Package 'PlackettLuce'

April 9, 2018

Type Package

Title Plackett-Luce Models for Rankings

Version 0.2-3

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 are handled by adding pseudo-rankings with a hypothetical item. 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 full details.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

Imports MASS, Matrix, igraph, methods, partykit, psychotools, psychotree, rARPACK, qvcalc, sandwich, stats

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

RoxygenNote 6.0.1

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 2018-04-09 12:58:12 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 the data must be prepared as grouped_rankings.

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 http://www.preflib.org.

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))
```

---

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)

# Convert these to numeric values, allowing us to impute the
# middle-ranked variety (a strict ranking is assumed here, so the
# sum of each row should be 6)
beans <- within(beans, {
   best <- match(best, c("A", "B", "C"))
   worst <- match(worst, c("A", "B", "C"))
   middle <- 6 - best - worst
})
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 <- as.matrix(beans[c("variety_a", "variety_b", "variety_c")])
head(varieties, 2)

# So we can convert the variety IDs to the variety names
n <- nrow(beans)
beans <- within(beans, {
    best <- varieties[cbind(seq_len(n), best)]
    worst <- varieties[cbind(seq_len(n), worst)]
    middle <- varieties[cbind(seq_len(n), middle)]
})
head(beans[c("best", "middle", "worst")], 2)

# Create a rankings object from the rankings of order three
```

```
## each ranking is a sub-ranking of three varieties from the full set
lab <- c("Local", sort(unique(as.vector(varieties))))
R <- as.rankings(beans[c("best", "middle", "worst")],
                 input = "ordering", labels = lab)
head(R)

# Convert worse/better columns to ordered pairs
head(beans[c("var_a", "var_b", "var_c")], 2)
paired <- list()
for (id in c("a", "b", "c")){
    ordering <- matrix("Local", nrow = n, ncol = 2)
    worse <- beans[[paste0("var_", id)]] == "Worse"
    ## put trial variety in column 1 when it is not worse than local
    ordering[!worse, 1] <- beans[[paste0("variety_", id)]][!worse]
    ## put trial variety in column 2 when it is worse than local
    ordering[worse, 2] <- beans[[paste0("variety_", id)]][worse]
    paired[[id]] <- ordering
}
head(paired[["a"]])

# Convert orderings to sub-rankings of full set and combine all rankings
paired <- lapply(paired, as.rankings, input = "ordering", labels = lab)
R <- rbind(R, paired[["a"]], paired[["b"]], paired[["c"]])
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 list 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)
coded_choices <- choices(R, names = FALSE)
attr(coded_choices, "objects")

## Coercion to tibble is straightforwards
if (require(tibble)){
    as.tibble(coded_choices)
}
```

---

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$membership
## keep items in cluster 1
X[,res$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
```

`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 passed to method (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

grouped_rankings *Grouped Rankings Object*

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

```
grouped_rankings(rankings, index, ...)

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

as.grouped_rankings(x, ...)

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

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

Arguments

rankings	a rankings object or an object that can be coerced by as.rankings.
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.
x	an object that can be coerced to a "grouped_rankings" object for as.group_rankings, or a "grouped_rankings" object for [and 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

# grouped rankings (first three in group 1, next two in group 2)
G <- grouped_rankings(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

A method for `itempar` to extract the item parameters (abilities or log-abilities) from a Plackett-Luce model.

Usage

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

Arguments

`object`	a fitted model object as returned by `PlackettLuce`.
`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 87 columns corresponding to the positions. Columns 1 to 43 contain the ID for the driver that came first to last place respectively. Columns 44 to 87 are zero, since only 43 out of 87 drivers participated in each race. 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, 1:45]
R <- as.rankings(nascar, input = "ordering")
colnames(R) <- attr(nascar, "drivers")
R[1:3, 1:3, as.rankings = FALSE]
R[1:3]

# 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 (not all objects ranked) and include ties of any order.

Usage

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

Arguments

rankings	a "rankings" object, or an object that can be coerced by as.rankings.
npseudo	when using pseudodata: the number of wins and losses to add between each object and a hypothetical reference object.
weights	an optional vector of weights for each ranking.
start	starting values for the worth parameters and the tie parameters: either the result of a call to coef.PlackettLuce, or a vector of parameters on the raw scale, as in the coefficients element of a "PlackettLuce" object.
method	the method to be used for fitting: "iterative scaling" (default: 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).
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	the maximum number of 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.
`df.residual`	The residual degrees of freedom.
`rank`	The rank of the model.
`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.
`maxTied`	The maximum number of objects observed in a tie.
`conv`	The convergence code: 0 for successful convergence; 1 if reached `maxit` 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 representing the prevalence of ties of order n, 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 C_j, A_j is 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 interpretable via the probability that two given 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 interpretable similarly, 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. Thus even for networks that are already connected, adding pseudo-rankings reduces both the bias and variance of the estimates.

Controlling the fit

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`.

See Also

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

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

Fit Plackett-Luce tree: `grouped_rankings`, `pltree`.

Example data sets: `beans`, `nascar`, `pudding`, `read.soc`.

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")

mod <- PlackettLuce(R)
coef(mod)
```

`pltree` *Plackett-Luce Trees*

Description

Recursive partitioning based on Plackett-Luce models.

Usage

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

## S3 method for class 'pltree'
predict(object, newdata = NULL, type = c("itempar", "rank",
  "best", "node"), ...)

## S3 method for class 'pltree'
AIC(object, newdata = 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, 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` by `pltree`; to `itempar` by `predict`, and to `model.frame` by `AIC`.
`object`	a fitted model object of class `"pltree"`.
`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

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.

Various methods are provided for "pltree" objects, most of them inherited from "modelparty" objects (e.g. print, summary), or "bttree" objects (plot). itempar extracts the abilities or item parameters from the Plackett-Luce models in each node of the tree using itempar.PlackettLuce. The plot method employs the node_btplot panel-generating function. AIC computes $-2L + 2df$ where L is the joint likelihood of the observed rankings under the tree model and df is the degrees of freedom used to fit the tree model.

Value

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

See Also

bttree

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 = 6)
    ## Topmodel2007[, -1] gives covariate values for each judge
    head(Topmodel2007[, -1], 2)

    ## 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)
    ## abilities with Anja as reference
    itempar(tm_tree, ref = "Anja")

    ## results for the first three judges
    newdata <- Topmodel2007[1:3,]
    ### fitted probabilities
    predict(tm_tree, newdata)
    ### fitted log-abilities, with Anni as reference
    predict(tm_tree, newdata, log = TRUE, ref = "Anni")
    ###  item ranks
    predict(tm_tree, newdata, type = "rank")
    ### top ranked item
    predict(tm_tree, newdata, type = "best")
    ### node the judge belongs to
    predict(tm_tree, newdata, type = "node")

}
```

`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

```
# reshape to give one row per unique ranking
nr <- 3*nrow(pudding)
R <- matrix(0, nrow = nr, ncol = 6,
            dimnames = list(NULL, seq_len(6)))
i <- rep(pudding$i, 3)
j <- rep(pudding$j, 3)
R[cbind(seq_len(nr), i)] <- rep(c(1, 2, 1), each = nrow(pudding))
R[cbind(seq_len(nr), j)] <- rep(c(2, 1, 1), each = nrow(pudding))
# weights are 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 = 1, ...)
```

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 http://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, verbose = TRUE, ...)

as.rankings(x, input = c("ranking", "ordering"), labels = NULL,
  verbose = TRUE, ...)

## Default S3 method:
as.rankings(x, input = c("ranking", "ordering"),
  labels = NULL, verbose = TRUE, ...)

## S3 method for class 'matrix'
as.rankings(x, input = c("ranking", "ordering"),
  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 = 40, ...)

## S3 method for class 'rankings'
rbind(..., labels = NULL)
```

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.
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 each row in the input matrix contains a numeric "ranking" (dense, standard/modified competition or fractional ranking) or an "ordering", i.e. the items ordered by rank.
labels	for input = "ordering" an optional vector of labels for the items. If NULL, the items will be labelled by 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. 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.

Rankings with less than 2 items are dropped.

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

Value

a "rankings" object, which is a matrix of dense rankings with one attribute omit the indices of any rankings that were omitted due to insufficient data (less than two non-missing ranks).

Examples

```
# create rankings from data in long form
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
unclass(R)[2:3,]
# extract rankings for item 1 as a vector
R[,1, as.rankings = FALSE]
```

`read.soc` *Read Strict Orders - Complete List Data from File*

Description

Read data from a "Strict Orders - Complete List" file, as provided by {PrefLib}: A Library for Preferences.

Usage

```
read.soc(file)
```

Arguments

`file`	a path or url for the Strict Orders - Complete List file, conventionally with a `.soc` extension.

Value

a data frame with first column `n`, 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. The data frame has an attribute `"item"` giving the labels corresponding to each item number.

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.

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

## End(Not run)
```

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

Description

Simulate from `PlackettLuce` fitted objects

Usage

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

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

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

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`	the type of coefficients to return: one of `"ties"`, `"worth"` or `"all"`.
`...`	additional arguments, currently ignored.

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.

Index