Package ‘TraMineRextras’

October 12, 2022

Version 0.6.5

Date 2022-10-09

Title TraMineR Extension

Depends R (>= 3.5.0), TraMineR (>= 2.2-1)

Imports grDevices, graphics, stats, cluster, RColorBrewer, colorspace,
survival, doParallel, parallel, foreach

Description
Collection of ancillary functions and utilities to be used in conjunction with the 'TraMineR' package for sequence data exploration. Includes, among others, specific functions such as state survival plots, position-wise group-typical states, dynamic sequence indicators, and dissimilarities between event sequences. Also includes contributions by non-members of the TraMineR team such as the relative frequency plot and methods for polyadic data.

License GPL (>= 2)

URL `http://traminer.unige.ch/`

Encoding UTF-8

Maintainer Gilbert Ritschard <`gilbert.ritschard@unige.ch`>

NeedsCompilation yes

Author Gilbert Ritschard [aut, cre, ths, cph]
(<`https://orcid.org/0000-0001-7776-0903`>),
Matthias Studer [aut] (<`https://orcid.org/0000-0002-6269-1412`>),
Reto Buergin [aut] (<`https://orcid.org/0000-0002-6212-1567`>),
Tim F. Liao [ctb] (<`https://orcid.org/0000-0002-1296-7660`>),
Alexis Gabadinho [ctb],
Pierre-Alexandre Fonta [ctb],
Nicolas S. Muller [ctb],
Patrick Rousset [ctb]

Repository CRAN

Date/Publication 2022-10-09 16:20:02 UTC

R topics documented:

`TraMineRextras-package` *TraMineR Extension*

Description

(Version: 0.6.5) Collection of ancillary functions and utilities to be used in conjunction with the 'TraMineR' package for sequence data exploration. Includes, among others, specific functions such as state survival plots, position-wise group-typical states, dynamic sequence indicators, and dissimilarities between event sequences. Also includes contributions by non-members of the TraMineR team such as the relative frequency plot and methods for polyadic data.

Author(s)

Gilbert Ritschard, Matthias Studer, Reto Buergin

`convert` *Converting between graphical formats*

Description

Wrapper function for converting graphics with ImageMagick

Usage

```
convert.g(path = NULL, fileroot= "*", from = "pdf",
    to = "png", create.path = TRUE, options = NULL)
```

Arguments

`path`	String: The path to the from graphic files.
`fileroot`	String: Graphic root name; default is "*" for all files with the `from` extension.
`from`	File type extension specifying the from format.
`to`	File type extension specifying the to format.
`create.path`	Logical: Should the output files be placed in a `to` subfolder.
`options`	Additional options to be passed to the ImageMagick `mogrify` function

Details

Conversion is done through a call to ImageMagick `mogrify` function. This means that ImageMagick should be installed on your system. It must also be listed in the path.

for some values such as `"pdf"` and `"eps"` of the `from` or `to` arguments ImageMagick works in conjunction with Gostscript. The latter should, therefore, also be accessible.

See Also

png, pdf

Examples

```
## Not run:
## Convert all .pdf graphics in the "figSW" directory
## into .png files and put the files in a "png" subfolder.
convert.g(path="figSW", from="pdf", to="png")

## Same, but convert to .jpg files.
convert.g(path="figSW", to="jpg")

## convert file "example.eps" in current path to ".pdf"
## and put it in same folder.
convert.g(fileroot = "example", create.folder=FALSE)

## End(Not run)
```

`createdatadiscrete` *Transform time to event data into a discrete data format*

Description

Transform time to event data (in a specific format, see the details below) into a person-period data format suitable for automatic sequential association rules extraction

Usage

```
createdatadiscrete(ids, data, vars, agemin, agemax,
    supvar=NULL)
```

Arguments

`ids`	a vector containing an unique identification number for each case
`data`	a data frame containing time to event data, with variables containing the durations named as in the vars argument, and those with the censoring indicators named as in the vars argument followed by "ST" (for example column A is duration until event A, and column AST is the censoring indicator). This data frame must contain an unique identification variable named "IDPERS".
`vars`	a vector with the names of the duration variables
`agemin`	a data frame with two variables : "IDPERS" for the unique identification variable, and "AGE" for the starting time of the observation
`agemax`	a data frame with two variables : "IDPERS" for the unique identification variable, and "AGE" for the ending time of the observation
`supvar`	a vector of variables to add to the resulting person-period data frame

Details

The data frame from the `data` argument must contain two variables for each event: a duration variable that indicates the time when the event occurred, and a status variable that indicates if the event occurred (1) or not (0). If the event did not occur, the observation for this individual will go until the age specified through the `agemax` argument. Each status variable must have the name of the corresponding duration variable suffixed by "ST". For example, if the duration variable for an event "divorce" is called "div", then the status variable has to be named "divST".

The result from this function is a list with one person-period data frame by event, where the dependent event is different each time. Please see the attached data file and code for an example.

The resulting object is one of the required argument for the `seqerulesdisc` function that computes the association rules, the hazard ratios and the p-values, using discrete-time regressions. Unlike the method presented in Müller et al. 2010, this function does not use Cox proportional hazard models, but discrete-time regression models with a complementary log-log link function, which gives similar results.

Value

a list with one person-period data frame by event, where the dependent event is different each time. Please see the attached data file and code for an example.

Author(s)

Nicolas S. Müller

References

Müller, N.S., M. Studer, G. Ritschard et A. Gabadinho (2010), Extraction de règles d'association séquentielle à l'aide de modèles semi-paramétriques à risques proportionnels, *Revue des Nouvelles Technologies de l'Information*, **Vol. E-19**, EGC 2010, pp. 25-36

See Also

`seqerulesdisc` to compute the association rules.

Examples

```
##
```

`dissvar.grp` *Discrepancy by group.*

Description

This function computes the dissimilarity-based discrepancy measure of the groups defined by the group variable. The function is a wrapper for the TraMineR `dissvar` function.

Usage

```
dissvar.grp(diss, group=NULL, ...)
```

Arguments

`diss`	a dissimilarity matrix or a `dist` object.
`group`	group variable. If `NULL` a single group is assumed.
`...`	additional arguments passed to `dissvar` .

Details

The function is a wrapper for running `dissvar` on the different groups defined by the `group` variable.

Value

A vector with the group discrepancy measures.

Note

This function is a pre-release and further testing is still needed, please report any problems.

Author(s)

Gilbert Ritschard

See Also

`dissvar`

Examples

```
## create the biofam.seq state sequence object from the biofam data.
data(biofam)
biofam <- biofam[1:100,]
biofam.seq <- seqdef(biofam[,10:25])
dist <- seqdist(biofam.seq, method="HAM")

## discrepancy based on non-squared dissimilarities
dissvar.grp(dist, biofam$plingu02)
## square root of discrepancy based on squared dissimilarities
sqrt(dissvar.grp(dist, biofam$plingu02, squared=TRUE))
```

`FCE_to_TSE` *Data conversion from Fixed Column Event format to TSE.*

Description

Data conversion from Fixed Column Event format to TSE.

Usage

```
FCE_to_TSE(seqdata, id = NULL, cols, eventlist = NULL, firstEvent = NULL)
```

Arguments

`seqdata`	data frame or matrix containing event sequence data in FCE format.
`id`	column containing the identification numbers for the sequences.
`cols`	Real. Column containing the timing of the event. A missing value is interpreted as a non-occurrence of the event.
`eventlist`	Event names, specified in the same order as `cols` argument. If `NULL` (default), column names are used.
`firstEvent`	Character. The name of an event to be added at the beginning of each event sequences. This allows to include individuals with no events. If `NULL` (default), no event is added.

Details

The usual data format for event sequence is TSE (see `seqecreate`).

Value

A `data.frame` with three columns: "id", "timestamp" and "event".

Note

This function is a pre-release and further testing is still needed, please report any problems.

Author(s)

Matthias Studer

See Also

`seqecreate`, `seqformat`

Examples

```
## Generate a ramdom data set
fce <- data.frame(id=1:100, event1=runif(100), event2=runif(100))

## Add missing values (ie non-occurrences)
fce[runif(100)<0.1, "event1"] <- NA
fce[runif(100)<0.1, "event2"] <- NA

tse <- FCE_to_TSE(fce, id="id", cols=c("event1", "event2"),
       eventlist=c("Marriage", "Child birth"), firstEvent="Birth")

seq <- seqecreate(tse)
print(seq[1:10])
```

`group.p` *Adds proportion of occurrences to each level names*

Description

Adds the proportion of occurrences of each level to the corresponding level name.

Usage

```
group.p(group, weights=NULL)
```

Arguments

`group`	A group variable.
`weights`	Vector of weights of same length as the group variable.

Details

The `group` variable can be a factor or a numerical variable. In the latter case it is transformed to a factor.

Author(s)

Gilbert Ritschard

See Also

`seqplot`.

Examples

```
data(actcal)
actcal <- actcal[1:100,]
actcal.seq <- seqdef(actcal[,13:24])
seqdplot(actcal.seq, group=group.p(actcal$sex))

levels(group.p(actcal$sex, weights=runif(length(actcal$sex))))
```

`HSPELL_to_STS` *Data conversion from Horizontal Spell to STS.*

Description

Convert data from Horizontal Spell to STS.

Usage

```
HSPELL_to_STS(seqdata, begin, end, status = NULL,
    fixed.status = NULL, pvar = NULL, overwrite = TRUE,
    fillblanks = NULL, tmin = NULL, tmax = NULL, id = NULL,
    endObs = NULL)
```

Arguments

`seqdata`	a data frame or matrix containing sequence data.
`begin`	Vector containing the columns (name or number) with the beginning position of each spell.
`end`	Vector containing the columns (name or number) with the end position of each spell.
`status`	Vector containing the columns (name or number) with the status of each spell.
`fixed.status`	Default status (for period not covered by any spell.)
`pvar`	names or numbers of the column containing the 'birth' time.
`overwrite`	Should the most recent episode overwrite the older one when they overlap? If `FALSE`, the most recent episode starts from the end of the previous one.
`fillblanks`	If not `NULL`, character used for filling gaps between episodes.
`tmin`	If sequences are to be defined on a calendar time axis, it defines the starting time of the axis. If set as NULL, the start time is set as the minimum of the 'begin' column in the data.
`tmax`	If year sequences are wanted, defines the ending year of the sequences. If set to NULL, it is guessed from the data (not so accurately!).
`id`	column containing the identification numbers for the sequences.
`endObs`	An optional end of observation date. Usefull for retrospective survey.

Details

Hortizontal spell data format has the following caracteristics: - One row per individual - Each spell is specified with three consecutive variables: a begin date, an end date, and the status. - For unused spells, begin and end values should be set as `NA`.

Value

A `data.frame` with the sequence in STS format.

Note

This function is a pre-release and further testing is still needed, please report any problems.

Author(s)

Matthias Studer

See Also

See Also `seqformat`.

Examples

```
hspell <- data.frame(begin1=rep(1, 5), end1=c(2:5, NA), status1=1:5,
                     begin2=c(3:6, NA), end2=rep(NA, 5), status2=5:1)
sts <- HSPELL_to_STS(hspell, begin=c("begin1", "begin2"), end=c("end1", "end2"),
                     status=c("status1", "status2"))
```

`pamward` *PAM from k-solution of hierarchical clustering*

Description

Runs a pam clustering (`pam`) from the solution in k groups of a hierarchical clustering (`agnes`) .

Usage

```
pamward(diss, k=3, method="ward", dist)
```

Arguments

`diss`	Distance matrix or object.
`k`	Integer. Number of clusters.
`method`	Method for the hierarchical clustering (see `agnes`).
`dist`	Deprecated. Use `diss` instead.

Details

The function first runs the hierarchical clustering, retrieves the medoids of the solution for the provided `k` and uses those medoids as start centers for the `pam` partitioning.

Value

An object of class `"pam"`. See `pam.object` for details.

Author(s)

Gilbert Ritschard

See Also

`agnes` and `pam`.

Examples

```
library(cluster)
data(actcal)
actcal.seq <- seqdef(actcal[1:200,13:24])
actcal.ham <- seqdist(actcal.seq, method = "HAM")
clust <- pamward(actcal.ham, k = 4)
table(clust$clustering)
```

`plot.dynin` *Dynamic index plot*

Description

Plot of dynamic (i.e. successive) cross-sectional summaries of an individual index. The successive values of the individual index for all sequences should be collected in a `dynin` table as produced by `seqindic.dyn`.

Usage

```
## S3 method for class 'dynin'
plot(x, fstat=weighted.mean, group=NULL, conf=FALSE,
     main=NULL, col=NULL, lty=NULL, lwd=3.5, ylim=NULL,
     ylab=NULL, xlab=NULL, xtlab=NULL, xtstep=NULL, tick.last=NULL,
     with.legend=TRUE, glabels=NULL, legend.pos="topright",
     horiz=FALSE, cex.legend=1, bcol=NULL, na.rm=FALSE, ret=FALSE, ...)
```

Arguments

`x`	object of class `dynin` as produced by `seqindic.dyn`
`fstat`	function: summary function to compute the values plotted. Default is `weighted.mean` with weights taken from the `weights` attribute of `x`. When `weighted.mean` and `x` has no weights, `mean` is used instead.
`group`	factor or discrete vector: group membership; a curve is drawn for each group. If `NULL` (default) a single curve for the whole set is drawn.
`conf`	logical or numeric: If logical, should confidence bands be displayed? If numeric, confidence probability. `TRUE` is equivalent to .95. Applies only when `fstat=mean` or `fstat=weighted.mean`.
`main`	character string: Plot title.
`col`	color vector. Group line colors. If `NULL` (default), colors are automatically assigned using `qualitative_hcl` with the `'Dark 3'` palette (see `hcl_palettes`).
`lty`	string vector. Group line types (see `lines`). If `NULL` (default), types are automatically assigned.
`lwd`	integer vector: Group line widths (see `lines`). If `NULL` (default), set as 3.5.
`ylim`	pair of numerics defining the range for the y-axis. If left `NULL`, the limits are defined from the data.
`ylab`	character string: y axis label.
`xlab`	character string: x axis label.
`xtlab`	vector of strings defining the x-axis tick labels. If `NULL`, column names of the `x` table are used.
`xtstep`	integer: step between tick marks on the x-axis. If unspecified, attribute `xtstep` of the `x` object is used.
`tick.last`	logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, attribute `tick.last` of the `x` object is used.
`glabels`	a vector of strings with the curve labels. If `NULL` curves are labeled with the levels of the group variable
`with.legend`	logical: Should the legend be plotted. Default is `TRUE`.
`legend.pos`	legend position: default is `"topright"`. See`legend` .
`horiz`	logical: Should the legend be displayed horizontally. Set as `FALSE` by default, i.e., legend is displayed vertically.
`cex.legend`	Scale factor for the legend.
`bcol`	color vector. For confidence bands. If `NULL` (default), colors are automatically assigned using `qualitative_hcl` with the `'Pastel 1'` palette (see `hcl_palettes`).
`na.rm`	logical. When `fstat` is `mean` or `weighted.mean`, should `NA`'s be stripped before computation? Ignored for any other `fstat` function.
`ret`	logical: Should the plotted values be returned?
`...`	additional plot parameters (see `par`).

Details

Together with `seqindic.dyn` this function implements the dynamic sequence analysis approach of *Pelletier et al. (2020).*

The function first computes the summary table using the `fstat` function. Each row of the summary table is then plotted as a line, except rows that contain `NA`s. Setting `na.rm=TRUE` helps sometimes to prevent some `NA`s in the summary table.

Confidence bands are computed for a confidence level of 95% and assuming a normal distribution.

Value

If `ret=TRUE`, a matrix with the successive group summaries (One row per group) and, when `conf=TRUE`, the matrices with the lower and upper bounds of the confidence intervals as attributes `L.grp` and `U.grp`.

Author(s)

Gilbert Ritschard

References

Pelletier, D., Bignami-Van Assche, S., & Simard-Gendron, A. (2020) Measuring Life Course Complexity with Dynamic Sequence Analysis, *Social Indicators Research* doi:10.1007/s11205-02002464y.

See Also

See Also `seqindic.dyn` (with examples)

Examples

```
## See examples on 'seqindic.dyn' help page
```

`plot.emlt` *Emlt Plots*

Description

Plots static and dynamic state structure from the outcome of `seqemlt`. Two types of plot are proposed: The evolution in time of the correlation between states, and the projection of situations (time-indexed states) on their principal planes.

Usage

```
## S3 method for class 'emlt'
plot(x, from, to, delay=NULL, leg=TRUE, type="cor", cex=0.7, compx=1, compy=2, ...)
```

Arguments

`x`	an object of class `emlt` as produced by `seqemlt`
`type`	character string: type of plot to be drawn. Possible types are `"cor"` for the evolution in time of the correlation between states, and `"pca"` for the projection of states/situations on their principal planes
`from`	vector of state labels: for type `"cor"`, origin state(s) to be considered.
`to`	state label: for type "cor", destination state.
`delay`	for type "cor", the delay (number of time periods) between `"from"` and `"to"` arguments. The correlation between state `"from"` at time t and `"to"` at t+`delay`. By default `delay` is 0.
`compx`	integer: for type `"pca"` first component, axis x
`compy`	integer: for type `"pca"` second component, axis y
`leg`	logical: should the legend be included
`cex`	numerical value: amount by which plotting text and symbols should be magnified relative to the default.
`...`	Arguments to be passed to methods, such as graphical parameters (see `par`)

Details

The evolution of the correlation reveals the evolution of the emlt Euclidean distance between the situations (time-indexed states) along the timeframe.

The `"pca"` components are the principal components of the emlt numerical coordinates of the sequences, see `seqemlt`.

Author(s)

Patrick Rousset, Senior researcher at Cereq, rousset@cereq.fr with the help of Matthias Studer

References

Rousset, Patrick and Jean-François Giret (2007), Classifying Qualitative Time Series with SOM: The Typology of Career Paths in France, in F. Sandoval, A. Prieto and M. Grana (Eds) *Computational and Ambient Intelligence*, Lecture Notes in Computer science, vol 4507, Berlin: Springer, pp 757-764.

Rousset, Patrick, Jean-François Giret and Yvette Grelet (2012) Typologies De Parcours et Dynamique Longitudinale, *Bulletin de méthodologie sociologique*, 114(1), 5-34.

Rousset, Patrick and Jean-François Giret (2008) A longitudinal Analysis of Labour Market Data with SOM, in J. Rabuñal Dopico, J. Dorado, & A. Pazos (Eds.) *Encyclopedia of Artificial Intelligence*, Hershey, PA: Information Science Reference, pp 1029-1035.

See Also

See Also `seqemlt` (with examples)

Examples

```
## See examples on 'seqemlt' help page
```

`plot.stslist.surv` *Plot method for objects produced by the seqsurv function*

Description

This is the plot method for objects of class *stslist.surv* produced by the `seqsurv` function.

Usage

```
## S3 method for class 'stslist.surv'
plot(x, cpal = NULL, ylab = NULL, xlab = NULL,
  xaxis = TRUE, yaxis = TRUE, xtstep = NULL, tick.last = NULL,
  cex.axis = 1, ...)
```

Arguments

`x`	An object of class `stslist.surv` as produced by the `seqsurv` function.
`cpal`	Vector of colors. Alternative color palette to be used for the drawn lines. The vector should be of length equal to the number of drawn survival curves, i.e., the number of selected states or number of groups when `x` was obtained with `per.state=TRUE`. When `cpal=NULL`, the default colors assigned to the `cpal` attribute of `x` are used.
`ylab`	Optional label for the y axis. If set as `NA`, no label is displayed. If `NULL`, a default label is used.
`xlab`	Optional label for the x axis. If set as `NA`, no label is displayed. If `NULL`, a default label is used.
`xaxis`	Logical. Should the x-axis be plotted. Default is `TRUE`.
`yaxis`	Logical. Should the y-axis be plotted. Default is `TRUE`.
`xtstep`	Optional interval at which the tick-marks of the x-axis are displayed. For example, with `xtstep = 3` a tick-mark is drawn at position 1, 4, 7, etc... The display of the corresponding labels depends on the available space and is dealt with automatically. If unspecified, the `xtstep` attribute of the `x` object is used.
`tick.last`	Logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, the `tick.last` attribute of the `x` object is used.
`cex.axis`	Expansion factor for the font size of the axis labels and names. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`...`	Further graphical parameters. For more details about the graphical parameter arguments, see `plot`, `plot.default` and `par`.

Details

This is the plot method for the output produced by the `seqsurv` function, i.e., objects of class *stslist.surv*. It displays the survival curves fitted for states in sequences.

Author(s)

Matthias Studer, Gilbert Ritschard, Pierre-Alexandre Fonta

See Also

`seqsurv`, `seqsplot`, `survfit`.

Examples

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
"Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.seq <- seqdef(biofam, 10:25, labels=biofam.lab)

## State survival plot
biofam.surv <- seqsurv(biofam.seq)
plot(biofam.surv)
```

`polyads` *Polyadic data example*

Description

This data set contains a data frame with 10 triads. The 10 first rows correspond to the 1st generation members, the next 10 to the second generation members and the last 10 to the third generation members.

Usage

```
data(polyads)
```

Format

A data frame with 30 rows and 11 columns.

Details

The first column is the generation variable Gen. The sequences of length 10 are in columns 2 to 11 named X1 to X10.

Author(s)

Gilbert Ritschard

`rowmode` *Modal state of a variable*

Description

Returns the modal state of a variable, e.g., the modal state in a sequence.

Usage

```
rowmode(v, except = NULL)
```

Arguments

`v`	A numerical or factor variable.
`except`	Vector of values that should be ignored; e.g., set `except="*"` to ignore missing states with default coding.

Details

The function tabulates the variable and returns the most frequent value.

Value

The modal value

Author(s)

Gilbert Ritschard

See Also

`table`.

Examples

```
data(actcal)
actcal.seq <- seqdef(actcal[1:10,13:24])
actcal.mod <- apply(as.matrix(actcal.seq), 1, rowmode)
head(actcal.mod)
```

`seqauto` *Auto-association between states*

Description

Computes auto-associations of order k = 1 to `order`, between current states and states lagged by k positions.

Usage

```
seqauto(seqdata, order = 1, measure = "cv")
```

Arguments

`seqdata`	A state sequence object or a data frame with sequential data in STS format.
`order`	Maximum wanted order of auto-association.
`measure`	Character string. Currently only `"cv"` (Cramer's v) is accepted.

Details

The function puts the data in `"SRS"` form by means of the `seqformat` function.

Value

A matrix with `order` rows and two columns: the auto-association and its p-value.

Warning

Function in development, not fully checked.

Author(s)

Gilbert Ritschard

See Also

`seqformat`

Examples

```
data(biofam)

biofam.seq <- seqdef(biofam[1:100,10:25])
aa <- seqauto(biofam.seq, order=5)
aa
```

`seqCompare` *BIC and Likelihood ratio test for comparing two sequence data*

Description

The function `seqCompare` computes the likelihood ratio test (LRT) and Bayesian Information Criterion (BIC) for comparing two groups within each of a series of set. The functions `seqBIC` and `seqLRT` are aliases that return only the BIC or the LRT.

Usage

```
seqCompare(seqdata, seqdata2=NULL, group=NULL, set=NULL,
    s=100, seed=36963, stat="all", squared="LRTonly",
    weighted=TRUE, opt=NULL, BFopt=NULL, method, ...)

seqLRT(seqdata, seqdata2=NULL, group=NULL, set=NULL, s=100,
    seed=36963, squared="LRTonly", weighted=TRUE, opt=NULL,
    BFopt=NULL, method, ...)

seqBIC(seqdata, seqdata2=NULL, group=NULL, set=NULL, s=100,
    seed=36963, squared="LRTonly", weighted=TRUE, opt=NULL,
    BFopt=NULL, method, ...)
```

Arguments

`seqdata`	Either a state sequence object (`stslist` created with `seqdef`) or a list of state sequence objects, e.g., `list(cohort1.seq,cohort2.seq,cohort3.seq)`.
`seqdata2`	Either a state sequence object (`stslist` or a list of state sequence objects. Must be `NULL` when `group` is not `NULL`. If not `NULL`, must be of same type than `seqdata`. See details.
`group`	Vector of length equal to number of sequences in `seqdata`. A dichotomous grouping variable. See details.
`set`	Vector of length equal to number of sequences in `seqdata`. Variable defining the sets. See details.
`s`	Integer. Default 100. The size of random samples of sequences. When 0, no sampling is done.
`seed`	Integer. Default 36963. Using the same seed number guarantees the same results each time. Set `s=NULL` if you don't want to set a seed. The random generator can be chosen with `RNGkind`.
`stat`	String. The requested statistics. One of `"LRT"`, `"BIC"`, or `"all"`
`squared`	Logical. Should squared distances be used? Can also be `"LRTonly"`, in which case the distances to the centers are computed using non-squared distances and LRT is computed with squared distances.

`weighted`	Logical or String. Should weights be taken into account when available? Can also be `"by.group"`, in which case weights are used and normalized to respect group sizes.
`opt`	Integer or `NULL`. Either 1 or 2. Computation option. When 1, the distance matrix is computed successively for each pair of samples of size s. When 2, the distances are computed only once for each pair of sets of observed sequences and the distances for the samples are extracted from that matrix. When `NULL` (default), 1 is chosen when the sum of sizes of the two groups is larger than 2*s and 2 otherwise.
`BFopt`	Integer or `NULL`. Either 1 or 2. Applies only when BIC is computed on multiple samples. When 1 the displayed Bayes Factor (BF) is the averaged BF. When 2, the displayed BF is obtained from the averaged BIC. When `NULL` both BFs are displayed.
`method`	String. Method for computing sequence distances. See documentation for `seqdist`. Additional arguments may be required depending on the method chosen.
`...`	Additional arguments passed to `seqdist`.

Details

The `group` and `set` arguments can only be used when `seqdata` is an `stslist` object (a state sequence object).

When `seqdata` and `seqdata2` are both provided, the LRT and BIC statistics are computed for comparing these two sets. In that case both `group` and `set` should be left at their default `NULL` value.

When `seqdata` is a list of `stslist` objects, `seqdata2` must be a list of the same number of `stslist` objects.

The default option `squared="LRTonly"` corresponds to the initial proposition of Liao and Fasang (2021). With that option, the distances to the virtual center are obtained from the pairwise non-squared dissimilarities and the resulting distances to the virtual center are squared when computing the LRT (which is in turn used to compute the BIC). With `squared=FALSE`, non-squared distances are used in both cases, and with `squared=TRUE`, squared distances are used in both cases.

The computation is based on the pairwise distances between the sequences. The `opt` argument permits to choose between two strategies. With `opt=1`, the matrix of distances is computed successively for each pair of samples of size s. When `opt=2`, the matrix of distances is computed once for the observed sequences and the distances for the samples are extracted from that matrix. Option 2 is often more efficient, especially for distances based on spells. It may be slower for methods such as OM or LCS when the number of observed sequences becomes large.

Value

The function `seqLRT` (and seqCompare with the default `"LRT"` stat value) outputs two variables, *LRT* and *p.LRT*.

`LRT`	This is the likelihood ratio test statistic for comparing the two groups.
`p.LRT`	This is the upper tail probability associated with the LRT.

The function `seqBIC` (and `seqLRT` with the `"BIC"` stat value) outputs two variables, *BIC* and *BF*.

`BIC`	This is the difference between two BICs for comparing the two groups.
`BF`	This is the Bayes factor associated with the BIC difference.

`seqCompare` with `stat="all"` outputs all four indicators.

Author(s)

Tim Liao and Gilbert Ritschard

References

Tim F. Liao & Anette E. Fasang (2021). "Comparing Groups of Life Course Sequences Using the Bayesian Information Criterion and the Likelihood Ratio Test.” *Sociological Methodology*, 55 (1), 44-85. doi:10.1177/0081175020959401.

Examples

```
## biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
                "Child", "Left+Child", "Left+Marr+Child", "Divorced")
alph <- seqstatl(biofam[10:25])
## To illustrate, we use only a sample of 150 cases
set.seed(10)
biofam <- biofam[sample(nrow(biofam),150),]
biofam.seq <- seqdef(biofam, 10:25, alphabet=alph, labels=biofam.lab)

## Defining the grouping variable
lang <- as.vector(biofam[["plingu02"]])
lang[is.na(lang)] <- "unknown"
lang <- factor(lang)

## Chronogram by language group
seqdplot(biofam.seq, group=lang)

## Extracting the sequence subsets by language
lev <- levels(lang)
l <- length(lev)
seq.list <- list()
for (i in 1:l){
  seq.list[[i]] <- biofam.seq[lang==lev[i],]
}

seqCompare(list(seq.list[[1]]),list(seq.list[[2]]), stat="all", method="OM", sm="CONSTANT")
seqBIC(biofam.seq, group=biofam$sex, method="HAM")
seqLRT(biofam.seq, group=biofam$sex, set=lang, s=80, method="HAM")
```

`seqcta` *Competing Trajectory Analysis (CTA)*

Description

Competing Trajectory Analysis (CTA) aims to simultaneously study the occurrence of an event and the trajectory following it over a pre-defined period of time. The `seqcta` function convert the data to run the analysis.

Usage

```
seqcta(seqdata, subseq = 5, time = NULL, event = NULL, initial.state = NULL, covar = NULL)
```

Arguments

`seqdata`	State sequence object created with the `seqdef` function. The whole trajectory followed by individuals.
`subseq`	Numeric. The length of the trajectory following the event to be considered.
`time`	Numeric. The time of occurrence of the event, can be `NA` for censored observations. If `NULL` (default), `initial.state` should be provided.
`event`	Logical. Whether the event occur for each trajectory. If `NULL` (default) and `time` is provided, `NA` `time` values are used to detect censored observations.
`initial.state`	Character. Only used if `time` is not provided. If provided, the end of the first spell of the sequence, but only in `initial.state` state, is used as the event of interest to compute the `time` argument.
`covar`	Optional `data.frame` storing covariates of interest. These covariates are added to the final data set.

Details

Competing Trajectory Analysis (CTA) works as follows. First, the sequence following the studied event are clustered. Second, the type of trajectory followed is linked with covariates using a competing risks model.

The `seqcta` function reorganizes the data to run CTA. More precisely, it provides a person-period data frame until the occurrence of the event. When the event occurs, the trajectory following it is also stored. Covariates specified using the `covar` arguments are also stored.

The example section below provides a step by step example of the whole procedure.

Value

A data frame with the following variables

`id`	Numeric. The ID of the observation as the row number in the original `seqdata`.
`time`	Numeric. The time unit from the beginning of the original sequence until the occurence of the event.

`event`	Logical. Whether the event occured within this time unit.
`lastobs`	Logical. Whether this is the last observation for an individual observation, censored or not. This is useful when one want only one row per individual, for instance to plot survival curves (see example).
`T1 until T...`	The state sequence following the event starting from 1 (time unit after the event) until `subseq` time units after the event. Only available for the rows where `event=TRUE`.
`Optional covariate list`	The covariates provided with the `covar` argument.

Author(s)

Matthias Studer

References

M. Studer, A. C. Liefbroer and J. E. Mooyaart, 2018. Understanding trends in family formation trajectories: An application of Competing Trajectories Analysis (CTA), *Advances in Life Course Research* 36, pp 1-12. doi:10.1016/j.alcr.2018.02.003

See Also

`seqsamm`, `seqsha`

Examples

```
## Create seq object for biofam data.
data(biofam)
bf.shortlab <- c("P","L","M","LM","C","LC", "LMC", "D")
bf.seq <- seqdef(biofam[,10:25], states=bf.shortlab)

## We focus on the occurrence of ending the first "P" spell and the trajectory that follows
## For the next subseq=5 years
## We also store the covariate sex and birthyr
## seqcta will transform the data to person-period until the end of the first "P" spell
## and store the following trajectory

cta <- seqcta(bf.seq, subseq=5, initial.state="P", covar=biofam[, c("sex", "birthyr")])
summary(cta)

## If the studied event is not a first state of the trajectory
## One can also provide the event using the time and event arguments
## Here we compute the time spent in "P" ourselves before providing it to seqcta

dur <- seqdur(bf.seq)
## If "P" is the first state, we use the time in this state (dur[, 1])
## Otherwise we use 0 (started immediatly at the beginning)
timeP <- ifelse(bf.seq[, 1]=="P", dur[, 1], 0)

## The event occured if timeP is inferior to the length of the sequence
## Otherwise they never left their parents.
```

```
eventP <- timeP < 16

cta2 <- seqcta(bf.seq, subseq=5, time=timeP, event=eventP, covar=biofam[, c("sex", "birthyr")])
##Identical results
summary(cta2)


## Not run to save computation time
## Not run:
library(survival)

## To plot a survival curve, we only need the last observation for each individual.
## Kaplan Meier curve for the occurrence of the event
ss <- survfit(Surv(time, event)~sex, data=cta, subset=lastobs)
plot(ss, col=1:2)

## Now we cluster the trajectories following the event
## Therefore we only keep lines where the event occured.
clusterTraj <- seqdef(cta[cta$event, 5:9])
##Compute distances
diss <- seqdist(clusterTraj, method="HAM")
##Clustering with pam
library(cluster)
pclust <- pam(diss, diss=TRUE, k=5, cluster.only=TRUE)
#Naming the clusters
pclustname <- paste("Type", pclust)
##Plotting the clusters to make senses of them.
seqdplot(clusterTraj, pclustname)

##Now we store back the clustering in the original person-period data
## We start by adding a variable storing "no event" for all lines
cta$traj.event <- "No event"
## Then we store the type of following trajectory
## only for those having experienced the event
cta$traj.event[cta$event] <- pclustname


## Checking the results
summary(cta)

## Now we can estimate a competing risk model
## Several strategies are available.
## Here we use multinomial model on the person period.

library(mlogit)
summary(mlogit(traj.event~1|time+sex, data=cta, shape="wide", reflevel="No event"))
library(nnet)
summary(multinom(traj.event~time+sex+scale(birthyr), data=cta))

## The model can also be estimated with cox regression
## However, we need to estimate one model for each competing risk
## ie. the type of following trajectory in our case.
```

```
## Compute the event variable for "Type 1"

cta$eventType1 <- cta$traj.event=="Type 1"
summary(coxph(Surv(time, eventType1)~sex+scale(birthyr), data=cta, subset=lastobs))



## End(Not run)
```

`seqe2stm` *Definition of an events to states matrix.*

Description

This function creates a matrix specifying for each state (given in row) to which state we fall when the event given in column happens.

Usage

```
seqe2stm(events, dropMatrix = NULL, dropList = NULL, firstState = "None")
```

Arguments

`events`	Character. The vector of all possible events.
`dropMatrix`	Logical matrix. Specifying the events to forget once a given event has occurred.
`dropList`	List. Same as `dropMatrix` but using a list (often more convenient).
`firstState`	Character. Name of the first state, before any event has occurred.

Details

This function creates a matrix with in each cell the new state which results when the column event (column name) occurs while we are in the corresponding row state (row name). Such a matrix is required by `TSE_to_STS`. By default, a new state is created for each combination of events that already has occurred.

`dropMatrix` and `dropList` allow to specify which events should be "forgotten" once a given event has occurred. For instance, we may want to forget the "marriage" event once the event "divorce" has occurred.

`dropMatrix` specifies for each event given in row, the previous events, given in column that should be forgotten. `dropList` uses a list to specify the same thing. The form is list(event1=c(..., events to forget), event2=c(..., events to forget)). See example below.

Value

A matrix.

Note

This function is a pre-release and further testing is still needed, please report any problems.

Author(s)

Matthias Studer

References

Ritschard, G., Gabadinho, A., Studer, M. & Müller, N.S. (2009), "Converting between various sequence representations", In Ras, Z. & Dardzinska, A. (eds) *Advances in Data Management*. Series: *Studies in Computational Intelligence*. Volume 223, pp. 155-175. Berlin: Springer.

See Also

`TSE_to_STS`

Examples

```
## Achieving same result using dropMatrix or dropList.
## List of possible events.
events <- c("marr", "child", "div")
dm <- matrix(FALSE, 3,3, dimnames=list(events, events))
dm[3, ] <- c(TRUE, TRUE, FALSE)
dm[1, 3] <- TRUE
## Using the matrix, we forget "marriage" and "child" events when "divorce" occurs.
## We also forget "divorce" after "marriage" occurs.
print(dm)
stm <- seqe2stm(events, dropMatrix=dm)

## Get same result with the dropList argument.
stmList <- seqe2stm(events, dropList=list("div"=c("marr", "child"), "marr"="div"))

## test that the results are the same
all.equal(stm, stmList)
```

`seqedist` *Distances between event sequences*

Description

Compute Optimal-Matching-like distances between event sequences. The distance measure is described in *Studer et al. 2010*.

Usage

```
seqedist(seqe, idcost, vparam, interval="No", norm="YujianBo")
```

Arguments

`seqe`	An event sequence `seqelist` object as defined by the `seqecreate` function.
`idcost`	Insertion/deletion cost of the different events (one entry per element of the event alphabet).
`vparam`	Positive real. The cost for a one-unit change in the time stamp of an event.
`norm`	Character. One of `"YujianBo"` (respects triangle inequality), `"max"` (maximum distance) or `"none"`.
`interval`	Character. One of `"No"` (absolute ages), `"previous"` (time spent since previous event) or `"next"` (time spent until next event).

Value

a distance matrix.

Author(s)

Matthias Studer

References

Studer, M., Müller, N.S., Ritschard, G. & Gabadinho, A. (2010), "Classer, discriminer et visualiser des séquences d'événements", In Extraction et gestion des connaissances (EGC 2010), *Revue des nouvelles technologies de l'information RNTI*. Vol. E-19, pp. 37-48.

Ritschard, G., Bürgin, R., and Studer, M. (2014), "Exploratory Mining of Life Event Histories", In McArdle, J.J. & Ritschard, G. (eds) *Contemporary Issues in Exploratory Data Mining in the Behavioral Sciences*. Series: Quantitative Methodology, pp. 221-253. New York: Routledge.

Examples

```
data(actcal.tse)
actcal.seqe <- seqecreate(actcal.tse[1:200,])[1:6,]
## We have 8 different events in this dataset
idcost <- rep(1, 8)
dd <- seqedist(actcal.seqe, idcost=idcost, vparam=.1)
```

`seqedplot` *Graphical representation of a set of events sequences.*

Description

This function provides two ways to represent a set of events. The first one (`type="survival"`) plots the survival curves of the first occurrence of each event. The second one (`type="hazard"`) plots the mean counts of each events in a given time frame.

Usage

```
seqedplot(seqe, group = NULL, breaks = 20, ages = NULL, main = NULL,
   type = "survival", ignore = NULL, withlegend = "auto", cex.legend = 1,
   use.layout = (!is.null(group) | withlegend != FALSE),
   legend.prop = NA, rows = NA, cols = NA, axes = "all", xlab = "time",
  ylab = ifelse(type == "survival", "survival probability", "mean number of events"),
   cpal = NULL, title, ...)
```

Arguments

`seqe`	an event sequence object as defined by the `seqecreate` function.
`group`	Plots one plot for each level of the factor given as argument.
`breaks`	Number of breaks defining a period.
`ages`	Two numeric values representing minimum and maximum ages to be represented.
`main`	title for the graphic. Default is `NULL`.
`type`	the type of the plot. If `type="survival"`, plots the survival curves of the first occurrence of each event. If `type="hazard"`, plots the mean numbers of each event in a given time frame.
`ignore`	Character. An optional list of events that will not be plotted.
`withlegend`	defines if and where the legend of the state colors is plotted. The default value `"auto"` sets the position of the legend automatically. Other possible values are `"right"` or `FALSE`. Obsolete value `TRUE` is equivalent to "auto".
`cex.legend`	expansion factor for setting the size of the font for the labels in the legend. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`use.layout`	if `TRUE`, layout is used to arrange plots when using the group option or plotting a legend. When layout is activated, the standard `par(mfrow=....)` for arranging plots does not work. With `withlegend=FALSE` and `group=NULL`, layout is automatically deactivated and `par(mfrow=....)` can be used.
`legend.prop`	proportion of the graphic area used for plotting the legend when `use.layout=TRUE` and `withlegend=TRUE`. Default value is set according to the place (bottom or right of the graphic area) where the legend is plotted. Values from 0 to 1.
`rows`	optional arguments to arrange plots when use.layout=TRUE.
`cols`	optional arguments to arrange plots when use.layout=TRUE.
`axes`	if set to "all" (default value) x-axes are drawn for each plot in the graphic. If set to "bottom" and group is used, axes are drawn only under the plots located at the bottom of the graphic area. If FALSE, no x-axis is drawn.
`xlab`	an optional label for the x-axis. If set to `NA`, no label is drawn.
`ylab`	an optional label for the y-axis. If set to `NA`, no label is drawn.
`cpal`	Color palette used for the events. If `NULL`, a new color palette is generated.
`title`	Deprecated. Use `main` instead.
`...`	Additional arguments passed to `lines`.

Author(s)

Matthias Studer

References

Studer, M., Müller, N.S., Ritschard, G. & Gabadinho, A. (2010), "Classer, discriminer et visualiser des séquences d'événements", In Extraction et gestion des connaissances (EGC 2010), *Revue des nouvelles technologies de l'information RNTI*. Vol. E-19, pp. 37-48.

Examples

```
data(actcal.tse)
actcal.tse <- actcal.tse[1:200,]
iseq <- unique(actcal.tse$id)
nseq <- length(iseq)
data(actcal)
actcal <- actcal[rownames(actcal) %in% iseq,]
actcal.seqe <- seqecreate(actcal.tse)
seqelength(actcal.seqe) <- rep(12, nseq)
seqedplot(actcal.seqe, type="hazard", breaks=6, group=actcal$sex, lwd=3)
seqedplot(actcal.seqe, type="survival", group=actcal$sex, lwd=3)
```

`seqemlt` *Euclidean Coordinates for Longitudinal Timelines*

Description

Computes the Euclidean coordinates of sequences from which we get the EMLT distance between sequences introduced in Rousset et al (2012).

Usage

```
seqemlt(seqdata, a = 1, b = 1, weighted = TRUE)
```

Arguments

`seqdata`	a state sequence object defined with the `seqdef` function.
`a`	optional argument for the weighting mechanism that controls the balancing between short term/long term transitions. The weighting function is $1/(a*s+b)$ where s is the transition step.
`b`	see argument `a`.
`weighted`	Logical: Should weights in the sequence object `seqdata` be used?

Details

The EMLT distance is the sum of the dissimilarity between the pairs of states observed at the successive positions, where the dissimilarity between states is defined at each position as the Chi-squared distance between the normalized vectors of transition probabilities (profiles of situations) from the current state to the next observed states in the sequence. Transition probabilities are down-weighted with the time distance to avoid exaggerated importance of transitions over long periods. The adjustment weight is $1/a * s + b$, where s is the period length over which the transition probability is measured.

The EMLT distance between two sequences is obtained as the Euclidean distance between the returned numerical sequence coordinates. So, providing `coord` as the data input to any clustering algorithm that uses the Euclidean metric is equivalent to cluster with the EMLT metric.

Each time-indexed state is called a situation, and the distance between two states at a position t is derived from the transition probabilities to other observed situations.

The distance between any situation and a situation that does not occur is coded as `NA`. Such non-occurring situations have no influence on the distance between sequences.

The obtained numerical representations of sequences may be used as input to any Euclidean algorithm (clustering algorithms, ...).

Value

An object of class `emlt` with the following components:

`coord`	Matrix with in each row the EMLT numerical coordinates of the corresponding sequence.
`states`	list of states
`situations`	list of situations (timestamped states)
`sit.freq`	Situation frequencies
`sit.transrate`	matrix of transition probabilities from each situation to future situations
`sit.profil`	profiles of situations. Each profile is the normalized vector of transition probabilities to future situations adjusted to down weight transitions over longer periods.
`sit.cor`	Matrix of correlations between situations. Two situations are highly correlated when their profiles are similar (i.e., when their transitions towards future are similar).

Author(s)

Patrick Rousset, Senior researcher at Cereq, rousset@cereq.fr with the help of Matthias Studer. Help page by Gilbert Ritschard.

References

Rousset, Patrick and Jean-François Giret (2007), Classifying Qualitative Time Series with SOM: The Typology of Career Paths in France, in F. Sandoval, A. Prieto and M. Grana (Eds) *Computational and Ambient Intelligence*, Lecture Notes in Computer science, vol 4507, Berlin: Springer, pp 757-764.

Rousset, Patrick, Jean-François Giret and Yvette Grelet (2012) Typologies De Parcours et Dynamique Longitudinale, *Bulletin de méthodologie sociologique*, 114(1), 5-34.

Rousset, Patrick and Jean-François Giret (2008) A longitudinal Analysis of Labour Market Data with SOM, in J. Rabuñal Dopico, J. Dorado, & A. Pazos (Eds.) *Encyclopedia of Artificial Intelligence*, Hershey, PA: Information Science Reference, pp 1029-1035.

Studer, Matthias and Gilbert Ritschard (2014) A comparative review of sequence dissimilarity measures. LIVES Working Paper, 33 doi:10.12682/lives.22961658.2014.33

See Also

`plot.emlt`

Examples

```
data(mvad)
mvad.seq <- seqdef(mvad[1:100, 17:41])
alphabet(mvad.seq)
head(labels(mvad.seq))
## Computing distance
mvad.emlt <- seqemlt(mvad.seq)

## typology1 with kmeans in 3 clusters
km <- kmeans(mvad.emlt$coord, 3)

##Plotting by clusters of typology1
seqdplot(mvad.seq, group=km$cluster)

## typology2: 3 clusters by applying hierarchical ward
##   on the centers of the 25 group kmeans solution
km<-kmeans(mvad.emlt$coord, 25)
hc<-hclust(dist(km$centers, method="euclidean"), method="ward")
zz<-cutree(hc, k=3)

##Plotting by clusters of typology2
seqdplot(mvad.seq, group=zz[km$cluster])

## Plotting the evolution of the correlation between states
plot(mvad.emlt, from="employment", to="joblessness", type="cor")
plot(mvad.emlt, from=c("employment","HE", "school", "FE"), to="joblessness", delay=0, leg=TRUE)
plot(mvad.emlt, from="joblessness", to="employment", delay=6)
plot(mvad.emlt, type="pca", cex=0.4, compx=1, compy=2)
```

`seqentrans` *Event sequence length and number of events*

Description

Adds the sequence length (number of transitions) and total number of events of event sequences to the data attribute of a `subseqelist` event sequence object.

Usage

```
seqentrans(fsubseq, avg.occ = FALSE)
```

Arguments

`fsubseq`	A `subseqelist` object as returned by `seqefsub`.
`avg.occ`	Logical: Should a column with average number of occurrences also be added?

Details

An event sequence object is an ordered list of transitions, with each transition a non-ordered list of events occurring at a same position.

Average occurrences by sequence may be useful when counts report number of occurrences rather than number of sequences containing the subsequence.

Value

The object `fsubseq` updated with the additional information.

Author(s)

Nicolas Müller and Gilbert Ritschard

Examples

```
data(actcal.tse)
actcal.seqe <- seqecreate(actcal.tse[1:500,])

##Searching for frequent subsequences appearing at least 10 times
fsubseq <- seqefsub(actcal.seqe, min.support=10)
fsubseq <- seqentrans(fsubseq)
## dispaying only those with at least 3 transitions
fsubseq[fsubseq$data$ntrans>2]
## dispaying only those with at least 3 events
fsubseq[fsubseq$data$nevent>2]

## Average occurrences when counting distinct occurrences
ct <- seqeconstraint(count.method="CDIST_O")
fsb <- seqefsub(actcal.seqe, min.support=10, constraint=ct)
fsb <- seqentrans(fsb, avg.occ=TRUE)
fsb[1:10,]
```

`seqerulesdisc` *Extract association rules using discrete time regression models*

Description

Extract association rules from an object created by the `createdatadiscrete` function, using discrete time regression models to assess the significance of the extracted rules.

Usage

```
seqerulesdisc(fsubseq, datadiscr, tsef, pvalue=0.1, supvars=NULL,
   adjust=TRUE, topt=FALSE, link="cloglog", dep=NULL)
```

Arguments

`fsubseq`	an object created using the `seqefsub` function and that contains the list of subsequences to be tested for an association
`datadiscr`	the object created by the `createdatadiscrete` function and that contains the person-period data
`tsef`	the data frame containing the original time-to-event dataset (equivalent to the `data` argument from the `createdatadiscrete` function)
`pvalue`	the default threshold p-value to consider an association rule as significative, default is 0.1
`supvars`	a vector of variable names to be used as control variables in the regression models (experimental)
`adjust`	if set to TRUE, a Bonferroni adjustment is applied to the p-value threshold specified in the `pvalue` argument
`topt`	if set to TRUE, use an alternative algorithm to extract the rules (very experimental) ; default to FALSE
`link`	the link function to be used in the generalized linear regression model. To obtain hazard ratios, use the complementary log-log link function ("cloglog", as default). The other choice is to use a logit link function ("logit").
`dep`	if set to NULL, test all possible association rules. If an event is set, the function will only test association rules ending with this event

Details

This function uses a list of subsequences created by the `seqefsub` function from the TraMineR package and tests each possible association rules. It then shows the association rules whose significance, assessed using a discrete time regression model, is higher than the specified p-value threshold.

The algorithm is described in the Müller et al. (2010) article, even though this function uses a discrete time regression model instead of the Cox regression model described in the article. A more complete explanation of the method is available in Müller (2011).

Value

a list with one person-period data frame by event, where the dependent event is different each time. Please see the attached data file and code for an example.

Author(s)

Nicolas S. Müller

References

Müller, N.S., M. Studer, G. Ritschard et A. Gabadinho (2010), Extraction de règles d'association séquentielle à l'aide de modèles semi-paramétriques à risques proportionnels, *Revue des Nouvelles Technologies de l'Information*, **Vol. E-19**, EGC 2010, pp. 25-36.

Müller, N.S. (2011), Inégalités sociales et effets cumulés au cours de la vie : concepts et méthodes, *Thèse de doctorat, Faculté des sciences économiques et sociales, Université de Genève*, http://archive-ouverte.unige.ch/unige:17746.

See Also

`createdatadiscrete` to create the object needed as the `datadiscr` argument. `seqefsub` to create the object needed as the `fsubseq` argument.

Examples

```
##
```

`seqgen.missing` *Generate random missing states within a state sequence object*

Description

The function assigns missing values (`nr` attribute of the object, which is `"*"` by default) to randomly selected positions in randomly selected cases.

Usage

```
seqgen.missing(seqdata, p.cases = 0.1, p.left = 0.2, p.gaps = 0, p.right = 0.3,
          mt.left="nr", mt.gaps="nr", mt.right="nr")
```

Arguments

`seqdata`	A state sequence object
`p.cases`	Proportion of cases with missing values.
`p.left`	Proportion of left missing values.
`p.gaps`	Proportion of gap missing values.
`p.right`	Proportion of right missing values.

`mt.left`	Type of left missing. One of `"nr"` (non response state) or `"void"`
`mt.gaps`	Type of gap missing. One of `"nr"` (non response state) or `"void"`
`mt.right`	Type of right missing. One of `"nr"` (non response state) or `"void"`

Details

The aim of the function is essentially pedagogical. It may serve to illustrate how results of a sequential analysis may be affected by the presence of random missing states.

States in the sequences are randomly replaced with missing values. For each selected sequence, first, a random proportion between 0 and `p.gaps` of gaps are randomly inserted, then a random proportion between 0 and `p.left` of positions from the start of the sequence are set as missing, and finally a random proportion between 0 and `p.right` of positions from the end of the sequence are set as missing. Left missing values may possibly overlap gaps, and right missing values may overlap gaps and/or right missing values.

Value

The resulting state sequence object.

Warning

This function needs further testing.

Author(s)

Gilbert Ritschard

See Also

`seqdef`

Examples

```
## create the biofam.seq state sequence object from the biofam data.
data(biofam)
biofam.seq <- seqdef(biofam[1:100,10:25])

## Generate missing states within 60% of the sequences.
biofamm.seq <- seqgen.missing(biofam.seq, p.cases=.6,
      p.left=.4, p.gaps=.2, p.right=.5)

## compare the rendering of the sequences before and after
##   introducing missing states.
par(mfrow=c(2,2))
seqIplot(biofam.seq, sortv="from.end", with.legend=FALSE)
seqIplot(biofamm.seq, sortv="from.end", with.legend=FALSE)
seqdplot(biofam.seq, with.missing=TRUE, border=NA, with.legend=FALSE)
seqdplot(biofamm.seq, with.missing=TRUE, border=NA, with.legend=FALSE)
dev.off()
```

`seqgranularity` *Changing sequence time granularity by aggregating positions*

Description

Changes time granularity of a state sequence object by aggregating successive positions into groups of a user-defined time length.

Usage

```
seqgranularity(seqdata, tspan = 3, method = "last")
```

Arguments

`seqdata`	A state sequence object.
`tspan`	Integer. Number of successive positions grouped together.
`method`	Character string. Aggregating method. One of `"first"`, `"first.valid"`, `"last"` (default), `"last.valid"`, \ or `"mostfreq"`.

Details

Successive positions are aggregated by group of `tspan` states. The aggregated state is, depending of the `method` chosen, either the first (`"first"`), the first valid (`"first.valid"`), the last (`"last"`), the last valid (`"last.valid"`), or the most frequent (`"mostfreq"`) state of the `tspan` long spell. The same applies to the last spell, even when it is shorter than `tspan`.

Methods (`"first"`) and (`"last"`) differ from (`"first.valid"`) and (`"last.valid"`) only when sequences contain missing values and/or have different lengths.

When there are (void or non void) missings, method `"mostfreq"` replaces each interval with the most frequent valid state on the interval or the missing state when there are no valid state.

End missings are set as void when there are voids in `seqdata` and as the `nr` attribute otherwise.

Value

A `stslist` object: The compacted state sequence object.

Author(s)

Matthias Studer and Gilbert Ritschard

See Also

`seqdef`

Examples

```
data(mvad)
mvad <- mvad[1:100,]
mvad.seq <- seqdef(mvad[,17:86], xtstep=12)
mvadg.seq <- seqgranularity(mvad.seq, tspan=6, method="first")
par(mfrow=c(2,1))
seqdplot(mvad.seq, with.legend=FALSE, border=NA)
seqdplot(mvadg.seq, with.legend=FALSE)
```

`seqimplic` *Position wise group-typical states*

Description

Visualization and identification of the states that best characterize a group of sequences versus the others at each position (time point). The typical states are identified at each position as those for which we have a high implication strength to be in when belonging to the group.

Usage

```
seqimplic(seqdata, group, with.missing = FALSE, weighted = TRUE, na.rm = TRUE)
## S3 method for class 'seqimplic'
plot(x, main = NULL, ylim = NULL, xaxis = TRUE,
    ylab = "Implication", yaxis = TRUE, axes = "all", xtlab = NULL,
    xtstep = NULL, tick.last = NULL, cex.axis = 1, with.legend = "auto",
    ltext = NULL, cex.legend = 1, legend.prop = NA, rows = NA, cols = NA,
    conf.level = 0.95, lwd = 1, only.levels = NULL, ...)
## S3 method for class 'seqimplic'
print(x, xtstep = NULL, tick.last = NULL, round = NULL,
    conf.level = NULL, na.print = "", ...)
```

Arguments

`seqdata`	a state sequence object (see `seqdef`).
`group`	a factor giving the group membership of each sequence in `seqdata`.
`with.missing`	Logical. If `FALSE` (default), missing values are discarded. If `TRUE`, missing values are coded as a specific state.
`weighted`	Logical. If `TRUE` (default), the implicative strength of the rules are computed using the weights assigned to the state sequence object (see `seqdef`). Set as `FALSE` to ignore the weights.
`na.rm`	Logical. If `TRUE` (default), observations with missing values on the `group` variable are discarded. If `FALSE`, the missing group value defines a specific group.
`x`	A sequence of typical state object as generated by `seqimplic`.

`xtstep`	Integer. Optional interval at which the tick-marks and labels of the x-axis are displayed. For example, with `xtstep=3` a tick-mark is drawn at position 1, 4, 7, etc... The display of the corresponding labels depends on the available space and is dealt with automatically. If unspecified, the `xtstep` attribute of the `x` object is used.
`tick.last`	Logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, the `tick.last` attribute of the `x` object is used.
`main`	title for the graphic. Default is `NULL`.
`ylim`	
`xaxis`	Logical. Should the x-axis (time) be plotted?.
`ylab`	Optional label for the y-axis. If set as `NA`, no label is drawn.
`yaxis`	Logical. Should the y axis be plotted?. When set as `TRUE`, sequence indexes are displayed.
`axes`	If set as `"all"` (default value) x-axes are drawn for each plot in the graphic. If set as `"bottom"`, axes are drawn only under the plots located at the bottom of the graphic area. If `FALSE`, no x-axis is drawn.
`xtlab`	optional labels for the x-axis ticks labels. If unspecified, the column names of the `seqdata` sequence object are used (see `seqdef`).
`cex.axis`	expansion factor for setting the size of the font for the axis labels and names. The default value is 1. Values lesser than 1 will reduce the size of the font, values greater than 1 will increase the size.
`with.legend`	One of `"auto"` (default), `"right"` or `FALSE`. Defines if and where the legend of the state colors is plotted. With `"auto"` sets the position of the legend is set automatically. The obsolete value `TRUE` is equivalent to `"auto"`.
`ltext`	optional description of the states to appear in the legend. Must be a vector of character strings with number of elements equal to the size of the alphabet. If unspecified, the `label` attribute of the `seqdata` sequence object is used (see `seqdef`).
`cex.legend`	expansion factor for setting the size of the font for the labels in the legend. The default value is 1. Values smaller than 1 reduce the size of the font, values greater than 1 increase the size.
`legend.prop`	Proportion (between 0 and 1) of the graphic area used for plotting the legend when `use.layout=TRUE` and `withlegend=TRUE`. The default value is set according to the place (bottom or right of the graphic area) where the legend is plotted.
`rows,cols`	optional arguments to arrange plots when `use.layout=TRUE`.
`lwd`	The line width, a positive number. See `lines`
`only.levels`	Optional list of levels of the `group` variable to be plotted. By default all levels are plotted.
`round`	Optional number of decimals when printing a `seqimplic` object.
`conf.level`	Confidence levels thresholds (default is 0.95).
`na.print`	Character string (or `NULL`) used for `NA` values in printed output, see `print.default`.
`...`	further arguments passed to `print.default` (for `print` method) or `lines` (for `plot` method).

Details

The `seqimplic` function builds an object with the position wise typical states. It can be used to visualize or identify the differences between each group of trajectories and the other ones. It presents at each time point the typical states of a subpopulation (for instance women, as opposed to men). A state at a given time point is considered to be typical of a group if the rule "Being in this group implies to be in that state at this time point" is relevant according to the implicative statistic.

The implicative statistic assesses the statistical relevance of a rule of the form "A implies B" (Gras et al., 2008). It does so by measuring the gap between the expected and observed numbers of counter examples. The rule is considered to be strongly implicative if we observe much less counter examples than expected under the independence assumption. This gap and its significance are computed using adjusted residuals of a contingency table with continuity correction as proposed by Ritschard (2005). In order to improve the readability of the graphs, we use here the opposite of the implicative statistic, which is highly negative for significant rules. The statistic $I(A \to B)$ measuring the relevance of the rule "A implies B" reads as follows:

$$I(A \to B) = -\frac{n_{\bar{B}A} + 0.5 - n^{e}_{\bar{B}A}}{\sqrt{n^{e}_{\bar{B}A}(n_{B.}/n)(1 - n_{.A}/n)}}$$

Where $n_{\bar{B}A}$ is the observed number of counter-examples, $n^{e}_{\bar{B}A}$ the expected number of counter-examples under the independence assumption, $n_{B.}$ the number of times that B is observed, $n_{.A}$ the number of times that A is observed and n the total number of cases.

The plot function can be used to visualize the results. It produces a separate plot for each level of the `group` variable. In each plot, it presents at each time point t, the relevance of the rule "Being in this group implies to be in this state at this time point". The higher the plotted value, the higher the relevance of the rule. The horizontal dashed lines indicate the confidence thresholds. A rule is considered as statistically significant at the 5% level if it exceeds the 95% confidence horizontal line. The strength of rules with negative implicative statistic are not displayed because they have no meaningful interpretation.

Value

`seqimplic` returns a `"seqimplic"` object that can be plotted and printed. The values of the implicative statistics at each time point are in the element `indices` of the object.

Author(s)

Matthias Studer.

References

Studer, Matthias (2015), Comment: On the Use of Globally Interdependent Multiple Sequence Analysis, Sociological Methodology 45, doi:10.1177/0081175015588095.

Gras, Régis and Kuntz, Pascale. (2008), An overview of the Statistical Implicative Analysis (SIA) development, in Gras, R., Suzuki, E., Guillet, F. and Spagnolo, F. (eds), Statistical Implicative Analysis: Theory and application, Series Studies in Computational Intelligence, Vol 127, Berlin: Springer-Verlag, pp 11-40.

Ritschard, G. (2005). De l'usage de la statistique implicative dans les arbres de classification. In Gras, R., Spagnolo, F., and David, J., editors, Actes des Troisièmes Rencontres Internationale ASI Analyse Statistique Implicative, volume Secondo supplemento al N.15 of Quaderni di Ricerca in Didattica, pages 305–314. Università a degli Studi di Palermo, Palermo.

Examples

```
data(mvad)

## Building a state sequence object
mvad.seq <- seqdef(mvad, 17:86)
## Sequence of typical states
mvad.si.gcse5eq <- seqimplic(mvad.seq, group=mvad$gcse5eq)

##Plotting the typical states
plot(mvad.si.gcse5eq, lwd=3, conf.level=c(0.95, 0.99))

## Printing the results
print(mvad.si.gcse5eq, xtstep=12)
```

`seqindic.dyn` *Dynamic index*

Description

Dynamic (i.e. successive) values of an individual index. For each sequence, the values of the selected index is computed on sliding windows.

Usage

```
seqindic.dyn(seqdata, indic="cplx", window.size = .2, sliding=TRUE,
     wstep=1, with.missing=FALSE, endmiss.as.void=FALSE, silent.indic=TRUE, ...)
```

Arguments

`seqdata`	state sequence object (`stslist`) as produced by `seqdef`
`indic`	character string: the individual index. Can be any value supported by `seqindic` except index group names.
`window.size`	integer or real. If an integer > 1, window size in number of positions. If real number in the range]0,1), the window size is set as that proportion of the length of the longest sequence.
`sliding`	logical: Should indic be computed on sliding windows? If `FALSE`, windows are incremented starting with a window of size `window.size`.
`wstep`	integer: size of position gap between successive windows.
`with.missing`	logical. Should the missing state be treated as a state of the alphabet?

`endmiss.as.void`	logical. When `with.missing=FALSE`, should missings at end of windows be considered as voids, i.e. should the sequence end before end missings? When `FALSE` (default), the index is set as `NA` for windows that end with a void and, when `TRUE`, for windows that end with a void or a missing. Ignored when `with.missing=TRUE`.
`silent.indic`	logical. Should messages issued during computation of indic be suppressed?
`...`	additional arguments passed to `seqindic`

Details

The function implements the dynamic sequence analysis approach of *Pelletier et al. (2020)* and generalizes the method to any of the over 20 indicators provided by `seqindic`.

The values of the `indic` index are computed for each sequence either on sliding windows of size `window.size` or on incremental windows starting from a first window of size `window.size`.

Column names refer to the end the windows.

Value

A matrix of class `dynin` with attributes `xtstep`, `tick.last`, `weights`, `window.size`, `sliding`, and `indic`. The first three as well as the row and column names are taken from `seqdata`.

There are `print` and `plot` methods for `dynin` objects. See `plot.dynin`.

Author(s)

Gilbert Ritschard

References

Pelletier, D., Bignami-Van Assche, S., & Simard-Gendron, A. (2020) Measuring Life Course Complexity with Dynamic Sequence Analysis, *Social Indicators Research* doi:10.1007/s11205-02002464y.

See Also

`seqindic`, `plot.dynin`

Examples

```
data(actcal)
cases <- 1:100
actcal <- actcal[cases,] ## Here, only a subset
actcal.seq <- seqdef(actcal[,13:24], alphabet=c('A','B','C','D'))

## Using windows every three positions
a.dyn <- seqindic.dyn(actcal.seq, indic='cplx', with.missing=FALSE, wstep=3)
plot(a.dyn, group=actcal[cases,'sex'])

## Trimmed mean (to illustrate fstat with specific arguments)
```

```
plot(a.dyn, group=actcal[cases,'sex'], fstat=function(x)mean(x, trim=.02))

## Incremental windows
ai.dyn <- seqindic.dyn(actcal.seq, indic='cplx', with.missing=FALSE, wstep=3,
                sliding=FALSE)
plot(ai.dyn, group=actcal[cases,'sex'])

#############
## Sequences of different lengths, and with missing values and weights
data(ex1)
s.ex1 <- seqdef(ex1[,1:13],weights=ex1[,"weights"])
seqlength(s.ex1)
seqlength(s.ex1, with.missing=FALSE)
group <- c(1,1,1,2,2,2,2)

ind.d <- seqindic.dyn(s.ex1, indic='cplx', with.missing=FALSE)
plot(ind.d, group=group, fstat=weighted.mean, na.rm=TRUE, conf=TRUE, ret=TRUE)

## Treating 'missing' as a regular state
ind.dm <- seqindic.dyn(s.ex1, indic='cplx', with.missing=TRUE)
plot(ind.dm, group=group, fstat=weighted.mean, na.rm=TRUE, conf=TRUE, ret=TRUE)
```

`seqplot.rf` *Relative Frequency Sequence Plots.*

Description

Relative Frequency Sequence Plots (RFS plots) plot a selection of representative sequences as sequence index plots (see `seqIplot`). RFS plots proceed in several steps. First a set of sequences is ordered according to a substantively meaningful principle, e.g. according to their score on the first factor derived by applying Multidimensional scaling (default) or a user defined sorting variable, such as the timing of a transition of interest. Then the sorted set of sequences is partitioned in to k equal sized frequency groups. For each frequency group the medoid sequence is selected as a representative. The selected representatives are plotted as sequence index plots. RFS plots come with an additional distance-to-medoid box plot that visualizes the distances of all sequences in a frequency group to their respective medoid. Further, an R2 and F-statistic are given that indicate how well the selected medoids represent a given set of sequences.

Usage

```
seqplot.rf(seqdata, k = floor(nrow(seqdata)/10), diss, sortv = NULL,
    ylab=NA, yaxis=FALSE, main=NULL, which.plot="both",
    grp.meth = "first", ...)
```

Arguments

`seqdata`	a state sequence object created with the `seqdef` function.
`k`	integer: Number of groupings (frequency groups?)

`diss`	matrix of pairwise dissimilarities between sequences in `seqdata` (see `seqdist`).
`sortv`	an optional sorting variable that may be used to compute the frequency groups. If `NULL`, an MDS is used. Ties are randomly ordered.
`ylab`	string. An optional label for the y-axis. If set as `NA` (default), no label is drawn. Does not apply to `which.plot="both"`.
`yaxis`	logical. Controls whether a y-axis is plotted. When set as `TRUE`, the indexes of the sequences are displayed.
`main`	main graphic title. Default is `NULL`.
`which.plot`	string. One of `"both"`, `"medoids"`, `"diss.to.med"`. When `"medoids"`, only the index plot of the medoids is displayed, when `"diss.to.med"`, the grouped boxplots of the distances to the medoids is displayed, and when `"both"` a combined plot of the two is displayed.
`grp.meth`	character string. One of `"first"` or `"random"`. When the number of sequences is not a multiple of the number groups, which groups should have their size augmented by one unit? If `"first"`, the first groups, and if `"random"` a random selection of groups.
`...`	arguments passed to `seqplot`.

Details

RFS plots are useful to visualize large sets of sequences that cannot be plotted with sequence index plots due to overplotting (see `seqIplot`). Due to the partitioning into equal sized frequency groups each selected sequence represents an equal portion of the original sample and thereby visually maintains the relative proportion of different types of sequences along the sorting criterion. The ideal number of k fequency groups depends on the size of the original sample and the empirical distribution of the sequences. The larger the sample and the more heterogeneous the sequences, higher numbers of k will be advisable. To avoid overplotting k should generally not be higher than 200.

Note that distance-to-medoid plots are meaningful only if there are at least 5-10 sequences in each frequency group. The distance-to-medoid plot is not only a quality criterion of how well the medoids represent a respective frequency group. They also provide additional substantive information about how large the variation of sequences is at a given location of the ordered sequences (see Fasang and Liao 2014).

Since ties in `sortv` or mds are randomly ordered (see argument `ties.method="random"` of function `rank`), one has to set the seed to reproduce exactly the same plot (see `set.seed`).

Unlike the other `TraMineR` plotting functions, the `seqplot.rf` function ignores the `weights` and does not support the `group` argument.

Value

A vector with the group membership (medoid of the group) of each sequence.

Author(s)

Matthias Studer, Anette Eva Fasang, Tim Liao, and Gilbert Ritschard.

References

Fasang, Anette Eva and Tim F. Liao. 2014. "Visualizing Sequences in the Social Sciences: Relative Frequency Sequence Plots." Sociological Methods & Research 43(4):643-676.

See Also

See also `seqplot`, `seqrf`, `seqrep`.

Examples

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
"Child", "Left+Child", "Left+Marr+Child", "Divorced")

## Here, we use only 100 cases selected such that all elements
## of the alphabet be present.
## (More cases and a larger k would be necessary to get a meaningful example.)
biofam.seq <- seqdef(biofam[501:600, ], 10:25, labels=biofam.lab)
diss <- seqdist(biofam.seq, method="LCS")

## Using 12 groups and default MDS sorting
seqplot.rf(biofam.seq, diss=diss, k=12,
   main="Non meaningful example (n=100)")


## With a user specified sorting variable
## Here time spent in parental home: there are ties
## We set a seed because of random order in ties
set.seed(123)
parentTime <- seqistatd(biofam.seq)[, 1]
seqplot.rf(biofam.seq, diss=diss, k=12, sortv=parentTime,
   main="Sorted by parent time")
```

`seqplot.tentrop` *Plotting superposed transversal-entropy curves*

Description

Functions to plot, in a same frame, transversal-entropy curves by group or multiple curves.

Usage

```
seqplot.tentrop(seqdata, group, main=NULL,
    col=NULL, lty=NULL, lwd=3.5, ylim=NULL, xtlab=NULL,
    xtstep=NULL, tick.last=NULL, with.legend=TRUE, glabels=NULL,
    legend.pos="topright", horiz=FALSE, cex.legend=1, ...)
```

```
seqplot.tentrop.m(seqdata.list, main=NULL,
    col=NULL, lty=NULL, lwd=3.5, ylim=NULL, xtlab=NULL,
    xtstep=NULL, tick.last=NULL, with.legend=TRUE, glabels=NULL,
    legend.pos="topright", horiz=FALSE, cex.legend=1, ...)
```

Arguments

`seqdata`	a state sequence object (see `seqdef`).
`seqdata.list`	a list of state sequence objects.
`group`	a factor giving the group membership of each sequence in `seqdata`.
`main`	a character string giving the title of the graphic; if `NULL`, a default title is printed.
`col`	a vector of colors for the different curves.
`lty`	a vector of line types for the different curves. See `lines`.
`lwd`	numeric or vector of numerics: width of curve lines. See `lines`.
`ylim`	pair of numerics defining the range for the y-axis. If left `NULL`, the limits are defined from the data.
`xtlab`	vector of strings defining the x-axis tick labels.
`xtstep`	integer: step between tick marks on the x-axis. If unspecified, attribute `xtstep` of (first) `seqdata` is used.
`tick.last`	logical. Should a tick mark be enforced at the last position on the x-axis? If unspecified, attribute `tick.last` of (first) `seqdata` is used.
`glabels`	a vector of strings with the curve labels. If `NULL` curves are labeled with the levels of the group variable or, for `seqplot.tentrop.m`, as `seq1, seq2, ...`
`with.legend`	logical: Should the legend be plotted. Default is `TRUE`.
`legend.pos`	legend position: default is `"topright"`. See`legend` .
`horiz`	logical: Should the legend be displayed horizontally. Set as `FALSE` by default, i.e., legend is displayed vertically.
`cex.legend`	Scale factor for the legend.
`...`	additional plot parameters (see `par`).

Details

Use `seqplot.tentrop` to plot curves of transversal entropies by groups of a same set of sequences, e.g. professional careers by sex.

Use `seqplot.tentrop.m` to plot multiple curves of transversal entropies corresponding to different sets of sequences such as sequences describing cohabitational and sequences describing occupational trajectories.

See Also

`seqHtplot` for an alternative way of plotting the transversal entropies and `seqstatd` to get the values of the entropies.

Examples

```
## Using the biofam data which has sequences from
## ages 15 to 30 years in columns 10 to 25
data(biofam)
biofam <- biofam[1:200,]
biofam.seq <- seqdef(biofam[,10:25], xtlab=as.character(15:30), xtstep=3)

## Plotting transversal entropies by sex
seqplot.tentrop(biofam.seq, group=biofam$sex, legend.pos="bottomright")

## Plotting transversal entropies for women
## by father's social status
group <- biofam$cspfaj[biofam$sex=="woman"]
seqplot.tentrop(biofam.seq[biofam$sex=="woman",], group=group,
    main="Women, by father's social status", legend.pos="bottomright")
```

`seqpolyads` *Measuring the Degree of Within-Polyadic Similarities*

Description

The function computes measures of the degree of similarities within polyadic member sequences compared to randomly assigned polyadic member sequences.

Usage

```
seqpolyads(seqlist, a=1, method="HAM", ...,
    w=rep(1,ncol(combn(1:length(seqlist),2))),
    s=36963, T=1000, core=1, replace=TRUE, weighted=TRUE,
    with.missing=FALSE, rand.weight.type=1, role.weights=NULL,
    show.time=FALSE)
```

Arguments

`seqlist`	A list of *J>1* state sequence `stslist` objects. List of input sets (polyads) of polyadic sequences. The state sequence objects in the list must all have the same number *N* of sequences and the same alphabet. The state sequence objects should be created with `seqdef` and the list with `list`. E.g., `list(gen1.seq,gen2.seq,gen3.seq)`.
`a`	Integer, 1 or 2. Random generation mechanism. If 1 (default), draws from the observed set of sequences, and if 2, in addition random draws of states from each randomly drawn sequence. See reference below for detail.
`method`	String. Method for computing sequence distances. See `seqdist`. Additional arguments may be required depending on the method chosen.
`...`	Additional arguments passed to `seqdist`

`s`	Integer. Default 36963. Using the same seed number on the same computer guarantees the same results each time. Set `s=NULL` if you don't want to set a seed. The random generator can be chosen with `RNGkind`.
`w`	Integer vector. Default 1. The weights assigned to between-polyadic member sets in the weight matrix. For example, for dyadic sequences, no weight is necessary and the distance computation takes on the default of 1. For triadic sequences, there are three weights between the first and the second members, the first and the third members, and the second and the third members, in a row-wise order. See reference below.
`T`	Integer. Default 1,000. The number of randomized computations.
`core`	Integer. Default 1. Number of cores for the computation. When greater than 1, the procedure utilizes parallel processing.
`replace`	Logical. When `a=2`, should state sampling in each sequence be done with replacement? Default is `TRUE`. Ignored when `a=1`.
`weighted`	Logical. Should we account for the weights when present in the sequence objects? See details. Default is `TRUE`.
`with.missing`	Logical. Should the missing state be considered as a regular state? Default is `FALSE`.
`rand.weight.type`	Integer, 1 or 2. Ignored when `weighted=FALSE`. If 1 (default), weight of each randomized polyad is the average of original weights of its members. If 2, member weights are adjusted by dividing them by the sum of weights of all drawn members of the same type.
`role.weights`	`NULL` or vector of non-negative weights of same length as the list `seqlist`. Ignored when `weighted=FALSE`. If non null, role weights for determining the weights of the randomized polyads.
`show.time`	Logical. Should elapsed time be displayed? Default is `FALSE`.

Details

The function computes the polyadic distance of the observed polyads, i.e., the (weighted) mean of the pairwise distances between members of the polyad. In addition, the following statistics are computed:

The U statistic measures for each observed polyad by how much its polyadic distance differs from the mean polyadic distance of `T` randomized polyads. *U.tp* is the p-value for a two-tailed t-test of the U statistic.

The V statistic is, for each observed polyad, the proportion of `T` randomized polyads that have a greater polyadic distance. *V.95* is an associated dummy that takes value 1 when the proportion V is greater than 95% and 0 otherwise.

When the sequence objects in `seqlist` have weights and `weighted=TRUE`, the randomized sequences are sampled using the weights of the first element in the list. Each member of an observed polyad is supposed to have the same weight. This does not hold for the randomized polyads that are obtained by sampling their members independently. The weights of each randomized sequence is set as the average of the weights of its members. When role weights are provided with `role.weights`, a weighted average of the member weights is used. When `rand.weight.type=1`, original member weights are used. When `rand.weight.type=2`, the weights of randomly selected members are adjusted by the sum of weights of all randomly drawn members of the same type.

When `core` > `1`, the function uses the `doParallel` package for parallel computation.

Value

The function outputs a list of seven objects:

`mean.dist`	Vector of length 2 with the average observed and random within-polyadic distances.
`U`	Vector of N number of U statistics (see reference).
`U.tp`	Vector of N number of p-values for a two-tailed t-test of the U statistic.
`V`	Vector of N number of V statistics (see reference).
`V.95`	Vector of N number of 1s or 0s: 1 if a V value is at least 95 percent confident, 0 otherwise.
`observed.dist`	Vector of within-polyadic distances for the observed polyadic members.
`random.dist`	Vector of within-polyadic distances for the T number of randomly matched polyadic members.

Author(s)

Tim Liao and Gilbert Ritschard

References

Tim F. Liao (2021), "Using Sequence Analysis to Quantify How Strongly Life Courses Are Linked." *Sociological Science* **8**, 48-72, doi:10.15195/v8.a3.

Examples

```
data(polyads)
Gen <- polyads$Gen
seqGrandP <- seqdef(polyads[Gen=="1st Generation",2:11])
seqParent <- seqdef(polyads[Gen=="2nd Generation",2:11])
seqChild <- seqdef(polyads[Gen=="3rd Generation",2:11])
Seq <- rbind(seqGrandP,seqParent,seqChild)
slgth <- ncol(Seq)
colnames(Seq) <- 21:30
seqIplot(Seq,group=Gen,ylim=c(10,0),ylab="Triad",xlab="Age")
seqL <- list(seqGrandP,seqParent,seqChild)
core=1
seqG2.Tim <- seqpolyads(seqL[1:2],method="HAM",a=1,core=core,T=100)
seqG3.Tim <- seqpolyads(seqL,method="HAM",a=1,core=core,T=100)
seqG2.Dur <- seqpolyads(seqL[1:2],method="CHI2",step=slgth,core=core,T=100)
seqG3.Dur <- seqpolyads(seqL,method="CHI2",step=slgth,core=core,T=100)
```

`seqrep.grp` *Finding representative sets by group and their quality statistics.*

Description

This function determines representative sequences by group and returns the representatives by group and/or the quality statistics of the representative sets. The function is a wrapper for the TraMineR `seqrep` function.

Usage

```
seqrep.grp(seqdata, group = NULL, diss = NULL, ret = "stat",
    with.missing = FALSE, mdis, ...)
```

Arguments

`seqdata`	state sequence object as defined by `seqdef`.
`group`	group variable. If `NULL` a single group is assumed.
`diss`	dissimilarity matrix. If `NULL` the "LCS" dissimilarity matrix is computed.
`ret`	What should be returned? One of `"stat"` (default), `"rep"` or `"both"`.
`with.missing`	Logical. When `diss` = `NULL`. Are there missing values in the sequences? Default is `FALSE`.
`mdis`	Deprecated. Use `diss` instead.
`...`	additional arguments passed to `seqrep` .

Details

The function is a wrapper for running `seqrep` on the different groups defined by the `group` variable.

When `diss` = `NULL`, `seqdist` is used to compute the dissimilarities.

Value

If `ret="stat"`, a list with the quality statistics for the set of representatives of each group.

If `ret="rep"`, a list with the set of representatives of each group. Each element of the list is an object of class `stslist.rep` returned by `seqrep`.

If `ret="both"`, a list with the two previous outcomes.

Note

This function is a pre-release and further testing is still needed, please report any problems.

Author(s)

Gilbert Ritschard

See Also

seqrep

Examples

```
data(biofam)
biofam <- biofam[1:100,]
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
"Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.short <- c("P","L","M","LM","C","LC","LMC","D")
biofam.seq <- seqdef(biofam[,10:25], alphabet=0:7,
  states=biofam.short, labels=biofam.lab)
dist <- seqdist(biofam.seq, method="HAM")

seqrep.grp(biofam.seq, group=biofam$plingu02, diss=dist, coverage=.2, pradius=.1)
seqrep.grp(biofam.seq, group=biofam$plingu02, diss=dist, ret="rep", coverage=.2, pradius=.1)

## sequences with missing values
data(ex1)
sqex1 <- seqdef(ex1[,1:13])
nrow(ex1)
gp <- rep(1,7)
gp[5:7] <- 2
seqrep.grp(sqex1, group=gp, method="LCS", ret="rep",
  coverage=.2, pradius=.1, with.missing=TRUE)
```

---

seqsamm *Sequence Analysis Multistate Model (SAMM) procedure*

Description

Sequence Analysis Multistate Model (SAMM) procedure aims to simultaneously study the occurrence of transitions out of (an exit from) a spell in a given state along trajectories and the subsequence (or subtrajectory) immediately following it over a pre-defined period of time. This strategy allows including time-varying covariates in the sequence analysis framework.

Usage

```
seqsamm(seqdata, sublength, covar = NULL)
## S3 method for class 'SAMM'
plot(x, type="d", ...)
seqsammseq(samm, spell)
seqsammeha(samm, spell, typology, persper = TRUE)
```

Arguments

`seqdata`	State sequence object created with the `seqdef` function. Sequences representing any temporal process can be of different length.
`sublength`	Numeric. The length of the subsequence (or subtrajectory) following a transition to be considered.
`covar`	Optional `data.frame` storing covariates of interest. These covariates are added to the final data set and can be used in subsequent analyses.
`x`	A SAMM object produced by `seqsamm`
`samm`	A SAMM object produced by `seqsamm`.
`type`	the type of the plot `seqplot`. Default "d" for state distribution plots (chronograms).
`spell`	Character. The (ending) spell in a given spell to consider. It should be one of the states of the `alphabet` of the sequences. A spell is a series of time points in the same state.
`typology`	Factor or character. The typology of the trajectories out of the specified ending `spell` generated by a cluster analyses (see example). It should contain one observation per observed ending spell.
`persper`	Logical. If `TRUE`, the data are returned in person-period format. Otherwise, only one line per observed spell is returned.
`...`	additional plot parameters passed to `seqplot`.

Details

The Sequence Analysis Multistate Model (SAMM) procedure works in three steps. First, the substrings over a given time span `sublength` following any transition out of (exit from) a spell in a given state of the alphabet are extracted from the trajectories `seqdata`. This step is achieved using the `seqsamm` function. Each substring starts with the last time-point of the spell in the state. Second, these substrings are clustered using SA to identify typical substrings of medium-term changes. This is achieved separately for each ending spell (see `spell` argument). The `seqsammseq` function can be used to retrieve the sub-trajectories following each ending spell. Third, multistate models are used to estimate the chance (or risk) to end a spell in a given spell by distinguishing the type of trajectory that follows (and identified with cluster analysis). This allows estimating the effect of covariates on the chances to start each type of sub-sequence. The `seqsammeha` prepare the data to estimate the competing risk models for each ending spell. Then usual competing risks models can be used.

Generally speaking, the SAMM procedure allows studying the time spent in each state as well as the patterns of medium-term changes after an exit from that state appears along the trajectories. The example section below provides a step by step example of how to use it.

Value

A `SAMM` object (data.frame), storing the reorganized data in person period form. Column variables are:

`id`	Numeric. The ID of the observation as the row number in the original `seqdata`.

`time`	Numeric. The time unit of the current observation (from the beginning of the original sequence).
`begin`	Numeric. The time of the beginning of the current spell (from the beginning of the original sequence).
`spell.time`	Numeric. The time elapsed from the beginning of the current spell.
`transition`	Logical. Whether a transition out of the current spell occurred within this time unit.
`s.1` until `s.sublength`	The state sequence following the current observation starting from 1 (current state) until `sublength` time units after the current observation.
`lastobs`	Logical. Whether this is the last observation of the current spell, censored or not. This is useful when one wants only one row per individual, for instance to plot survival curves (see example).
`x`	object of class `SAMM` as produced by `seqsamm`
`Optional covariate list`	The covariates provided with the `covar` argument.

The function `seqsammseq` returns an `stslist` sequence object (see `seqdef`) of the trajectories following an ending spell.

The function `seqsammeha` returns a `data.frame` storing the person period data of a specific ending spell (see `spell` argument) considering the given `typology` as competing risks (see `typology` argument). Several variables are added to the `SAMM` objects (see above):

`SAMMtypology`	Factor. The events ending the specified spell using `"None"` when no event occurs.
`SAMM...`	Logical. A logical vector specifying whether the current observation ends the spell with the following `...` type of trajectory.

Author(s)

Matthias Studer

References

Studer, M., Struffolino, E., & Fasang, A. E. (2018). Estimating the Relationship between Time-varying Covariates and Trajectories: The Sequence Analysis Multistate Model Procedure. *Sociological Methodology*, 48(1), 103–135. doi:10.1177/0081175017747122

See Also

`seqcta`, `seqsha`

Examples

```
data(mvad)
mvad.seq <- seqdef(mvad, 17:86)

## For sake of simplicity we recode all "education" states to only one common state.
```

```
mvad.seq  <- seqrecode(mvad.seq, list("education"=c("FE", "HE", "school", "training")))
## We now have three states
seqdplot(mvad.seq)

###########################################################################
##  STEP I: Subsequence extraction
###########################################################################

## We start by extracting all subsequence of length 6
## We also add covariates from the mvad data frame
mvad.samm <- seqsamm(mvad.seq, 6, covar=mvad[, c("Grammar", "funemp", "gcse5eq")])
## Plotting the results to visualize the transitions out of each states.
plot(mvad.samm)
## Descriptive information on the seqsamm object
summary(mvad.samm)


###########################################################################
### STEP II: Typology of trajectory out of joblessness
###########################################################################
## We retrieve the subsequences following a transition out of a joblessness spell
jlseq <- seqsammseq(mvad.samm, "joblessness")


## Now we create a typology of these subsequences.

## Compute the clustering using LCS
jldist <- seqdist(jlseq, method="LCS")
## For sake of simplicity, use only 2 groups
library(cluster)
jlclust <- pam(jldist, diss=TRUE, k=2, cluster.only=TRUE)
## Specify the names of the types in the 2-cluster typology (here joblessness1 or joblessness2).
jltype <- paste0("joblessness", jlclust)


###########################################################################
### STEP III: Competing risks model of trajectories out of joblessness
###########################################################################

## Get the data to estimate competing risks models of the kind of trajectory
## out of jobjlessness
## We specify the SAMM object, the ending spell (joblessness) and our typology.
jleha <- seqsammeha(mvad.samm, "joblessness", jltype)

## Not run:
## Now jleha stores the data in person period format for competing risks
## Discrete time model using multinomial regression
## SAMMtypology and spell.time are variables created and stored in the jleha dataset
library(nnet)
multinom(SAMMtypology~spell.time+Grammar+funemp+gcse5eq, data=jleha)

## We can also have only one line per ending spell
## Plot the results
```

```
library(survival)
jleha <- seqsammeha(mvad.samm, "joblessness", jltype, persper=FALSE)
plot(survfit(Surv(spell.time, SAMMjoblessness1)~1, data=jleha))
## Cox model
summary(coxph(Surv(spell.time, SAMMjoblessness1)~gcse5eq+Grammar+funemp, data=jleha))
## Most of the time methods for recurrent events should be used.
## See for instance the coxme library to do so.

library(coxme)
summary(coxme(Surv(spell.time, SAMMjoblessness1)~gcse5eq+Grammar+funemp+(1|id), data=jleha))

## End(Not run)

############################################################################
### Now repeat steps II and III for employment and then education
### (Not shown here)
############################################################################
```

`seqsha` *Sequence History Analysis (SHA)*

Description

Sequence History Analysis (SHA) aims to study how a previous trajectory is linked to an upcoming event. This procedure relies on sequence analysis typologies to identify the type of previous trajectory as a time-varying covariate and uses discrete-time survival models to estimate its relationship with the upcoming event under consideration.

Usage

```
seqsha(seqdata, time, event, include.present = FALSE, align.end = FALSE, covar = NULL)
```

Arguments

`seqdata`	State sequence object created with the `seqdef` function. The whole trajectory followed by individuals.
`time`	Numeric. The time of occurrence of the event or the observation time for censored observations.
`event`	Logical. Whether the event occured or not (censored observations).
`include.present`	Logical. If `FALSE` (default), the state occurring at the current time is not included in the previous trajectory.
`align.end`	Logical. If `FALSE` (default), the previous trajectories are aligned at the beginning of the trajectory. If `TRUE`, the previous trajectories are aligned on the end.
`covar`	Optional `data.frame` storing covariates of interest. These covariates are added to the final data set.

Details

SHA works in four steps. First, it makes use of a discrete-time representation of the data also known as person-period file. In this format, one observation is generated for each individual at each time point. Second, the previous trajectory at each time point is coded as the sequence of states from the beginning (t=1 in our example) until the previous position t-1. Third, a typology of the previous trajectory is created using standard sequence analysis procedure. This results in a new time-varying covariate coding the type of previous trajectory at each time point. Fourth, the relationship between the previous trajectory and the subsequent event is estimated using a discrete-time model, which includes the past trajectory type as a covariate. In this step, other covariates can be included as well.

The `seqsha` function can be used to automatically reorganize the data according to the first two steps described above. Then, a standard procedure can be applied on the resulting data set. The example section below provides an example of the whole procedure.

Value

A data frame with the following variables:

`id`	Numeric. The ID of the observation as the row number in the original `seqdata`.
`time`	Numeric. The time unit from the beginning of the original sequence until the occurence of the event.
`event`	Logical. Whether the event occured within this time unit.
`T1` until `T...`	The state sequence coding the previous trajectory. Columns names depends on `seqdata` and `aligne.end`.
`Optional covariate list`	The covariates provided with the `covar` argument.

Author(s)

Matthias Studer

References

Rossignon F., Studer M., Gauthier JA., Le Goff JM. (2018). Sequence History Analysis (SHA): Estimating the Effect of Past Trajectories on an Upcoming Event. In: Ritschard G., Studer M. (eds) *Sequence Analysis and Related Approaches*. Life Course Research and Social Policies, vol 10. Springer: Cham. doi:10.1007/9783319954202_6

See Also

`seqcta`, `seqsamm`

Examples

```
## Create seq object for biofam data.
data(biofam)
## Reduce the biofam data to accelerate example
biofam <- biofam[100:300,]
```

```
bf.shortlab <- c("P","L","M","LM","C","LC", "LMC", "D")
bf.seq <- seqdef(biofam[,10:25], states=bf.shortlab)

## We focus on the occurrence the start of a LMC spell
## The code below aims to find when this event occurred (and whether it occurred).
bf.seq2 <- seqrecode(bf.seq, recodes=list(LMC="LMC"), otherwise = "Other")
dss <- seqdss(bf.seq2)
## Time until LMC spell
time <- ifelse(dss[, 1]=="LMC", 1, seqdur(bf.seq2)[, 1])
## Whether the event (start of LMC spell) started or not
event <- dss[, 1]=="LMC"|dss[, 2]=="LMC"

## The seqsha function will convert the data to person period.
## At each time point, the previous trajectory until that point is stored
sha <- seqsha(bf.seq, time, event, covar=biofam[, c("sex", "birthyr")])
summary(sha)

## Not run:
## Now we build a sequence object for the previous trajectory
previousTraj <- seqdef(sha[, 4:19])
seqdplot(previousTraj)
## Now we cluster the previous trajectories
##Compute distances using only the dss
## Ensure high sensitivity to ordering of the states
diss <- seqdist(seqdss(previousTraj), method="LCS")
##Clustering with pam
library(cluster)
pclust <- pam(diss, diss=TRUE, k=4, cluster.only=TRUE)
#Naming the clusters
sha$pclustname <- factor(paste("Type", pclust))
##Plotting the clusters to make senses of them.
seqdplot(previousTraj, sha$pclustname)


## Now we use a discrete time model include the type of previous trajectory as covariate.
summary(glm(event~time+pclustname+sex, data=sha, family=binomial))

## End(Not run)
```

`seqsplot` *Plot survival curves of the states in sequences*

Description

High level plot function for state sequence objects that produces survival curves of states in sequences. Usage is similar to the generic `seqplot` function of TraMineR, with a special handling of the `group` argument when `per.state=TRUE` is included in the `...` list.

Usage

```
seqsplot(seqdata, group = NULL, main = NULL, cpal = NULL,
  missing.color = NULL, ylab = NULL, yaxis = TRUE, axes = "all",
  xtlab = NULL, cex.axis = 1, with.legend = "auto", ltext = NULL,
  cex.legend = 1, use.layout = (!is.null(group) | with.legend != FALSE),
 legend.prop = NA, rows = NA, cols = NA, which.states, title, cex.plot, withlegend, ...)
```

Arguments

`seqdata`	State sequence object created with the `seqdef` function.
`group`	Grouping variable of length equal to the number of sequences. When `per.state = FALSE` (default), a distinct plot is generated for each level of `group`. When `per.state = TRUE`, the curves for each group level are drawn in a same plot for each distinct value of `alphabet(seqdata)`.
`main`	Character string. Title for the graphic. Default is `NULL`.
`cpal`	Vector. Color palette used for the states or the groups when `per.state=TRUE` is given along the `...` list. Default is `NULL`, in which case the `cpal` attribute of the `seqdata` sequence object is used (see `seqdef`) or the default colors assigned to groups when `type="s"` and `per.state=TRUE`. If user specified, a vector of colors of length and order corresponding to `alphabet(seqdata)` or, if for groups, the number of levels of the group variable.
`missing.color`	Color for representing missing values inside the sequences. By default, this color is taken from the `missing.color` attribute of `seqdata`.
`ylab`	Character string. an optional label for the y-axis. If set as `NA`, no label is drawn.
`yaxis`	Logical. Should the y-axis be plotted?
`axes`	Character string or logical. If set as `"all"` (default value) x-axes are drawn for each plot in the graphic. If set as `"bottom"` and `group` is used, axes are drawn only under the plots located at the bottom of the graphic area. If `FALSE`, no x-axis is drawn.
`xtlab`	Vector of length equal to the number of columns of `seqdata`. Optional labels for the x-axis tick labels. If unspecified, the column names of the `seqdata` sequence object are used (see `seqdef`).
`cex.axis`	Real. Axis annotation magnification. See `par`.
`with.legend`	Character string or logical. Defines if and where the legend of the state colors is plotted. The default value `"auto"` sets the position of the legend automatically. Other possible value is `"right"`. Obsolete value `TRUE` is equivalent to `"auto"`.
`ltext`	Vector of character strings of length and order corresponding to `alphabet(seqdata)` or, when for groups, to the levels of the group variable. Optional description for the color legend. If unspecified, the `label` attribute of the `seqdata` sequence object is used (see `seqdef`) or, when for groups, the levels of the group variable.
`cex.legend`	Real. Legend magnification. See `legend`.
`use.layout`	Logical. Should `layout` be used to arrange plots when using the group option or plotting a legend? When layout is activated, the standard '`par(mfrow=....)`' for arranging plots does not work. With `with.legend=FALSE` and `group=NULL`, layout is automatically deactivated and '`par(mfrow=....)`' can be used.

`legend.prop`	Real in range [0,1]. Proportion of the graphic area devoted to the legend plot when `use.layout=TRUE` and `with.legend=TRUE`. Default value is set according to the place (bottom or right of the graphic area) where the legend is plotted.
`rows,cols`	Integers. Number of rows and columns of the plot panel when `use.layout=TRUE`.
`which.states`	Vector of short state names. List of the states for which survival curves should be plotted.
`title`	Deprecated. Use `main` instead.
`cex.plot`	Deprecated. Use `cex.axis` instead.
`withlegend`	Deprecated. Use `with.legend` instead.
`...`	arguments to be passed to the function called to produce the appropriate statistics and the associated plot method (see details), or other graphical parameters. For example `per.spell` argument will typically be used for survival plots.

Details

This is a specific version of `seqplot` for type="s". It implements a dedicated handling of the group variable passed as `group` argument when `per.sate=TRUE` is included in the `...` list.

Invalid or non observed states are removed the list given as `which.states` argument. When `which.states = NULL`, `which.states` will be defined as the list of states present in the data.

When `per.sate=TRUE`, a distinct plot is generated for each state in the `which.states` list and, when a grouping variable is provided, the survival curves of all groups are plotted in each plot.

When `per.state=FALSE`, a distinct plot is generated for each group and the survival curves of all states listed as `which.states` are plotted in each plot.

Author(s)

Gilbert Ritschard (based on TraMineR seqplot function)

References

Gabadinho, A., G. Ritschard, N. S. Müller and M. Studer (2011). Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* **40**(4), 1-37.

See Also

`plot.stslist.surv`, `seqsurv`, `seqplot`,

Examples

```
## ======================================================
## Creating state sequence objects from example data sets
## ======================================================

data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
                "Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.short <- c("P","L","M","LM","C","LC","LMC","D")
```

```
sple <- 1:200  ## only the first 200 sequences
seqstatl(biofam[sple,10:25]) ## state 4 not present
biofam <- biofam[sple,]

biofam.seq <- seqdef(biofam[,10:25], alphabet=0:7, states=biofam.short, labels=biofam.lab)

## defining two birth cohorts
biofam$wwii <- factor(biofam$birthyr > 1945,
  labels=c("Born Before End of Word War II","Born After Word War II"))

## =============================
## Plots of state survival curves
## =============================

seqsplot(biofam.seq) ## all states, no group
seqsplot(biofam.seq, group=biofam$wwii, lwd=2) ## all states for each group
seqsplot(biofam.seq, group=biofam$wwii, per.state=TRUE, lwd=2) ## groups for each state

## For a selection of states only

seqsplot(biofam.seq, group=biofam$wwii, which.states= c('LM'), lwd=2)
## changing default color
seqsplot(biofam.seq, group=biofam$wwii, which.states= c('LM'),
  cpal="orange", lwd=2)
seqsplot(biofam.seq, group=biofam$wwii, which.states= c('LM','LMC'),
  cpal=c("orange","brown"), lwd=2)
seqsplot(biofam.seq, group=biofam$wwii, which.states= c('LM','LMC'), per.state=TRUE)
```

---

seqstart *Aligning sequence data on a new start time.*

Description

Changing the position alignment of a set of sequences.

Usage

```
seqstart(seqdata, data.start, new.start, tmin = NULL, tmax = NULL, missing = NA)
```

Arguments

`seqdata`	a data frame or matrix containing sequence data.
`data.start`	Integer. The actual starting date of the sequences. In case of sequence-dependent start dates, should be a vector of length equal to the number of rows of seqdata.
`new.start`	Integer. The new starting date. In case of sequence-dependent start dates, should be a vector of length equal to the number of rows of seqdata.
`tmin`	Integer. Start position on new position axis. If `NULL`, it is guessed from the data.
`tmax`	Integer. End position on new position axis. If `NULL`, it is guessed from the data.
`missing`	Character. Code used to fill missing data in the new time axis.

Value

A matrix.

Note

Warning: This function needs further testing.

Author(s)

Matthias Studer

Examples

```
#An example data set
paneldata <- matrix(c("A" ,"A" , "B" , "B" , "B",
"A" , "A" , "B" , "B" , "B",
"A" , "A",  "B" , "B" , "B" ,
"A"  ,"A" , "A" , "B"  ,"B" ,
"A"  ,"A" , "A" , "A" , "B"), byrow=TRUE, ncol=5)
colnames(paneldata) <- 2000:2004

print(paneldata)

## Assuming data are aligned on calendar years, starting in 2000
## Change from calendar date to age alignment
startyear <- 2000
birthyear <- 1995:1999
agedata <- seqstart(paneldata, data.start=startyear, new.start=birthyear)
colnames(agedata) <- 1:ncol(agedata)
print(agedata)

## Retaining only ages between 3 and 7 (4th and 8th year after birthyear).
seqstart(paneldata, data.start=startyear, new.start=birthyear, tmin=4, tmax=8, missing="*")

## Changing back from age to calendar time alignment
ageatstart <- startyear - birthyear
seqstart(agedata, data.start=1,  new.start=ageatstart)
## Same but dropping right columns filled with NA's
seqstart(agedata, data.start=1,  new.start=ageatstart, tmax=5)
```

`seqsurv` *Generate a survfit object for state survival times.*

Description

The function considers the spells in the different states in sequences and fits survival curves for each state. Alternatively, for a selected state, it fits the survival curves for each level of a stratifying group variable.

Survival curves are fitted with the `survfit` function.

Usage

```
seqsurv(seqdata, groups = NULL, per.state = FALSE, state = NULL,
    with.missing = FALSE)
```

Arguments

`seqdata`	A sequence `stslist` object as defined by the `seqdef` function.
`groups`	A stratifying group variable of length equal to the number of sequences.
`per.state`	Logical. Should the survival probabilites be computed for the state specified as `state` argument? If set as `TRUE`, the `state` argument must also be specified.
`state`	Single state value or a vector. The short name of the state for which to compute survival probabilities. If a vector of state names, survival probabilities are computed for the subset defined by those states. If `NULL`, survival probabilities are computed for all cases.
`with.missing`	Logical. Should the missing state be accounted for? (Not yet implemented!)

Details

The function considers the spells in the different states of a state sequence object (of class `stslist`).

When `per.state = FALSE`, it fits survival curves for each state in the alphabet. Currently, `per.state = FALSE` cannot be used with a non-`NULL` `groups` argument. However, `seqsplot` handles this case.

When `per.state = TRUE`, the survival curve is fitted only for the state provided as `state` argument. This is done for each level of the `groups` variable.

Survival curves are fitted with the `survfit` function.

Value

An object of class *stslist.surv*. There is a `plot` method for such objects.

Author(s)

Matthias Studer, Gilbert Ritschard, Pierre-Alexandre Fonta

See Also

`plot.stslist.surv` for basic plots of *stslist.surv* objects and `seqsplot` for more elaborated plots.

Examples

```
## Defining a sequence object with the data in columns 10 to 25
## (family status from age 15 to 30) in the biofam data set
data(biofam)
biofam.lab <- c("Parent", "Left", "Married", "Left+Marr",
                "Child", "Left+Child", "Left+Marr+Child", "Divorced")
biofam.short <- c("P","L","M","LM","C","LC","LMC","D")

sple <- 500:700  ## want a sample with all elements of the alphabet
##seqstatl(biofam[sple,10:25])
biofam <- biofam[sple,]

## creating the state sequence object
biofam.seq <- seqdef(biofam[,10:25], alphabet=0:7, states=biofam.short, labels=biofam.lab)

## Spell survival curves
(biofam.surv <- seqsurv(biofam.seq))

## Cohort distinguishing between those born before or after World War II
biofam$wwii <- biofam$birthyr <= 1945

## Separate survival curves in a given state (here LMC "Left+Marr+Child") according to wwii
(biofam.surv <- seqsurv(biofam.seq, groups=biofam$wwii, per.state=TRUE, state="LMC"))
plot(biofam.surv)
```

`seqtabstocc` *Frequencies of state co-occurrence patterns*

Description

Computes the frequencies of co-occurring state patterns.

Usage

```
seqtabstocc(seqdata, with.missing=FALSE, ...)
```

Arguments

`seqdata`	A state sequence (`stslist`) object as returned by `seqdef`.
`with.missing`	Logical. Should the missing state be considered as a regular state?
`...`	Additional arguments to be passed to `seqtab`.

Details

The function extracts the list of states co-occurring in each sequence. For each sequence, the co-occurring states are extracted as the sequence of the alphabetically sorted distinct states. The frequencies of the extracted sets of states is then obtained by means of the TraMineR `seqtab` function.

Returned patterns with a single state correspond to sequences that contain only that state.

Value

A `stslist.freq` object with co-occurrence patterns sorted in descending frequency order.

Author(s)

Gilbert Ritschard

See Also

`seqtab`

Examples

```
## Creating a sequence object from the first 500 actcal data.
data(actcal)
actcal.seq <- seqdef(actcal[1:500,13:24])

## 10 most frequent state patterns in the data
seqtabstocc(actcal.seq)

## All state patterns
seqtabstocc(actcal.seq, idxs=0)

## Example with missing states
data(ex1)
ex1 <- ex1[,1:13] ## dropping last weight column
## adding 3 sequences with no gap and left missing state
ex1 <- rbind(ex1,c(rep("A",4),rep(NA,9)))
ex1 <- rbind(ex1,c(rep("A",4),rep(NA,9)))
ex1 <- rbind(ex1,rep("A",13))
s.ex1 <- seqdef(ex1)
seqtabstocc(s.ex1, with.missing=TRUE)
```

`sortv` *Sort sequences by states at the successive positions*

Description

Returns a sorting vector to sort state sequences in a TraMineR sequence object (`seqdef`) by the states at the successive positions.

Usage

```
sorti(seqdata, start = "end", sort.index=TRUE)

sortv(seqdata, start = "end")
```

Arguments

`seqdata`	A state sequence object as returned by `seqdef`.
`start`	Where to start the sort. One of `"beg"` (beginning) or `"end"`.
`sort.index`	Should the function return sort indexes? If `FALSE`, sort values are returned.

Details

With `start = "end"` (default), the primary sort key is the final state, then the previous one and so on. With `start = "beg"`, the primary sort key is the state at the first position, then at the next one and so on.

With `sort.index = FALSE`, the function returns a vector of values whose order will determine the wanted order. This should be used as `sortv` argument of the `seqiplot` function. With `sort.index = TRUE`, the function returns a vector of indexes to be used for indexing.

The `sortv` form is an alias for `sorti(..., sort.index = FALSE)`.

Value

If `sort.index = FALSE`, the vector of sorting values.
Otherwise the vector of sorting indexes.

Author(s)

Gilbert Ritschard

See Also

Details about `type = "i"` or `type = "I"` in `seqplot` .

Examples

```
data(actcal)
actcal.seq <- seqdef(actcal[1:100,13:24])
par(mfrow=c(1,2))
seqIplot(actcal.seq, sortv=sortv(actcal.seq), with.legend = FALSE)
seqIplot(actcal.seq, sortv=sortv(actcal.seq, start="beg"), with.legend = FALSE)
actcal.seq[sorti(actcal.seq)[90:100],]

data(mvad)
mvad.seq <- seqdef(mvad[1:100,17:86])
par(mfrow=c(1,2))
seqIplot(mvad.seq, sortv=sortv(mvad.seq, start="end"), with.legend = FALSE)
seqIplot(mvad.seq, sortv=sortv(mvad.seq, start="beg"), with.legend = FALSE)
print( mvad.seq[sorti(mvad.seq,start="beg")[90:100],], format="SPS")
```

`toPersonPeriod` *Converting into person-period format.*

Description

Converts the STS sequences of a state sequence object into person-period format.

Usage

```
toPersonPeriod(seqdata)
```

Arguments

`seqdata`	A state sequence object as returned by `seqdef`.

Value

A data frame with three columns: `id`, `state` and `timestamp`.

Author(s)

Matthias Studer

See Also

`seqformat` .

Examples

```
data(mvad)
mvad.labels <- c("employment", "further education", "higher education",
    "joblessness", "school", "training")
mvad.scodes <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad.seq <- seqdef(mvad, 15:86, states = mvad.scodes, labels = mvad.labels)

mvad2 <- toPersonPeriod(mvad.seq[1:20,])
```

`TSE_to_STS` *Converting TSE data into STS (state sequences) format.*

Description

Conversion from TSE (time stamped event sequences) vertical format into STS (state sequences) data format.

Usage

```
TSE_to_STS(seqdata, id = 1, timestamp = 2, event = 3, stm = NULL, tmin = 1,
    tmax = NULL, firstState = "None")
```

Arguments

`seqdata`	a data frame or matrix with event sequence data in TSE format.
`id`	Name or index of the column containing the id's of the sequences.
`timestamp`	Name or index of the column containing the timestamps of the events.
`event`	Name or index of the column containing the events.
`stm`	An event to state transition matrix (See `seqe2stm`).
`tmin`	Integer. Starting time of the state sequence.
`tmax`	Integer. Ending time of the state sequence.
`firstState`	Character. The name of the state before any events has occurred.

Details

Convert TSE (time stamped event sequences) data into STS (state sequences) format. By default, the states are defined has the combination of events that already occurred. Different schemes may be specified using function `seqe2stm` and the `stm` argument.

Value

A `data.frame` with the sequences in STS format.

Note

This function is a pre-release and further testing is still needed, please report any problems.

Author(s)

Matthias Studer

References

Ritschard, G., Gabadinho, A., Studer, M. & Müller, N.S. (2009), "Converting between various sequence representations", In Ras, Z. & Dardzinska, A. (eds) *Advances in Data Management*. Series: *Studies in Computational Intelligence*. Volume 223, pp. 155-175. Berlin: Springer.

See Also

See Also `seqe2stm`, `seqformat`.

Examples

```
data(actcal.tse)
events <- c("PartTime", "NoActivity", "FullTime", "LowPartTime")
## Dropping all previous events.
stm <- seqe2stm(events, dropList=list(PartTime=events[-1],
   NoActivity=events[-2], FullTime=events[-3], LowPartTime=events[-4]))
mysts <- TSE_to_STS(actcal.tse[1:100,], id=1, timestamp=2, event=3,
   stm=stm, tmin=1, tmax=12, firstState="None")
```

Index