Package 'PlackettLuce'

September 16, 2019

Type Package

Title Plackett-Luce Models for Rankings

Version 0.2-9

URL https://hturner.github.io/PlackettLuce/

BugReports https://github.com/hturner/PlackettLuce/issues

Description Functions to prepare rankings data and fit the Plackett-Luce model jointly attributed to Plackett (1975) <doi:10.2307/2346567> and Luce (1959, ISBN:0486441369). The standard Plackett-Luce model is generalized to accommodate ties of any order in the ranking. Partial rankings, in which only a subset of items are ranked in each ranking, are also accommodated in the implementation. Disconnected/weakly connected networks implied by the rankings may be handled by adding pseudo-rankings with a hypothetical item. Optionally, a multivariate normal prior may be set on the log-worth parameters and ranker reliabilities may be incorporated as proposed by Raman and Joachims (2014) <doi:10.1145/2623330.2623654>. Maximum a posteriori estimation is used when priors are set. Methods are provided to estimate standard errors or quasi-standard errors for inference as well as to fit Plackett-Luce trees. See the package website or vignette for further details.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

Imports Matrix, igraph, methods, partykit, psychotools, psychotree, RSpectra, qvcalc, sandwich, stats

Suggests BiocStyle, BayesMallows, BradleyTerry2, BradleyTerryScalable, Matrix.utils, PLMIX, ROlogit, StatRank, covr, hyper2, kableExtra, knitr, lbfgs, gnm, pmr, rmarkdown, testthat

RoxygenNote 6.1.1

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Author Heather Turner [aut, cre] (<https://orcid.org/0000-0002-1256-3375>),
Ioannis Kosmidis [aut] (<https://orcid.org/0000-0003-1556-0302>),
David Firth [aut] (<https://orcid.org/0000-0003-0302-2312>),
Jacob van Etten [ctb] (<https://orcid.org/0000-0001-7554-2558>)

Maintainer Heather Turner <ht@heatherturner.net>

Repository CRAN

Date/Publication 2019-09-16 16:00:02 UTC

R topics documented:

`PlackettLuce-package` *Plackett-Luce Models for Rankings*

Description

Plackett-Luce provides functions to prepare rankings data in order to fit the Plackett-Luce model or Plackett-Luce trees. The implementation can handle ties, sub-rankings and rankings that imply disconnected or weakly connected preference networks. Methods are provided for summary and inference.

Details

The main function in the package is the model-fitting function PlackettLuce and the help file for that function provides details of the Plackett-Luce model, which is extended here to accommodate ties.

Rankings data must be passed to PlackettLuce in a specific form, see rankings for more details. Other functions for handling rankings include choices to express the rankings as choices from alternatives; adjacency to create an adjacency matrix of wins and losses implied by the rankings and connectivity to check the connectivity of the underlying preference network.

Several methods are available to inspect fitted Plackett-Luce models, help files are available for less common methods or where arguments may be specified: coef, deviance, fitted, itempar, logLik, print, qvcalc, summary, vcov.

PlackettLuce also provides the function pltree to fit a Plackett-Luce tree i.e. a tree that partitions the rankings by covariate values, identifying subgroups with different sets of worth parameters for the items. In this case group must be used to prepare the data.

Several data sets are provided in the package: beans, nascar, pudding. The help files for these give further illustration of preparing rankings data for modelling. The read.soc function enables further example data sets of "Strict Orders - Complete List" format (i.e. complete rankings with no ties) to be downloaded from PrefLib.

A full explanation of the methods with illustrations using the package data sets is given in the vignette, vignette("Overview",package = "PlackettLuce").

---

adjacency *Create an Adjacency Matrix for a set of Rankings*

Description

Convert a set of rankings to an adjacency matrix summarising wins and losses between pairs of items.

Usage

```
adjacency(object, weights = NULL, ...)
```

Arguments

object	a rankings object, or an object that can be coerced by as.rankings.
weights	an optional vector of weights for the rankings.
...	further arguments passed to/from methods.

Details

For a "rankings" object based on N items, the adjacency matrix is an N by N matrix, with element (i, j) being the number of times item i wins over item j. For example, in the ranking {1} > {3, 4} > {2}, item 1 wins over items 2, 3, and 4, and items 3 and 4 win over item 2.

If weights is specified, the values in the adjacency matrix are the weighted counts.

Value

An N by N matrix, where N is the number of items that can be ranked.

Examples

```
X <- matrix(c(2, 1, 2, 1, 2,
              3, 2, 0, 0, 1,
              1, 0, 2, 2, 3), nrow = 3, byrow = TRUE)
X <- as.rankings(X)
adjacency(X)

adjacency(X, weights = c(1, 1, 2))
```

---

aggregate *Aggregate Rankings*

Description

Aggregate rankings, returning an "aggregated_rankings" object of the unique rankings and their frequencies. The frequencies can be extracted via the function freq().

Usage

```
## S3 method for class 'rankings'
aggregate(x, freq = NULL, ...)

as.aggregated_rankings(x, ...)

## S3 method for class 'aggregated_rankings'
x[i, j, ..., drop = FALSE,
  as.aggregated_rankings = TRUE]

freq(x)
```

Arguments

x	A "rankings" object for aggregate(); an object that can be coerced to a "aggregated_rankings" object for as.aggregated_rankings(), otherwise an "aggregated_rankings" object.
freq	A vector of frequencies for rankings that have been previously aggregated.
...	Additional arguments, currently unused.
i	indices specifying rankings to extract, as for [.
j	indices specifying items to extract, as for [.
drop	if TRUE return single row/column matrices as a vector.

`as.aggregated_rankings`
: if `TRUE` create an `"aggregated_rankings"` object from the indexed rankings. Otherwise index the underlying matrix of ranks and return in a data frame with the corresponding frequencies.

Value

A data frame of class `"aggregated_rankings"`, with columns

`ranking`
: A `"rankings"` object of the unique rankings.

`freq`
: The corresponding frequencies.

Methods are available for `rbind()` and `as.matrix()`.

See Also

`preflib()` for an object that can be coerced to an `"aggregated_rankings"` object.

Examples

```
# create a rankings object with duplicated rankings
R <- matrix(c(1, 2, 0, 0,
              0, 1, 2, 3,
              2, 1, 1, 0,
              1, 2, 0, 0,
              2, 1, 1, 0,
              1, 0, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) <- c("apple", "banana", "orange", "pear")
R <- as.rankings(R)

# aggregate the rankings
A <- aggregate(R)

# subsetting applies to the rankings, e.g. first two unique rankings
A[1:2]

# (partial) rankings of items 2 to 4 only
A[, 2:4]

# convert to a matrix
as.matrix(A)

# frequencies are automatically used as weights by PlackettLuce()
mod <- PlackettLuce(A)
mod$weights
```

beans *Preferred Bean Varieties in Nicaragua*

Description

This is a subset of data from trials of bean varieties in Nicaragua over five growing seasons. Farmers were asked to try three varieties of bean from a total of ten varieties and to rank them in order of preference. In addition, for each variety the farmers were asked to compare each trial variety to the local variety and state whether they considered it to be better or worse.

Usage

```
beans
```

Format

A data frame with 842 records and 11 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.

`best` The variety the farmer ranked in first place ("A", "B" or "C").

`worst` The variety the farmer ranked in last place ("A", "B" or "C").

`var_a` How the farmer ranked variety A compared to the local variety ("Worse" or "Better").

`var_b` How the farmer ranked variety B compared to the local variety ("Worse" or "Better").

`var_c` How the farmer ranked variety C compared to the local variety ("Worse" or "Better").

`season` A factor specifying the growing season ("Po - 15", "Ap - 15", "Pr - 16", "Po - 16", "Ap - 16".

`year` The year of planting.

`maxTN` The maximum temperature at night during the vegetative cycle (degrees Celsius).

Details

There are three crop seasons in Central America:

Primera May - August.

Postrera September - October.

Apante November - January.

Beans can be planted near the beginning of each season, though are most commonly planted in the Postrera or Apante seasons.

Source

The data were provided by Bioversity International, a CGIAR research centre https://www.bioversityinternational.org.

Examples

```
# Consider the best and worst rankings. These give the variety the
# farmer thought was best or worst, coded as A, B or C for the
# first, second or third variety assigned to the farmer
# respectively.
data(beans)
head(beans[c("best", "worst")], 2)

# Fill in the missing item
beans$middle <- complete(beans[c("best", "worst")],
                         items = c("A", "B", "C"))
head(beans[c("best", "middle", "worst")], 2)

# This gives an ordering of the three varieties the farmer was
# given. The names of these varieties are stored in separate
# columns
varieties <- beans[c("variety_a", "variety_b", "variety_c")]
head(varieties, 2)

# Use these names to decode the orderings of order 3
order3 <- decode(beans[c("best", "middle", "worst")],
                 items = beans[c("variety_a", "variety_b", "variety_c")],
                 code = c("A", "B", "C"))

# Now consider the paired comparisons agains the local variety
head(beans[c("var_a", "var_b", "var_c")], 2)

# Convert these results to a vector and get the corresponding trial variety
outcome <- unlist(beans[c("var_a", "var_b", "var_c")])
trial_variety <- unlist(beans[c("variety_a", "variety_b", "variety_c")])

# Create a data frame of the implied orderings of order 2
order2 <- data.frame(Winner = ifelse(outcome == "Worse",
                                     "Local", trial_variety),
                     Loser = ifelse(outcome == "Worse",
                                    trial_variety, "Local"),
                     stringsAsFactors = FALSE, row.names = NULL)
head(order2, 2)

# Finally combine the rankings of order 2 and order 3
R <- rbind(as.rankings(order3, input = "orderings"),
           as.rankings(order2, input = "orderings"))
head(R)
tail(R)
```

`choices` *Choices Object*

Description

Convert a set of rankings to a list of choices, alternatives, and rankings. The choices and the corresponding alternatives make up the exchangeable part of the Plackett-Luce with ties.

Usage

```
choices(rankings, names = FALSE)
```

Arguments

rankings	a "rankings" object, or an object that can be coerced by as.rankings.
names	logical: if TRUE use the object names in the returned "choices" object, else use object indices.

Value

A data frame of class "choices" with elements:

choices	A list where each element represents the set of items chosen for a single rank in the ranking.
alternatives	A list where each element represents the set of items to choose from for a single rank in the ranking.
ranking	A list where each element represents the ranking that the choice belongs to.

The list stores the number of choices and the names of the objects as the attributes "nchoices" and "objects" respectively.

Examples

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

actual_choices <- choices(R, names = TRUE)
actual_choices[1:6,]

coded_choices <- choices(R, names = FALSE)
coded_choices[1:2,]
as.data.frame(coded_choices)[1:2,]
attr(coded_choices, "objects")
```

complete *Complete Orderings with the Missing Redundant Rank*

Description

Given orderings with one rank missing, complete the ordering by assigning the remaining item(s) to the final rank.

Usage

```
complete(orderings, items)
```

Arguments

orderings	A data frame of orderings with one rank missing.
items	A vector of item names.

Value

A vector of the missing items, which will be a list if there are any ties.

Examples

```
# Orderings of 3 items, when only the best and worst are recorded
orderings <- data.frame(best = c("A", "B", "A"),
                        worst = c("C", "C", NA))
orderings$middle <- complete(orderings, items = c("A", "B", "C"))
```

connectivity *Check Connectivity of Rankings*

Description

Check the connectivity of the network underlying a set of rankings.

Usage

```
connectivity(x, verbose = TRUE)
```

Arguments

x	an adjacency matrix as returned by adjacency, a "rankings" object, or an object that can be coerced by as.rankings.
verbose	logical, if TRUE, a message is given if the network is not strongly connected.

Details

Ranked items are connected in a directed graph according to the implied wins and loses between pairs of items. The wins and losses can be summarised as an adjacency matrix using adjacency. From this adjacency matrix, the graph is inferred and it is checked for connectivity. A message is given if the network is not strongly connected, i.e. with at least one win and one loss between all partitions of the network into two groups. Features of clusters in the network are returned - if the network is strongly connected, all items belong to the same cluster.

Value

A list with elements

membership	a labelled vector of indices specifying membership of clusters in the network of items
csize	the sizes of clusters in the network of items
no	the number of clusters in the network of items

Examples

```
## weakly connected network:
## one win between two clusters
X <- matrix(c(1, 2, 0, 0,
              2, 1, 3, 0,
              0, 0, 1, 2,
              0, 0, 2, 1), ncol = 4, byrow = TRUE)
X <- as.rankings(X)
res <- connectivity(X)
res$membership
## keep items in cluster 1
na.omit(X[,res$membership == 1])

## two weakly connected items:
## item 1 always loses; item 4 only wins against item 1
X <- matrix(c(4, 1, 2, 3,
              0, 2, 1, 3), nr = 2, byrow = TRUE)
X <- as.rankings(X)
res <- connectivity(X)
res$membership

## item 1 always wins; item 4 always loses
X <- matrix(c(1, 2, 3, 4,
              1, 3, 2, 4), nr = 2, byrow = TRUE)
res <- connectivity(as.rankings(X))
res$membership

## all in separate clusters: always 1 > 2 > 3 > 4
## also miscoded rankings and redundant ranking
X <- matrix(c(1, 2, 3, 4,
              1, 0, 2, 3,
              1, 1, 2, 0,
              1, 0, 3, 4,
```

```
              2, 2, 0, 4,
              0, 0, 3, 0,
              2, 4, 0, 0), ncol = 4, byrow = TRUE)
res <- connectivity(as.rankings(X))
res$membership
```

---

decode *Decode Orderings using a Key to Item Names*

Description

Decode orderings by replacing numeric or character coded values with item names.

Usage

```
decode(orderings, items, code = NULL)
```

Arguments

orderings	A data frame of coded orderings.
items	A data frame of the items in each ranking, or a vector of common items.
code	(Optional) a vector giving the key to the code. If missing, `names(items)` is used for a character code, while `seq(items)` is used for a numeric code.

Value

A data frame with the coded values replaced by the item names.

Examples

```
# orderings of up to 3 items coded as A, B, C
orderings <- data.frame(Rank1 = c("A", "B"),
                        Rank2 = c("C", "A"),
                        Rank3 = c("B", NA),
                        stringsAsFactors = FALSE)
items <- data.frame(A = c("banana", "apple"),
                    B = c("orange", "pear"),
                    C = c("apple", NA),
                    stringsAsFactors = FALSE)
decode(orderings, items)

# orderings with ties of up to 3 items, coded 1:3
orderings <- data.frame(Rank1 = c(1, 3),
                        Rank2 = I(list(c(2, 3), 2)),
                        Rank3 = c(NA, 1),
                        stringsAsFactors = FALSE)
items <- data.frame(A = c("banana", "apple"),
                    B = c("orange", "pear"),
```

```
                    C = c("apple", "orange"),
                    stringsAsFactors = FALSE)
decode(orderings, items)

# same items in each comparison
items <- c(A = "banana", B = "orange", C = "pear")
decode(orderings, items)
```

`fitted.PlackettLuce` *Fitted Probabilities for PlackettLuce Objects*

Description

Fitted probabilities for all choice/alternative combinations in the data.

Usage

```
## S3 method for class 'PlackettLuce'
fitted(object, aggregate = TRUE, free = TRUE,
  ...)

## S3 method for class 'pltree'
fitted(object, aggregate = TRUE, free = TRUE, ...)
```

Arguments

`object`	an object as returned by `PlackettLuce` or `pltree`.
`aggregate`	logical; if `TRUE` observations of the same choice from the same set of alternatives are aggregated.
`free`	logical; if `TRUE` only free choices are included, i.e. choices of one item from a set of one item are excluded.
`...`	further arguments, currently ignored.

Value

A list with the following components

`choices`	The selected item(s).
`alternatives`	The set of item(s) that the choice was made from.
`ranking`	The ranking(s) including this choice.
`n`	The weighted count of rankings including this choice (equal to the ranking weight if `aggregate = FALSE`.
`fitted`	The fitted probability of making this choice.

If `object` was a `"pltree"` object, the list has an additional element, `node`, specifying which node the ranking corresponds to.

See Also

choices

Examples

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

mod <- PlackettLuce(R)
fit <- fitted(mod)
fit
```

---

group *Group Rankings*

Description

Create an object of class "grouped_rankings" which associates a group index with an object of class "rankings". This allows the rankings to be linked to covariates with group-specific values as the basis for model-based recursive partitioning, see pltree.

Usage

```
group(x, index, ...)

as.grouped_rankings(x, ...)

## S3 method for class 'paircomp'
as.grouped_rankings(x, ...)

## S3 method for class 'grouped_rankings'
x[i, j, ..., drop = TRUE,
  as.grouped_rankings = TRUE]

## S3 method for class 'grouped_rankings'
format(x, max = 2L, width = 20L, ...)
```

Arguments

x	a "rankings" object for group(); an object that can be coerced to a "grouped_rankings" object for as.grouped_rankings(), otherwise a "grouped_rankings" object.
index	a numeric vector of length equal to the number of rankings specifying the subject for each ranking.

...	additional arguments passed on to as.rankings by grouped_rankings or as.grouped_rankings; unused by format.
i	indices specifying groups to extract, may be any data type accepted by [.
j	indices specifying items to extract, as for [.
drop	if TRUE return single row/column matrices as a vector.
as.grouped_rankings	if TRUE return a grouped_rankings object, otherwise return a matrix/vector.
max	the maximum number of rankings to format per subject.
width	the maximum width in number of characters to format each ranking.

Value

An object of class "grouped_rankings", which is a vector of of group IDs with the following attributes:

rankings	The "rankings" object.
index	An index match each ranking to each group ID.
R	A matrix with items ordered from last to first place, for each ranking.
S	The rankings matrix with the ranks replaced by the size of the chosen set for free choices and zero for forced choices.
id	A list with elements of the adjacency matrix that are incremented by each ranking.

See Also

pltree

Examples

```
# ungrouped rankings (5 rankings, 4 items)
R <- as.rankings(matrix(c(1, 2, 0, 0,
                          0, 2, 1, 0,
                          0, 0, 1, 2,
                          2, 1, 0, 0,
                          0, 1, 2, 3), ncol = 4, byrow = TRUE))
length(R)
R

# group rankings (first three in group 1, next two in group 2)
G <- group(R, c(1, 1, 1, 2, 2))
length(G)

## by default up to 2 rankings are shown per group, "..." indicates if
## there are further rankings
G
print(G, max = 1)
```

```
## select rankings from group 1
G[1,]

## exclude item 3 from ranking
G[, -3]

## rankings from group 2, excluding item 3
## - note group 2 becomes the first group
G[2, -3]

## index underlying rankings without creating new grouped_rankings object
G[2, -3, as.grouped_rankings = FALSE]
```

`itempar.PlackettLuce` *Extract Item Parameters of Plackett-Luce Models*

Description

Methods for `itempar` to extract the item parameters (abilities or log-abilities) from a Plackett-Luce model or tree. In the case of a tree, item parameters are extracted for each terminal node.

Usage

```
## S3 method for class 'PlackettLuce'
itempar(object, ref = NULL, alias = TRUE,
  vcov = TRUE, log = FALSE, ...)

## S3 method for class 'pltree'
itempar(object, ...)
```

Arguments

`object`	a fitted model object as returned by `PlackettLuce` or `pltree`.
`ref`	a vector of labels or position indices of item parameters which should be used as restriction/for normalization. If `NULL` (the default), all items are used with a zero sum (`log = TRUE`) or unit sum (`log = FALSE`) constraint.
`alias`	logical. If `TRUE` (the default), the aliased parameter is included in the return vector (and in the variance-covariance matrix if `vcov = TRUE`). If `FALSE`, it is removed. If the restriction given in ref depends on several parameters, the first parameter of the restriction specified is (arbitrarily) chosen to be removed if alias is `FALSE`.
`vcov`	logical. If `TRUE` (the default), the (transformed) variance-covariance matrix of the item parameters is attached as attribute `vcov`. If `FALSE`, a `NA`-matrix is attached.
`log`	logical. Whether to return log-abilities (`TRUE`) or abilities (`FALSE`).
`...`	further arguments which are currently not used.

Value

An object of class "itempar", see itempar.

Examples

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

mod <- PlackettLuce(R)
coef(mod)

# equivalent to default coefficients, i.e. log abilities
itempar(mod, ref= 1, log = TRUE)

# abilities, normalized so abilities for apple and pear sum to 1
itempar(mod, ref = 1:2)
```

---

nascar *Results from 2002 NASCAR Season*

Description

This is an example dataset from *Hunter 2004* recording the results of 36 car races in the 2002 NASCAR season in the United States. Each record is an ordering of the drivers according to their finishing position.

Usage

```
nascar
```

Format

A matrix with 36 rows corresponding to the races and 43 columns corresponding to the positions. The columns contain the ID for the driver that came first to last place respectively. The "drivers" attribute contains the names of the 87 drivers.

References

Hunter, D. R. (2004) MM algorithms for generalized Bradley-Terry models. *The Annals of Statistics*, **32(1)**, 384–406.

Examples

```
# convert orderings to rankings
nascar[1:2, ]
R <- as.rankings(nascar, input = "orderings", items = attr(nascar, "drivers"))
R[1:2, 1:4, as.rankings = FALSE]
format(R[1:2], width = 60)

# fit model as in Hunter 2004, excluding drivers that only lose
keep <- seq_len(83)
R2 <- R[, keep]
mod <- PlackettLuce(R2, npseudo = 0)

# show coefficients as in Table 2 of Hunter 2004
avRank <- apply(R, 2, function(x) mean(x[x > 0]))
coefs <- round(coef(mod)[order(avRank[keep])], 2)
head(coefs, 3)
tail(coefs, 3)
```

`PlackettLuce` *Fit a Plackett-Luce Model*

Description

Fit a Plackett-Luce model to a set of rankings. The rankings may be partial (each ranking completely ranks a subset of the items) and include ties of arbitrary order.

Usage

```
PlackettLuce(rankings, npseudo = 0.5, normal = NULL, gamma = NULL,
  adherence = NULL, weights = freq(rankings),
  na.action = getOption("na.action"), start = NULL,
  method = c("iterative scaling", "BFGS", "L-BFGS"), epsilon = 1e-07,
  steffensen = 0.1, maxit = c(500, 10), trace = FALSE,
  verbose = TRUE, ...)
```

Arguments

`rankings`	a `"rankings"` object, or an object that can be coerced by `as.rankings`. An `"aggregated_rankings"` object can be used to specify rankings and weights simultaneously. A `"grouped_rankings"` object should be used when estimating adherence for rankers with multiple rankings per ranker.
`npseudo`	when using pseudodata: the number of wins and losses to add between each object and a hypothetical reference object.
`normal`	a optional list with elements named `mu` and `Sigma` specifying the mean and covariance matrix of a multivariate normal prior on the *log* worths.

gamma	a optional list with elements named shape and rate specifying parameters of a gamma prior on adherence parameters for each ranker (use grouped_rankings to group multiple rankings by ranker). The short-cut TRUE may be used to specify a Gamma(10, 10) prior. If NULL (or FALSE), adherence is fixed to adherence for all rankers.
adherence	an optional vector of adherence values for each ranker. If missing, adherence is fixed to 1 for all rankers. If gamma is not NULL, this specifies the starting values for the adherence.
weights	an optional vector of weights for each ranking.
na.action	a function to handle any missing rankings, see na.omit().
start	starting values for the worth parameters and the tie parameters on the raw scale (worth parameters need not be scaled to sum to 1). If normal is specified, exp(normal$mu) is used as starting values for the worth parameters. Coefficients from a previous fit can be passed as the result of a call to coef.PlackettLuce, or the coefficients element of a "PlackettLuce" object.
method	the method to be used for fitting: "iterative scaling" (iterative scaling to sequentially update the parameter values), "BFGS" (the BFGS optimisation algorithm through the optim interface), "L-BFGS" (the limited-memory BFGS optimisation algorithm as implemented in the lbfgs package). Iterative scaling is used by default, unless a prior is specified by normal or gamma, in which case the default is "BFGS".
epsilon	the maximum absolute difference between the observed and expected sufficient statistics for the ability parameters at convergence.
steffensen	a threshold defined as for epsilon after which to apply Steffensen acceleration to the iterative scaling updates.
maxit	a vector specifying the maximum number of iterations. If gamma is NULL, only the first element is used and specifies the maximum number of iterations of the algorithm specified by method. If gamma is not NULL, a second element may be supplied to specify the maximum number of iterations of an alternating algorithm, where the adherence parameters are updated alternately with the other parameters. The default is to use 10 outer iterations.
trace	logical, if TRUE show trace of iterations.
verbose	logical, if TRUE show messages from validity checks on the rankings.
...	additional arguments passed to optim or lbfgs. In particular the convergence tolerance may be adjusted using e.g. control = list(reltol = 1e-10).

Value

An object of class "PlackettLuce", which is a list containing the following elements:

call	The matched call.
coefficients	The model coefficients.
loglik	The maximized log-likelihood.
null.loglik	The maximized log-likelihood for the null model (all alternatives including ties have equal probability).

`df.residual`	The residual degrees of freedom.
`df.null`	The residual degrees of freedom for the null model.
`rank`	The rank of the model.
`logposterior`	If a prior was specified, the maximised log posterior.
`gamma`	If a gamma prior was specified, the list of parameters.
`normal`	If a normal prior was specified, the list of parameters.
`iter`	The number of iterations run.
`rankings`	The rankings passed to `rankings`, converted to a `"rankings"` object if necessary.
`weights`	The weights applied to each ranking in the fitting.
`adherence`	The fixed or estimated adherence per ranker.
`ranker`	The ranker index mapping rankings to rankers (the `"index"` attribute of `rankings` if specified as a `"grouped_rankings"` object.)
`maxTied`	The maximum number of objects observed in a tie.
`conv`	The convergence code: 0 for successful convergence; 1 if reached `maxit` (outer) iterations without convergence; 2 if Steffensen acceleration cause log-likelihood to increase; negative number if L-BFGS algorithm failed for other reason.

Model definition

A single ranking is given by

$$R = \{C_1, C_2, \ldots, C_J\}$$

where the items in set C_1 are ranked higher than (better than) the items in C_2, and so on. If there are multiple objects in set C_j these items are tied in the ranking.

For a set if items S, let

$$f(S) = \delta_{|S|} \left(\prod_{i \in S} \alpha_i \right)^{\frac{1}{|S|}}$$

where $|S|$ is the cardinality (size) of the set, δ_n is a parameter related to the prevalence of ties of order n (with $\delta_1 \equiv 1$), and α_i is a parameter representing the worth of item i. Then under an extension of the Plackett-Luce model allowing ties up to order D, the probability of the ranking R is given by

$$\prod_{j=1}^{J} \frac{f(C_j)}{\sum_{k=1}^{\min(D_j, D)} \sum_{S \in \binom{A_j}{k}} f(S)}$$

where D_j is the cardinality of A_j, the set of alternatives from which C_j is chosen, and $\binom{A_j}{k}$ is all the possible choices of k items from A_j. The value of D can be set to the maximum number of tied items observed in the data, so that $\delta_n = 0$ for $n > D$.

When the worth parameters are constrained to sum to one, they represent the probability that the corresponding item comes first in a ranking of all items, given that first place is not tied.

The 2-way tie prevalence parameter δ_2 is related to the probability that two items *of equal worth* tie for first place, given that the first place is not a 3-way or higher tie. Specifically, that probability is $\delta_2/(2 + \delta_2)$.

The 3-way and higher tie-prevalence parameters are similarly interpretable, in terms of tie probabilities among equal-worth items.

Pseudo-rankings

In order for the maximum likelihood estimate of an object's worth to be defined, the network of rankings must be strongly connected. This means that in every possible partition of the objects into two nonempty subsets, some object in the second set is ranked higher than some object in the first set at least once.

If the network of rankings is not strongly connected then pseudo-rankings may be used to connect the network. This approach posits a hypothetical object with log-worth 0 and adds `npseudo` wins and `npseudo` losses to the set of rankings.

The parameter `npseudo` is the prior strength. With `npseudo = 0` the MLE is the posterior mode. As `npseudo` approaches infinity the log-worth estimates all shrink towards 0. The default, `npseudo = 0.5`, is sufficient to connect the network and has a weak shrinkage effect. Even for networks that are already connected, adding pseudo-rankings typically reduces both the bias and variance of the estimators of the worth parameters.

Incorporating prior information on log-worths

Prior information can be incorporated by using `normal` to specify a multivariate normal prior on the *log*-worths. The log-worths are then estimated by maximum a posteriori (MAP) estimation. Model summaries (deviance, AIC, standard errors) are based on the log-likelihood evaluated at the MAP estimates, resulting in a finite sample bias that should disappear as the number of rankings increases. Inference based on these model summaries is valid as long as the prior is considered fixed and not tuned as part of the model.

Incorporating a prior is an alternative method of penalization, therefore `npseudo` is set to zero when a prior is specified.

Incorporating ranker adherence parameters

When rankings come from different rankers, the model can be extended to allow for varying reliability of the rankers, as proposed by Raman and Joachims (2014). In particular, replacing $f(S)$ by

$$h(S) = \delta_{|S|} \left(\prod_{i \in S} \alpha_i \right)^{\frac{\eta_g}{|S|}}$$

where $\eta_g > 0$ is the adherence parameter for ranker g. In the standard model, all rankers are assumed to have equal reliability, so $\eta_g = 1$ for all rankers. Higher $\eta_g = 1$ increases the distance between item worths, giving greater weight to the ranker's choice. Conversely, lower $\eta_g = 1$ shrinks the item worths towards equality so the ranker's choice is less relevant.

The adherence parameters are not estimable by maximum likelihood, since for given item worths the maximum likelihood estimate of adherence would be infinity for rankers that give rankings consistent with the items ordered by worth and zero for all other rankers. Therefore it is essential to include a prior on the adherence parameters when these are estimated rather than fixed. Setting `gamma = TRUE` specifies the default $\Gamma(10, 10)$ prior, which has a mean of 1 and a probability of 0.99 that the adherence is between 0.37 and 2. Alternative parameters can be specified by a list with elements `shape` and `rate`. Setting scale and rate to a common value θ specifies a mean of 1; $\theta \geq 2$ will give low prior probability to near-zero adherence; as θ increases the density becomes more concentrated (and more symmetrical) about 1.

Since the number of adherence parameters will typically be large and it is assumed the worth and tie parameters are of primary interest, the adherence parameters are not included in model summaries, but are included in the returned object.

Controlling the fit

For models without priors, using `nspseudo = 0` will use standard maximum likelihood, if the network is connected (and throw an error otherwise).

The fitting algorithm is set by the `method` argument. The default method `"iterative scaling"` is a slow but reliable approach. In addition, this has the most control on the accuracy of the final fit, since convergence is determined by direct comparison of the observed and expected values of the sufficient statistics for the worth parameters, rather than a tolerance on change in the log-likelihood.

The `"iterative scaling"` algorithm is slow because it is a first order method (does not use derivatives of the likelihood). From a set of starting values that are 'close enough' to the final solution, the algorithm can be accelerated using Steffensen's method. `PlackettLuce` attempts to apply Steffensen's acceleration when all differences between the observed and expected values of the sufficient statistics are less than `steffensen`. This is an ad-hoc rule defining 'close enough' and in some cases the acceleration may produce negative worth parameters or decrease the log-likelihood. `PlackettLuce` will only apply the update when it makes an improvement.

The `"BFGS"` and `"L-BFGS"` algorithms are second order methods, therefore can be quicker than the default method. Control parameters can be passed on to `optim` or `lbfgs`.

For models with priors, the iterative scaling method cannot be used, so BFGS is used by default.

Note

As the maximum tie order increases, the number of possible choices for each rank increases rapidly, particularly when the total number of items is high. This means that the model will be slower to fit with higher D. In addition, due to the current implementation of the `vcov()` method, computation of the standard errors (as by `summary()`) can take almost as long as the model fit and may even become infeasible due to memory limits. As a rule of thumb, for > 10 items and > 1000 rankings, we recommend `PlackettLuce()` for ties up to order 4. For higher order ties, a rank-ordered logit model, see `ROlogit::rologit()` or generalized Mallows Model as in `BayesMallows::compute_mallows()` may be more suitable, as they do not model tied events explicitly.

References

Raman, K. and Joachims, T. (2014) Methods for Ordinal Peer Grading. arXiv:1404.3656.

See Also

Handling rankings: `rankings`, `aggregate`, `group`, `choices`, `adjacency`, `connectivity`.

Inspect fitted Plackett-Luce models: `coef`, `deviance`, `fitted`, `itempar`, `logLik`, `print`, `qvcalc`, `summary`, `vcov`.

Fit Plackett-Luce tree: `pltree`.

Example data sets: `beans`, `nascar`, `pudding`, `preflib`.

Vignette: `vignette("Overview",package = "PlackettLuce")`.

Examples

```
# Six partial rankings of four objects, 1 is top rank, e.g
# first ranking: item 1, item 2
# second ranking: item 2, item 3, item 4, item 1
# third ranking: items 2, 3, 4 tie for first place, item 1 second
R <- matrix(c(1, 2, 0, 0,
              4, 1, 2, 3,
              2, 1, 1, 1,
              1, 2, 3, 0,
              2, 1, 1, 0,
              1, 0, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) <- c("apple", "banana", "orange", "pear")

# create rankings object
R <- as.rankings(R)

# Standard maximum likelihood estimates
mod_mle <- PlackettLuce(R, npseudo = 0)
coef(mod_mle)

# Fit with default settings
mod <- PlackettLuce(R)
# log-worths are shrunk towards zero
coef(mod)

# independent N(0, 9) priors on log-worths, as in Raman and Joachims
prior <- list(mu = rep(0, ncol(R)),
              Sigma = diag(rep(9, ncol(R))))
mod_normal <- PlackettLuce(rankings = R, normal = prior)
# slightly weaker shrinkage effect vs pseudo-rankings,
# with less effect on tie parameters (but note small number of rankings here)
coef(mod_normal)

# estimate adherence assuming every ranking is from a separate ranker
mod_separate <- PlackettLuce(rankings = R, normal = prior, gamma = TRUE)
coef(mod_separate)
# gives more weight to rankers 4 & 6 which rank apple first,
# so worth of apple increased relative to banana
mod_separate$adherence

# estimate adherence based on grouped rankings
#  - assume two rankings from each ranker
G <- group(R, rep(1:3, each = 2))
mod_grouped <- PlackettLuce(rankings = G, normal = prior, gamma = TRUE)
coef(mod_grouped)
# first ranker is least consistent so down-weighted
mod_grouped$adherence
```

---

pltree *Plackett-Luce Trees*

Description

Recursive partitioning based on Plackett-Luce models.

Usage

```
pltree(formula, data, subset, na.action, cluster, ref = NULL, ...)
```

Arguments

formula	a symbolic description of the model to be fitted, of the form y ~ x1 + ... + xn where y should be an object of class grouped_rankings and x1, ..., xn are used as partitioning variables.
data	an optional data frame containing the variables in the model.
subset	A specification of the rows to be used, passed to model.frame.
na.action	how NAs are treated, applied to the underlying rankings and then passed to model.frame.
cluster	an optional vector of cluster IDs to be employed for clustered covariances in the parameter stability tests, see mob.
ref	an integer or character string specifying the reference item (for which log ability will be set to zero). If NULL the first item is used.
...	additional arguments, passed to PlackettLuce.

Details

Plackett-Luce trees are an application of model-based recursive partitioning (implemented in mob) to Plackett-Luce models for rankings. The partitioning is based on ranking covariates, e.g. attributes of the judge making the ranking, or conditions under which the ranking is made. The response should be a grouped_rankings object that groups rankings with common covariate values. This may be included in a data frame alongside the covariates.

Most arguments of PlackettLuce can be passed on by pltree. However, Plackett-Luce tree with fixed adherence are not implemented. Arguably it makes more sense to estimate adherence or reliability within the nodes of the Plackett-Luce tree.

Various methods are provided for "pltree" objects, most of them inherited from "modelparty" objects (e.g. print, summary), or "bttree" objects (plot). The plot method employs the node_btplot panel-generating function. The See Also section gives details of separately documented methods.

Value

An object of class "pltree" inheriting from "bttree" and "modelparty".

See Also

bttree For fitting Bradley-Terry trees (equivalent to the Plackett-Luce model for paired comparisons without ties).

coef, vcov, AIC and predict methods are documented on pltree-summaries.

itempar, extracts the abilities or item parameters in each node of the tree using itempar.PlackettLuce.

fitted, computes probabilities for the observed choices based on the full tree.

Examples

```
# Bradley-Terry example

if (require(psychotree)){
    ## Germany's Next Topmodel 2007 data
    data("Topmodel2007", package = "psychotree")
    ## convert paircomp object to grouped rankings
    R <- as.grouped_rankings(Topmodel2007$preference)
    ## rankings are grouped by judge
    print(R[1:2,], max = 4)
    ## Topmodel2007[, -1] gives covariate values for each judge
    print(Topmodel2007[1:2, -1])

    ## fit partition model based on all variables except preference
    ## set npseudo = 0 as all judges rank all models
    tm_tree <- pltree(R ~ ., data = Topmodel2007[, -1], minsize = 5,
                      npseudo = 0)

    ## plot shows abilities constrained to sum to 1
    plot(tm_tree, abbreviate = 1, yscale = c(0, 0.5))
    ## instead show log-abilities with Anja as reference (need to used index)
    plot(tm_tree, abbreviate = 1, worth = FALSE, ref = 6,
         yscale = c(-1.5, 2.2))

    ## log-abilities, zero sum contrast
    itempar(tm_tree, log = TRUE)
}
```

`pltree-summaries` *Plackett-Luce Tree Summaries*

Description

Obtain the coefficients, variance-covariance matrix, AIC, or predictions from a Plackett-Luce tree fitted by `pltree()`.

Usage

```
## S3 method for class 'pltree'
coef(object, node = NULL, drop = TRUE, ...)

## S3 method for class 'pltree'
vcov(object, node = nodeids(object, terminal = TRUE),
  ...)

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

## S3 method for class 'pltree'
```

```
predict(object, newdata = NULL, type = c("itempar",
  "rank", "best", "node"), ...)
```

Arguments

object	a fitted model object of class "pltree".
node	a vector of node ids specifying the nodes to summarise, by default the ids of the terminal nodes.
drop	if TRUE return the coefficients as a vector when only one node is selected.
...	additional arguments passed to itempar by predict, and to model.frame by AIC.
newdata	an optional data frame to use instead of the original data. For AIC this must include the response variable.
type	the type of prediction to return for each group, one of: "itempar" to give the result of itempar (by default the fitted probability of each item being ranked first out of all objects), "rank" the corresponding rank, "best" the topped ranked item, or "node" the node of the tree the group belongs to.

Details

AIC computes $-2L + 2p$ where L is the joint likelihood of the observed rankings under the tree model and p is the degrees of freedom used to fit the tree model.

Examples

```
data(beans)
# fit tree based on pairwise comparisons with variety B
pairB <- data.frame(Winner = ifelse(beans$var_b == "Worse",
                                    "Local", beans$variety_b),
                    Loser = ifelse(beans$var_b == "Worse",
                                   beans$variety_b, "Local"),
                    stringsAsFactors = FALSE, row.names = NULL)
beans$G <- as.rankings(pairB, input = "orderings",
                       index = rep(seq(nrow(beans)), 1))

mod <- pltree(G ~ ., data = beans[c("G", "maxTN")])

coef(mod, node = 3)
AIC(mod)

# treat first row from each year as new data
newdata <- beans[!duplicated(beans$year),]

## fitted probabilities
predict(mod, newdata)

## fitted log-abilities, with Local as reference
predict(mod, newdata, log = TRUE, ref = "Local")

## variety ranks
```

```
predict(mod, newdata, type = "rank")

## top ranked variety
predict(mod, newdata, type = "best")

## node the trial belongs to
predict(mod, newdata, type = "node")
```

`preflib` *Read Preflib Election Data Files*

Description

Read orderings from `.soc`, `.soi`, `.toc` or `.toi` file types storing election data as defined by PrefLib: A Library for Preferences.

Usage

```
read.soc(file)

read.soi(file)

read.toc(file)

read.toi(file)

## S3 method for class 'preflib'
as.aggregated_rankings(x, ...)
```

Arguments

`file`	An election data file, conventionally with extension `.soc`, `.soi`, `.toc` or `.toi` according to data type.
`x`	An object of class `"preflib"`.
`...`	Additional arguments passed to `as.rankings()`: `freq`, `input` or `items` will be ignored with a warning as they are set automatically.

Details

The file types supported are

.soc Strict Orders - Complete List

.soi Strict Orders - Incomplete List

.toc Orders with Ties - Complete List

.toi Orders with Ties - Incomplete List

Note that the file types do not distinguish between types of incomplete orderings, i.e. whether they are a complete ranking of a subset of items (as supported by `PlackettLuce()`) or top-n rankings of n items from the full set of items (not currently supported by `PlackettLuce()`).

The numerically coded orderings and their frequencies are read into a data frame, storing the item names as an attribute. The `as.aggregated_rankings` method converts these to an `"aggregated_rankings"` object with the items labelled by the item names.

A Preflib file may be corrupt, in the sense that the ordered items do not match the named items. In this case, the file can be read is as a data frame (with a warning) using the corresponding `read.*` function, but `as.aggregated_rankings` will throw an error.

Value

A data frame of class `"preflib"` with first column `Freq`, giving the frequency of the ranking in that row, and remaining columns `Rank 1`, ..., `Rank p` giving the items ranked from first to last place in that ranking. Ties are represented by vector elements in list columns. The data frame has an attribute `"items"` giving the labels corresponding to each item number.

Note

The Netflix and cities datasets used in the examples are from Caragiannis et al (2017) and Bennet and Lanning (2007) respectively. These data sets require a citation for re-use.

References

Mattei, N. and Walsh, T. (2013) PrefLib: A Library of Preference Data. *Proceedings of Third International Conference on Algorithmic Decision Theory (ADT 2013)*. Lecture Notes in Artificial Intelligence, Springer.

Caragiannis, I., Chatzigeorgiou, X, Krimpas, G. A., and Voudouris, A. A. (2017) Optimizing positional scoring rules for rank aggregation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.

Bennett, J. and Lanning, S. (2007) The Netflix Prize. *Proceedings of The KDD Cup and Workshops*.

Examples

```
# can take a little while depending on speed of internet connection

## Not run:
# url for preflib data in the "Election Data" category
preflib <- "http://www.preflib.org/data/election/"

# strict complete orderings of four films on Netflix
netflix <- read.soc(file.path(preflib, "netflix/ED-00004-00000101.soc"))
head(netflix)
attr(netflix, "items")

head(as.rankings(netflix))

# strict incomplete orderings of 6 random cities from 36 in total
```

```
cities <- read.soi(file.path(preflib, "cities/ED-00034-00000001.soi"))

# strict incomplete orderings of drivers in the 1961 F1 races
# 8 races with 17 to 34 drivers in each
f1 <- read.soi(file.path(preflib, "f1/ED-00010-00000001.soi"))

# complete orderings with ties of 30 skaters
skaters <- read.toc(file.path(preflib, "skate/ED-00006-00000001.toc"))

# incomplete orderings with ties of 10 sushi items from 100 total
# orderings were derived from numeric ratings
sushi <- read.toi(file.path(preflib, "sushi/ED-00014-00000003.toi"))

## End(Not run)
```

---

pudding *Paired Comparisons of Chocolate Pudding*

Description

This is an example dataset from *Davidson 1970* comprising paired comparisons of chocolate pudding, with six brands in total. The responses include tied outcomes, i.e. no preference.

Usage

```
pudding
```

Format

A data frame with 15 records and 6 variables:

`i` The first brand in the comparison.

`j` The second brand in the comparison.

`r_ij` The frequency of paired comparisons of brand i and brand j.

`w_ij` The frequency of preferences for i over j.

`w_ji` The frequency of preferences for j over i.

`t_ij` The frequency of no preference between i and j.

References

Davidson, R. R. (1970). On extending the Bradley-Terry model to accommodate ties in paired comparison experiments. *Journal of the American Statistical Association*, **65**, 317–328.

Examples

```
# create orderings for each set of paired comparisons

# wins for brand i and wins for brand j
i_wins <- data.frame(Winner = pudding$i, Loser = pudding$j)
j_wins <- data.frame(Winner = pudding$j, Loser = pudding$i)

# ties: use an array list (easier with R >= 3.6.0)
if (getRversion() < "3.6.0"){
  n <- nrow(pudding)
  ties <- data.frame(Winner = array(split(pudding[c("i", "j")], 1:n), n),
                     Loser = rep(NA, 15))
} else {
  ties <- data.frame(Winner = asplit(pudding[c("i", "j")], 1),
                     Loser = rep(NA, 15))
}
head(ties, 2)

# convert to rankings
R <- as.rankings(rbind(i_wins, j_wins, ties),
                 input = "orderings")
head(R, 2)
tail(R, 2)

# define weights as frequencies of each ranking
w <- unlist(pudding[c("w_ij", "w_ji", "t_ij")])

# fit Plackett-Luce model: limit iterations to match paper
mod <- PlackettLuce(R, npseudo = 0, weights = w, maxit = 7)
```

`qvcalc.PlackettLuce` *Quasi Variances for Model Coefficients*

Description

A method for `qvcalc` to compute a set of quasi variances (and corresponding quasi standard errors) for estimated item parameters from a Plackett-Luce model.

Usage

```
## S3 method for class 'PlackettLuce'
qvcalc(object, ref = 1L, ...)
```

Arguments

`object`	a `"PlackettLuce"` object as returned by `PlackettLuce`.
`ref`	An integer or character string specifying the reference item (for which log worth will be set to zero). If `NULL` the sum of the log worth parameters is set to zero.
`...`	additional arguments, currently ignored..

Details

For details of the method see Firth (2000), Firth (2003) or Firth and de Menezes (2004). Quasi variances generalize and improve the accuracy of "floating absolute risk" (Easton et al., 1991). This device for economical model summary was first suggested by Ridout (1989).

Ordinarily the quasi variances are positive and so their square roots (the quasi standard errors) exist and can be used in plots, etc.

Value

A list of class `"qv"`, with components

`covmat`	The full variance-covariance matrix for the item parameters.
`qvframe`	A data frame with variables `estimate`, `SE`, `quasiSE` and `quasiVar`, the last two being a quasi standard error and quasi-variance for each parameter.
`dispersion`	`NULL` (dispersion is fixed to 1).
`relerrs`	Relative errors for approximating the standard errors of all simple contrasts.
`factorname`	`NULL` (not required for this method).
`coef.indices`	`NULL` (not required for this method).
`modelcall`	The call to `PlackettLuce` to fit the model from which the item parameters were estimated.

References

Easton, D. F, Peto, J. and Babiker, A. G. A. G. (1991) Floating absolute risk: an alternative to relative risk in survival and case-control analysis avoiding an arbitrary reference group. *Statistics in Medicine* **10**, 1025–1035.

Firth, D. (2000) Quasi-variances in Xlisp-Stat and on the web. *Journal of Statistical Software* **5.4**, 1–13. At https://www.jstatsoft.org

Firth, D. (2003) Overcoming the reference category problem in the presentation of statistical models. *Sociological Methodology* **33**, 1–18.

Firth, D. and de Menezes, R. X. (2004) Quasi-variances. *Biometrika* **91**, 65–80.

Menezes, R. X. de (1999) More useful standard errors for group and factor effects in generalized linear models. *D.Phil. Thesis*, Department of Statistics, University of Oxford.

Ridout, M.S. (1989). Summarizing the results of fitting generalized linear models to data from designed experiments. In: *Statistical Modelling: Proceedings of GLIM89 and the 4th International Workshop on Statistical Modelling held in Trento, Italy, July 17–21, 1989* (A. Decarli et al., eds.), pp 262–269. New York: Springer.

See Also

`worstErrors`, `plot.qv`.

Examples

```
# Six partial rankings of four objects, 1 is top rank, e.g
# first ranking: item 1, item 2
# second ranking: item 2, item 3, item 4, item 1
# third ranking: items 2, 3, 4 tie for first place, item 1 second
R <- matrix(c(1, 2, 0, 0,
              4, 1, 2, 3,
              2, 1, 1, 1,
              1, 2, 3, 0,
              2, 1, 1, 0,
              1, 0, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) <- c("apple", "banana", "orange", "pear")

mod <- PlackettLuce(R)
qv <- qvcalc(mod)
qv
plot(qv)
```

---

rankings *Rankings Object*

Description

Create a "rankings" object from data or convert a matrix of rankings or ordered items to a "rankings" object.

Usage

```
rankings(data, id, item, rank, aggregate = FALSE, verbose = TRUE, ...)

as.rankings(x, ..., verbose = TRUE)

## Default S3 method:
as.rankings(x, input = c("rankings", "orderings"),
  freq = NULL, index = NULL, aggregate = FALSE, items = NULL,
  labels = NULL, ..., verbose = TRUE)

## S3 method for class 'matrix'
as.rankings(x, input = c("rankings", "orderings"),
  freq = NULL, index = NULL, aggregate = FALSE, items = NULL,
  labels = NULL, ..., verbose = TRUE)

## S3 method for class 'rankings'
x[i, j, ..., drop = TRUE, as.rankings = TRUE]

## S3 method for class 'rankings'
format(x, width = 40L, ...)
```

Arguments

data	a data frame with columns specified by id, item and rank.
id	an index of data specifying the column containing ranking IDs.
item	an index of data specifying the column containing item IDs,
rank	an index of data specifying the column containing item ranks.
aggregate	if TRUE, aggregate the rankings via aggregate() before returning.
verbose	logical; if TRUE print messages when changes are made to rankings data.
...	further arguments passed to/from methods.
x	for as.rankings, a matrix with one column per item and one row per ranking, or an object that can be coerced to such as matrix; for [and format, a "rankings" object.
input	for as.rankings, whether rows in the input matrix contain numeric "rankings" (dense, standard/modified competition or fractional rankings) or "orderings", i.e. the items ordered by rank.
freq	an optional column index (number, character or logical) specifying a column of x that holds ranking frequencies, or a vector of ranking frequencies. If provided, an "aggregated_rankings" object will be returned.
index	an optional column index (number, character or logical) specifying a column of x that holds a grouping index, or a numeric vector to for grouping. If provided, the rankings will be grouped by group() before returning.
items	for input = "orderings", a character vector specifying the full set of items. Values in x are matched to this by value (if character) or position (if numeric). Use decode() for orderings requiring more complex decoding.
labels	for input = "orderings" an optional vector of labels for the items, corresponding to the sorted unique values of x.
i	indices specifying rankings to extract, as for [.
j	indices specifying items to extract, as for [.
drop	if TRUE return single row/column matrices as a vector.
as.rankings	if TRUE return a rankings object, otherwise return a matrix/vector.
width	the width in number of characters to format each ranking - rankings that are too wide will be truncated.

Details

Each ranking in the input data will be converted to a dense ranking, which rank items from 1 (first place) to n_r (last place). Items not ranked should have a rank of 0 or NA. Tied items are given the same rank with no rank skipped. For example 1, 0, 2, 1, ranks the first and fourth items in first place and the third item in second place; the second item is unranked.

Records in data with missing id or item are dropped. Duplicated items in the rankings are resolved if possible: redundant or inconsistent ranks are set to NA. Rankings with only 1 item are set to NA (rankings with zero items are automatically treated as NA). Any issues causing records to be removed or recoded produce a message if verbose = TRUE.

For as.rankings with input = "orderings", unused ranks may be filled with zeroes for numeric x or NA. It is only necessary to have as many columns as ranks that are used.

The method for [will return a reduced rankings object by default, recoding as dense rankings and setting invalid rankings to NA as necessary. To extract rows and/or columns of the rankings as a matrix or vector, set as.rankings = FALSE, see examples.

Value

By default, a "rankings" object, which is a matrix of dense rankings with methods for several generics including aggregate, [, format, rbind() and as.matrix().

If the object is created with aggregate = TRUE, or ranking frequencies are specified via freq, the rankings are post-processed to create an "aggregated_rankings" object.

If a group index is specified via index, the (possibly aggregated) rankings are post-processed to create a "grouped_rankings" object.

Examples

```
# create rankings from data in long form

# example long form data
x <- data.frame(ranking = c(rep(1:4, each = 4), 5, 5, 5),
                letter = c(LETTERS[c(1:3, 3, 1:4, 2:5, 1:2, 1)], NA,
                           LETTERS[3:5]),
                rank = c(4:1, rep(NA, 4), 3:4, NA, NA, 1, 3, 4, 2, 2, 2, 3))

# ranking 1 has different rank for same item, but order of items unambiguous
# all ranks are missing in ranking 2
# some ranks are missing in ranking 3
# ranking 4 has inconsistent ranks for two items and a rank with missing item
# ranking 5 is fine - an example of a tie
split(x, x$ranking)

# fix issues when creating rankings object
rankings(x, id = "ranking", item = "letter", rank = "rank")

# convert existing matrix of rankings

R <- matrix(c(1, 2, 0, 0,
              4, 1, 2, 3,
              2, 1, 1, 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)

# first three rankings
R[1:3,]

# exclude pear from the rankings
R[, -4]
```

```
# extract rankings 2 and 3 as numeric matrix
R[2:3, , as.rankings = FALSE]

# same as
as.matrix(R)[2:3,]

# extract rankings for item 1 as a vector
R[,1, as.rankings = FALSE]
```

`simulate.PlackettLuce` *Simulate from* `PlackettLuce` *fitted objects*

Description

Simulate from `PlackettLuce` fitted objects

Usage

```
## S3 method for class 'PlackettLuce'
simulate(object, nsim = 1L, seed = NULL,
  multinomial = FALSE, max_combinations = 20000, ...)
```

Arguments

`object`	an object representing a fitted model.
`nsim`	number of response vectors to simulate. Defaults to `1`.
`seed`	an object specifying if and how the random number generator should be initialised. Either `NULL` or an integer that will be used in a call to `set.seed` before simulating the rankings. If set, the value is saved as the `seed` attribute of the returned value. The default, `NULL`, will not change the random generator state, and return `.Random.seed` as the `seed` attribute.
`multinomial`	use multinomial sampling anyway? Default is `FALSE`. see Details.
`max_combinations`	a positive number. Default is `20000`. See Details.
`...`	additional optional arguments.

Details

If `multinomial` is `FALSE` (default) and there are no tie parameters in the object (i.e. `object$maxTied == 1`), then rankings are sampled by ordering exponential random variates with rate 1 scaled by the estimated item-worth parameters `object$coefficients` (see, Diaconis, 1988, Chapter 9D for details).

In all other cases, the current implementation uses direct multinomial sampling, and will throw an error if there are more than `max_combinations` combinations of items that the sampler has to

decide from. This is a hard-coded exit to prevent issues relating to the creation of massive objects in memory.

If `object$maxTied > 1` the user's setting for `multinomial` is ignored and `simulate.PlackettLuce` operates as if `multinomial` is `TRUE`.

Value

A `data.frame` of `rankings` objects of the same dimension as `object$rankings`.

References

Diaconis (1988). *Group Representations in Probability and Statistics*. Institute of Mathematical Statistics Lecture Notes 11. Hayward, CA.

Examples

```
R <- matrix(c(1, 2, 0, 0,
              4, 1, 2, 3,
              2, 1, 1, 1,
              1, 2, 3, 0,
              2, 1, 1, 0,
              1, 0, 3, 2), nrow = 6, byrow = TRUE)
colnames(R) <- c("apple", "banana", "orange", "pear")
mod <- PlackettLuce(R)
simulate(mod, 5)

s1 <- simulate(mod, 3, seed = 112)
s2 <- simulate(mod, 2, seed = 112)

identical(s1[1:2], s2[1:2])
```

---

summaries *Plackett-Luce Model Summaries*

Description

Obtain the coefficients, model summary or coefficient variance-covariance matrix for a model fitted by `PlackettLuce`.

Usage

```
## S3 method for class 'PlackettLuce'
coef(object, ref = 1L, log = TRUE,
  type = "all", ...)

## S3 method for class 'PlackettLuce'
summary(object, ref = 1L, ...)
```

```
## S3 method for class 'PlackettLuce'
vcov(object, ref = 1L, type = c("expected",
  "observed"), ...)
```

Arguments

object	An object of class "PlackettLuce" as returned by PlackettLuce.
ref	An integer or character string specifying the reference item (for which log worth will be set to zero). If NULL the sum of the log worth parameters is set to zero.
log	A logical indicating whether to return parameters on the log scale with the item specified by ref set to zero.
type	For coef, the type of coefficients to return: one of "ties", "worth" or "all". For vcov, the type of Fisher information to base the estimation on: either "expected" or "observed".
...	additional arguments, passed to vcov by summary.

Details

By default, parameters are returned on the log scale, as most suited for inference. If log = FALSE, the worth parameters are returned, constrained to sum to one so that they represent the probability that the corresponding item comes first in a ranking of all items, given that first place is not tied.

The variance-covariance matrix is returned for the worth and tie parameters on the log scale, with the reference as specified by ref. For models estimated by maximum likelihood, the variance-covariance is the inverse of the Fisher information of the log-likelihood.

For models with a normal or gamma prior, the variance-covariance is based on the Fisher information of the log-posterior. When adherence parameters have been estimated, the log-posterior is not linear in the parameters. In this case there is a difference between the expected and observed Fisher information. By default, vcov will return the variance-covariance based on the expected information, but type gives to option to use the observed information instead. For large samples, the difference between these options should be small. Note that the estimation of the adherence parameters is accounted for in the computation of the variance-covariance matrix, but only the sub-matrix corresponding to the worth and tie parameters is estimated.

Index