Package 'prefio'

September 7, 2023

Title Structures for Preference Data

Description Convenient structures for creating, sourcing, reading, writing and manipulating ordinal preference data. Methods for writing to/from PrefLib formats. See Nicholas Mattei and Toby Walsh ``PrefLib: A Library of Preference Data" (2013) <doi:10.1007/978-3-642-41575-3_20>.

Version 0.1.1

Depends R (>= 2.10)

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.3

URL https://github.com/fleverest/prefio/, https://fleverest.github.io/prefio/

BugReports https://github.com/fleverest/prefio/issues/

Imports dplyr, magrittr, tidyr, stats

Suggests covr, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Floyd Everest [aut, cre] (<https://orcid.org/0000-0002-2726-6736>), Heather Turner [aut] (<https://orcid.org/0000-0002-1256-3375>), Damjan Vukcevic [aut] (<https://orcid.org/0000-0001-7780-9586>)

Maintainer Floyd Everest <me@floydeverest.com>

Repository CRAN

Date/Publication 2023-09-07 12:30:02 UTC

R topics documented:

`adjacency` *Create an Adjacency Matrix for a set of Preferences*

Description

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

Usage

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

Arguments

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

Details

For a `preferences` object with 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 preferences {1} > {3, 4} > {2}, item 1 wins over items 2, 3, and 4, while 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.

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.preferences(X, format = "ranking", item_names = LETTERS[1:5])
adjacency(X)

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

`aggregate.preferences` *Aggregate Preferences*

Description

Aggregate `preferences`, returning an `aggregated_preferences` object of the unique preferences and their frequencies. The frequencies can be accessed via the function `frequencies()`.

Usage

```
## S3 method for class 'preferences'
aggregate(x, frequencies = NULL, ...)

as.aggregated_preferences(x, ...)

## S3 method for class 'aggregated_preferences'
x[i, j, ...]

frequencies(x)
```

Arguments

`x`	A `preferences` object for `aggregate()`; an object that can be coerced to an `aggregated_preferences` object for `as.aggregated_preferences()`, otherwise an `aggregated_preferences` object.
`frequencies`	A vector of frequencies for preferences that have been previously aggregated.
`...`	Additional arguments, currently unused.
`i`	indices specifying preferences to extract.
`j`	indices specifying items to extract.
`as.aggregated_preferences`	if `TRUE` create an `aggregated_preferences` object from the indexed preferences Otherwise index the underlying matrix of ranks and return in a data frame with the corresponding frequencies.

Value

A data frame of class `aggregated_preferences`, with columns:

preferences A `preferences` object of the unique preferences

frequencies The corresponding frequencies.

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

Examples

```
# create a preferences object with duplicated preferences
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.preferences(R, format = "ranking")

# aggregate the preferences
A <- aggregate(R)

# Or pass `aggregate = TRUE` to `as.preferences`
A <- as.preferences(R, aggregate = TRUE)

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

# (partial) preferences projected to items 2-4 only
A[, 2:4]

# Project preferences onto their hightest ranking
A[, 1, by.rank = TRUE]

# convert to a matrix
as.matrix(A)
```

`choices` *Choices Object*

Description

Convert a set of preferences to a list of choices, alternatives, and preferences.

Usage

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

Arguments

`preferences`	a `preferences` object, or an object that can be coerced by `as.preferences`.
`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 items chosen for a single rank in the ordering.

alternatives A list where each element represents the alternatives (i.e. the set of remaining items to choose from) for a single rank.

ordering A list where each element represents the ordering 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")
R <- preferences(R, format = "ranking")

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")
```

`group` *Group Preferences*

Description

Create an object of class `grouped_preferences` which associates a group index with an object of class `preferences`. This allows the preferences to be linked to covariates with group-specific values.

Usage

```
group(x, ...)

## S3 method for class 'preferences'
group(x, index, ...)
```

```
## S3 method for class 'grouped_preferences'
x[i, j, ...]

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

Arguments

x	A preferences object for group(); otherwise a grouped_preferences object.
...	Additional arguments passed on to as.preferences by grouped_preferences; unused by format.
index	A numeric vector or a factor with length equal to the number of preferences specifying the subject for each set.
i	Indices specifying groups to extract, may be any data type accepted by [.
j	Indices specifying items to extract. object, otherwise return a matrix/vector.
max	The maximum number of preferences to format per subject.
width	The maximum width in number of characters to format the preferences.

Value

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

preferences	The preferences object.
index	An index matching each preference set to each group ID.

Examples

```
# ungrouped preferences (5 preference sets, 4 items)
R <- as.preferences(
  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),
  format = "ranking",
  item_names = LETTERS[1:4]
)
length(R)

# group preferences (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 preference sets are shown per group, "..." indicates if
```

```
## there are further preferences
G
print(G, max = 1)

## select preferences from group 1
G[1, ]

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

## Project preferences in all groups to their first preference
G[, 1, by.rank = TRUE]

## preferences from group 2, excluding item 3
## - note group 2 becomes the first (and only) group
G[2, -3]

# Group preferences by a factor
G <- group(R, factor(c("G1", "G1", "G1", "G2", "G2")))

G
print(G, max = 1)

## select preferences from group G1
G["G1"]
```

`preferences` *Preferences Object*

Description

Create a `preferences` object for representing Ordinal Preference datasets.

Usage

```
preferences(
  data,
  format = c("long", "ordering", "ranking"),
  id = NULL,
  rank = NULL,
  item = NULL,
  item_names = NULL,
  frequencies = NULL,
  aggregate = FALSE,
  verbose = TRUE,
  ...
)
```

```
## S3 method for class 'preferences'
x[i, j, ..., by.rank = FALSE, as.ordering = FALSE]

as.preferences(x, ...)

## S3 method for class 'grouped_preferences'
as.preferences(x, aggregate = FALSE, verbose = TRUE, ...)

## Default S3 method:
as.preferences(
  x,
  format = c("long", "ranking", "ordering"),
  id = NULL,
  item = NULL,
  rank = NULL,
  item_names = NULL,
  aggregate = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'matrix'
as.preferences(
  x,
  format = c("long", "ranking"),
  id = NULL,
  item = NULL,
  rank = NULL,
  item_names = NULL,
  aggregate = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'aggregated_preferences'
as.preferences(x, ...)

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

Arguments

data	A data frame or matrix in one of three formats: **"ordering"** Orderings must be a data frame with list-valued columns. Each row represents an ordering of the items from first to last, representing ties by a list of vectors corresponding to the items. **"ranking"** Each row assigns a rank to each item, with columns representing items. Note that rankings will be converted to 'dense' rankings in the output

(see Details).

"long" Three columns: an `id` column grouping the rows which correspond to a single set of preferences, an `item` column specifying (either by index or by name) the item each row refers to, and a `rank` column specifying the rank for the associated item.

`format` The format of the data: one of "ordering", "ranking", or "long" (see above). By default, `data` is assumed to be in "long" format.

`id` For `data` in long-format: the column representing the preference set grouping.

`rank` For `data` in long-format: the column representing the rank for the associated item.

`item` For `data` in long-format: the column representing the items by name or by index, in which case the `item_names` parameter should also be passed, or the items will be named as integers.

`item_names` The names of the full set of items. When loading data using integer-valued indices in place of item names, the `item_names` character vector should be in the correct order.

`frequencies` An optional integer vector containing the number of occurences of each preference. If provided, the method will return a `aggregated_preferences` object with the corresponding frequencies.

`aggregate` If `TRUE`, aggregate the preferences via `aggregate.preferences` before returning. This returns an `aggregated_preferences` object.

`verbose` If `TRUE`, diagnostic messages will be sent to stdout.

`...` Unused.

`x` The `preferences` object to subset.

`i` The index of the preference-set to access.

`j` The item names or indices to project onto, e.g. if `j = 1` the preferences will be projected only onto the first item; if `by.rank = TRUE` `j` corresponds to the rank of the items to subset to, e.g. if `j = 1` then preferences will be truncated to only contain their highest-preference.

`by.rank` When `FALSE`, the index `j` corresponds to items, when true the index corresponds to rank.

`as.ordering` When `FALSE`, returns a `preferences` object: internally rows i contain the ranking assigned to each item in preference p_i. When `TRUE`, returns a data frame where columns group the items by rank.

`width` The width in number of characters to format each preference, truncating by "..." when they are too long.

Details

Ordinal preferences can order every item, or they can order a subset. Some ordinal preference datasets will contain ties between items at a given rank. Hence, there are four distinct types of preferential data:

`soc` Strict Orders - Complete List

`soi` Strict Orders - Incomplete List

`toc` Orders with Ties - Complete List

`toi` Orders with Ties - Incomplete List

The data type is stored alongside the `preferences` as an attribute `attr(preferences, "preftype")`. The data type is determined automatically. If every preference ranks every item, then the data type will be "soc" or "soi". Similarly, if no preference contains a tie the data type will be "toc" or "toi".

A set of preferences can be represented either by `ranking` or by `ordering`. These correspond to the two ways you can list a set of preferences in a vector:

`ordering` The items are listed in order of most preferred to least preferred, allowing for multiple items being in the same place in the case of ties.

`ranking` A rank is assigned to each item. Conventionally, ranks are integers in increasing order (with larger values indicating lower preference), but they can be any ordinal values. Any given rankings will be converted to 'dense' rankings: positive integers from 1 to some maximum rank, with no gaps between ranks.

When reading preferences from an `ordering` matrix, the index on the items is the order passed to the `item_names` parameter. When reading from a `rankings` matrix, if no `item_names` are provided, the order is inferred from the named columns.

A `preferences` object can also be read from a long-format matrix, where there are three columns: `id`, `item` and `rank`. The `id` variable groups the rows of the matrix which correspond to a single set of preferences, which the `item:rank`, pairs indicate how each item is ranked. When reading a matrix from this format and no `item_names` parameter is passed, the order is determined automatically.

Value

By default, a `preferences` object, which is a data frame with list-valued columns corresponding to preferences on the items. This may be an ordering on subsets of the items in the case of ties, or a potentially-partial strict ordering. In the case of partial or tied preferences, some entries may be empty lists.

Examples

```
# create rankings from data in long form

# Example long-form data
x <- data.frame(
  id = c(rep(1:4, each = 4), 5, 5, 5),
  item = 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)
)

# * Set #1 has two different ranks for the same item (item C
# has rank 1 and 2). This item will be excluded from the preferences.
# * All ranks are missing in set #2, a technically valid partial ordering
# * Some ranks are missing in set #3, a perfectly valid partial ordering
```

```
# * Set #4 has inconsistent ranks for two items, and a rank with a
# missing item.
# * Set #5 is not a dense ranking. It will be converted to be dense and then
# inferred to be a regular partial ordering with ties.
split(x, x$rank)

# Creating a preferences object with this data will attempt to resolve these
# issues automatically, sending warnings when assumptions need to be made.
preferences(x, id = "id", item = "item", rank = "rank")

# Convert an existing matrix of rankings to a preferences object.
rnk <- 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(rnk) <- c("apple", "banana", "orange", "pear")

rnk <- as.preferences(rnk, format = "ranking")

# Convert an existing data frame of orderings to a preferences object.
e <- character() # short-hand for empty ranks
ord <- preferences(
  as.data.frame(
    rbind(
      list(1, 2, e, e), # apple, banana
      list("banana", "orange", "pear", "apple"),
      list(c("banana", "orange", "pear"), "apple", e, e),
      list("apple", "banana", "orange", e),
      list(c("banana", "orange"), "apple", e, e),
      list("apple", "pear", "orange", e)
    )
  ),
  format = "ordering",
  item_names = c("apple", "banana", "orange", "pear")
)

# Access the first three sets of preferences
ord[1:3, ]

# Truncate preferences to the top 2 ranks
ord[, 1:2, by_rank = TRUE]

# Exclude pear from the rankings
ord[, -4]

# Get the highest-ranked items and return as a data.frame of orderings
ord[, 1, by_rank = TRUE, as.ordering = TRUE]

# Convert the preferences to a ranking matrix
```

```
as.matrix(ord)

# Get the rank of apple in the third preference-set
as.matrix(ord)[3, 1]

# Get all the ranks assigned to apple as a vector
as.matrix(ord)[, "apple"]
```

`read_preflib` *Read Ordinal Preference Data From PrefLib*

Description

Read orderings from `.soc`, `.soi`, `.toc` or `.toi` files storing ordinal preference data format as defined by {PrefLib}: A Library for Preferences into a `preferences` object.

Usage

```
read_preflib(
  file,
  from_preflib = FALSE,
  preflib_url = "https://www.preflib.org/static/data"
)
```

Arguments

`file`	A preferential data file, conventionally with extension `.soc`, `.soi`, `.toc` or `.toi` according to data type.
`from_preflib`	A logical which, when `TRUE` will attempt to source the file from PrefLib by adding the database `HTTP` prefix.
`preflib_url`	The URL which will be prepended to `file`, if `from_preflib` is `TRUE`.

Details

Note that PrefLib refers to the items being ordered by "alternatives".

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

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

A PrefLib file may be corrupt, in the sense that the ordered alternatives do not match their names. In this case, the file can be read in as a data frame (with a warning), but `as.aggregated_preferences` will throw an error.

Value

An aggregated_preferences object containing the PrefLib data.

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.

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


# strict complete orderings of four films on Netflix
netflix <- read_preflib("netflix/00004-00000138.soc", from_preflib = TRUE)
head(netflix)
names(netflix$preferences)

# strict incomplete orderings of 6 random cities from 36 in total
cities <- read_preflib("cities/00034-00000001.soi", from_preflib = TRUE)
```

---

write_preflib *Write Ordinal Preference Data to PrefLib Formats*

Description

Write preferences to .soc, .soi, .toc or .toi file types, as defined by the PrefLib specification: {PrefLib}: A Library for Preferences.

Usage

```
write_preflib(
  x,
  file = "",
  title = NULL,
  publication_date = NULL,
  modification_type = NULL,
  modification_date = NULL,
  description = NULL,
```

```
  relates_to = NULL,
  related_files = NULL
)
```

Arguments

x	An aggregated_preferences object to write to file. If x is of a different class, it attempts to coerce x into an aggregated_preferences object via as.aggregated_preferences().
file	Either a character string naming the a file or a writeable, open connection. The empty string "" will write to stdout.
title	The title of the data file, for instance the name of the election represented in the data file. If not provided, we check for the presence of attr(x, "preflib"), and if it exists we check for TITLE.
publication_date	The date at which the data file was published for the first time. If not provided, we check for the presence of attr(x, "preflib"), and if it exists we check for PUBLICATION DATE.
modification_type	The modification type of the data: one of original, induced, imbued or synthetic (see Details). If not provided, we check for the presence of attr(x, "preflib"), and if it exists we check for MODIFICATION TYPE.
modification_date	The last time the data was modified. If not provided, we check for the presence of attr(x, "preflib"), and if it exists we check for MODIFICATION DATE.
description	A description of the data file, providing additional information about it. If not provided, we check for the presence of attr(x, "preflib"), and if it exists we check for DESCRIPTION.
relates_to	The name of the data file that the current file relates to, typically the source file in case the current file has been derived from another one. If not provided, we check for the presence of attr(x, "preflib"), and if it exists we check for RELATES TO.
related_files	The list of all the data files related to this one, comma separated. If not provided, we check for the presence of attr(x, "preflib"), and if it exists we check for RELATED FILES.

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

The PrefLib format specification requires some additional metadata. Note that the additional metadata required for the PrefLib specification is not necessarily required for the write_preflib method; any missing fields required by the PrefLib format will simply show "NA".

TITLE (required) The title of the data file, for instance the year of the election represented in the data file.

DESCRIPTION (optional) A description of the data file, providing additional information about it.

RELATES TO (optional) The name of the data file that the current file relates to, typically the source file in case the current file has been derived from another one.

RELATED FILES (optional) The list of all the data files related to this one, comma separated.

PUBLICATION DATE (required) The date at which the data file was published for the first time.

MODIFICATION TYPE (required) The modification type of the data. One of:

original Data that has only been converted into a PrefLib format.

induced Data that has been induced from another context. For example, computing a pairwise relation from a set of strict total orders. No assumptions have been made to create these files, just a change in the expression language.

imbued Data that has been imbued with extra information. For example, extending an incomplete partial order by placing all unranked candidates tied at the end.

synthetic Data that has been generated artificially.

MODIFICATION DATE (optional) The last time the data was modified.

In addition to these fields, some required PrefLib fields will be generated automatically depending on arguments to `write_preflib()` and the attributes of the `aggregated_preferences` object being written to file:

FILE NAME The name of the output file.

DATA TYPE The data type (one of `soc`, `soi`, `toc` or `toi`).

NUMBER ALTERNATIVES The number of items.

ALTERNATIVE NAME `X` The name of each item, where `X` ranges from `0` to `length(items)`.

NUMBER VOTERS The total number of orderings.

NUMBER UNIQUE ORDERS The number of distinct orderings.

Note that PrefLib refers to the items as "alternatives". The "alternatives" in the output file will be the same as the "items" in the `aggregated_preferences` object.

Value

No return value. Output will be written to file or stdout.

Index