set_paircomp, *27*, *29*, *31*, 39, *39*

top_items *(*plot.pltree*)*, 21

worth_bar *(*worth_map*)*, 40
worth_map, *24*, 40