Package 'TraMineR'

November 3, 2009

Version 1.4-1

Date 2009-10-27

Title Sequences and trajectories mining for social scientists

Author Alexis Gabadinho <alexis.gabadinho@unige.ch>, Matthias Studer <matthias.studer@unige.ch>, Nicolas S. Muller <nicolas.muller@unige.ch>, Gilbert Ritschard <gilbert.ritschard@unige.ch>.

Maintainer Alexis Gabadinho <alexis.gabadinho@unige.ch>

Depends R (>= 2.7.1), RColorBrewer, boot

Suggests cluster

Description This package is a toolbox for sequence manipulation, description, rendering and more generally sequence data mining in the field of social sciences. Though it is primarily intended for analyzing state or event sequences that describe life courses such as family formation histories or professional careers its features apply indeed also to many other kinds of categorical sequence data. It accepts as input many different sequence representations and provides tools for translating sequences from one format to another. It offers several statistical functions for describing and rendering sequences, for computing distances between sequences with different metrics among which optimal matching, the longest common prefix and the longest common subsequence, and simple functions for extracting the most frequent subsequences and identifying the most discriminating ones among them. A user's guide can be found on TraMineR's web page.

License GPL (>= 2)

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

Repository CRAN

Date/Publication 2009-11-02 17:50:03

R topics documented:

`actcal` *Example data set: Activity calendar from the Swiss Household Panel*

Description

This data set contains individual monthly activity statuses from January to December 2000. It is a subsample of data collected by the Swiss Household Panel (SHP).

The state column (variable) names are ‘jan00’, ‘feb00’, etc...

There are four possible states:

A = Full-time paid job (> 37 hours)
B = Long part-time paid job (19-36 hours)
C = Short part-time paid job (1-18 hours)
D = Unemployed (no work)

The data set contains also the following covariates:

age00 (age in 2000)
educat00 (education level)
civsta00 (civil status)
nbadul00 (number of adults in household)
nbkid00 (number of children)
aoldki00 (age of oldest kid)
ayouki00 (age of youngest kid)
region00 (residence region)
com2.00 (residence commune type)
sex (sex of respondent)
birthy (birth year)

Usage

```
data(actcal)
```

Format

A data frame with 2000 rows, 12 state variables, 1 id variable and 11 covariates.

Source

Swiss Household Panel

References

`www.swisspanel.ch`

`actcal.tse` *Example data set: Activity calendar from the Swiss Household Panel (time stamped event format)*

Description

This data set contains events defined from the state sequences in the actcal data set. It was created with the code shown in the examples section. It is provided to symplify example of event sequence mining.

Usage

```
data(actcal.tse)
```

Format

Time stamped events derived from state sequences in the actcal data set.

Source

Swiss Household Panel

See Also

`seqformat`, `seqformat`

Examples

```
data(actcal)
actcal.seq <- seqdef(actcal[,13:24])

## Defining the transition matrix
transition <- seqetm(actcal.seq, method="transition")
transition[1,1:4] <- c("FullTime"         , "Decrease,PartTime",
     "Decrease,LowPartTime", "Stop")
transition[2,1:4] <- c("Increase,FullTime", "PartTime"         ,
     "Decrease,LowPartTime", "Stop")
transition[3,1:4] <- c("Increase,FullTime", "Increase,PartTime",
    "LowPartTime"         , "Stop")
transition[4,1:4] <- c("Start,FullTime"   , "Start,PartTime"   ,
    "Start,LowPartTime"   , "NoActivity")
transition

## Converting STS data to TSE
actcal.tse <- seqformat(actcal,var=13:24, from='STS',to='TSE',
        tevent=transition)

## Defining the event sequence object
actcal.seqe <- seqecreate(id=actcal.tse$id,
        time=actcal.tse$time, event=actcal.tse$event)
```

`alphabet` *Get or set the alphabet of a sequence object*

Description

This function gets or sets the (short) labels associated to the states in the alphabet of a sequence object (the list of all possible states, some of which states may not appear in the data).

Usage

```
alphabet(seqdata)
alphabet(seqdata) <- value
```

Arguments

`seqdata`	a state sequence object as defined with the `seqdef` function.
`value`	a character vector of the same length as the vector returned by the `alphabet` function, i.e. one label for each state in the alphabet.

Details

A state sequence object — created with the seqdef function — stores sequences as a matrix where columns are factors. The levels of the factors are made of the alphabet as well as the codes for missing value and void elements. The alphabet function retrieves or sets the "alphabet" attribute of the sequence object. The state names composing the alphabet are preferably short labels, since they are used for printing sequences. Longer labels for describing more precisely each state in legend are stored in the "labels" attribute of the sequence object.

Value

For 'alphabet' a character vector containing the alphabet.

For 'alphabet <-' the updated sequence object.

See Also

seqdef

Examples

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Retrieving the alphabet
alphabet(actcal.seq)

## Setting the alphabet
alphabet(actcal.seq) <- c("FT", "PT", "LT", "NO")
```

---

biofam *Example data set: Family life states from the Swiss Household Panel biographical survey*

Description

The *biofam* data set was constructed by Müller et al. (2007) from the data of the retrospective biographical survey carried out by the Swiss Household Panel (SHP) in 2002. The data set contains sequences of family life states from age 15 to 30 (sequence length is 16) and a series of covariates. The sequences are a sample of 2000 sequences of those created from the SHP biographical survey. It includes thus only individuals who were at least 30 years old at the time of the survey. The *biofam* data set describes thus family life courses of 2000 individuals born between 1909 and 1972.

The states numbered from 0 to 7 are defined from the combination of five basic states, namely Living with parents (Parent), Left home (Left), Married (Marr), Having Children (Child), Divorced:

0 = "Parent"
1 = "Left"

2 = "Married"
3 = "Left+Marr"
4 = "Child"
5 = "Left+Child"
6 = "Left+Marr+Child"
7 = "Divorced"

The covariates are:

sex
birthyr (birth year)
nat_1_02 (first nationality)
plingu02 (language of questionnaire)
p02r01 (religion)
p02r04 (religious participation)
cspfaj (father's social status)
cspmoj (mother's social status)

Two additional weights variables are inserted for illustrative purpose ONLY (since biofam is a subsample of the original data, these weights are not adapted):

wp00tbgp (weights inflating to the swiss population)
wp00tbgs (weights keeping sample size)

Usage

```
data(biofam)
```

Format

A data frame with 2000 rows, 16 state variables, 1 id variable and 7 covariates and 2 weights variables.

Source

Swiss Household Panel `www.swisspanel.ch`

References

Müller, N. S., M. Studer, G. Ritschard (2007). Classification de parcours de vie à l'aide de l'optimal matching. In *XIVe Rencontre de la Société francophone de classification (SFC 2007), Paris, 5 - 7 septembre 2007*, pp. 157–160.

`cpal`	*Get or set the color palette of a sequence object*

Description

This function gets or sets the color palette of a sequence object, that is, the list of colors used to represent the states.

Usage

```
cpal(seqdata)
cpal(seqdata) <- value
```

Arguments

`seqdata`	a state sequence object as defined by the `seqdef` function.
`value`	a vector containing the colors, of length equal to the number of states in the alphabet. The colors can be passed as character strings representing color names such as returned by the `colors` function, as hexadecimal values or as RGB vectors using the `rgb` function. Each color is attributed to the corresponding state in the alphabet, the order being the one returned by the `alphabet`.

Details

In the plot functions provided for visualizing sequence objects, a different color is associated to each state of the alphabet. The color palette is defined when creating the sequence object, either automatically using the `brewer.pal` function of the `RColorBrewer` package or by specifying a user defined color vector. The `cpal` function can be used to get or set the color palette of a previously defined sequence object.

Value

For 'cpal' a vector containing the colors.

For 'cpal<-' the updated sequence object.

See Also

`seqdef`

Examples

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
## The color palette is automatically set
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Retrieving the color palette
```

```
cpal(actcal.seq)
seqiplot(actcal.seq)

## Setting a user defined color palette
cpal(actcal.seq) <- c("blue","red", "green", "yellow")
seqiplot(actcal.seq)
```

`dissassoc` *Analysis of discrepancy based on dissimilarity measure*

Description

Compute the discrepancy (defined by a dissimilarity measure) explained by a categorical variable.

Usage

```
dissassoc(diss, group, R = 1000)
```

Arguments

`diss`	A dissimilarity matrix or a dist object (see `dist`)
`group`	The grouping variable
`R`	Number of permutations for computing the p-value. If equal to 1, no permutation test is performed.

Details

The association is based on a generalization of the ANOVA principle to any kind of distance metric. The test returns a pseudo R-squared that can be interpreted as a usual R-squared. The statistical significance of the association is computed by means of permutation tests. This function also performs a test of discrepancy homogeneity (equality of variance) using a generalization of the T statistic. There are `print` and `hist` methods (the latter producing an histogram of the significance values).

Value

Returns an object of class `dissassoc` with the following components:

`groups`	A data frame containing the number of cases and the discrepancy of each group
`anova.table`	The pseudo ANOVA table
`stat`	The value of the statistics and their p-values
`perms`	The permutation object, see `boot`

References

Studer, M., G. Ritschard, A. Gabadinho, and N. S. Müller (2009) Discrepancy analysis of complex objects using dissimilarities. In H. Briand, F. Guillet, G. Ritschard, and D. A. Zighed (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence. Berlin: Springer.

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009). Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7–18.

Batagelj, V. (1988) Generalized Ward and related clustering problems. In H. Bock (Ed.), *Classification and related methods of data analysis*, Amsterdam: North-Holland, pp. 67–74.

Anderson, M. J. (2001) A new method for non-parametric multivariate analysis of variance. *Austral Ecology* **26**, 32–46.

See Also

`dissvar` to compute the pseudo variance from dissimilarities and for a basic introduction to concepts of pseudo variance analysis.
`disstree` for an induction tree analyse of objects characterized by a dissimilarity matrix.
`disscenter` to compute the distance of each object to its group center from pairwise dissimilarities.
`dissmfac` to perform multi-factor analysis of variance from pairwise dissimilarities.

Examples

```
## Defining a state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])

## Building dissimilarities
mvad.lcs <- seqdist(mvad.seq, method="LCS")

## R=1 imply no permutation test
da <- dissassoc(mvad.lcs, group=mvad$gcse5eq, R=10)
print(da)
hist(da)
```

`disscenter` *Compute distance to the center of a group*

Description

Compute the dissimilarity between a set of objects and their group center using a pairwise dissimilarity matrix.

Usage

```
disscenter(diss, group=NULL, medoids.index=NULL, allcenter = FALSE)
```

Arguments

`diss`	a dissimilarity matrix such as generated by `seqdist`, or a `dist` object (see `dist`
`group`	if null, only one group is considered, otherwise group to compute center
`medoids.index`	if NULL, return dissimilarity to center. If equal to "first", return the index of the first encountered most central sequence. One index per group is returned. If equal to "all", all medoids index are returned. If `group` is set, one list per group is returned.
`allcenter`	logical. If `TRUE`, returns a `data.frame` containing the dissimilarity between each object and its group center, each column corresponding to a group.

Details

This function computes the dissimilarity between given objects and their group center. The group center may not belong to the space formed by the objects (in the same way, the average do not belong to a space formed by discrete measure). This distance can also be understood as the contribution to the discrepancy (see `dissvar`). The dissimilarity between a given object and its group center may be negative if the dissimilarity measure does not respect the triangle inequality.

It can be shown that this dissimilarity is equal to *Batagelj (1988)*:

$$d_{x\tilde{g}} = \frac{1}{n}\big(\sum_{i=1}^{n} d_{xi} - SS\big)$$

Where SS is the sum of squares (see `dissvar`).

Value

A vector with the dissimilarity to center of group for each sequence, or a list of medoid indexes.

References

Studer, M., G. Ritschard, A. Gabadinho, and N. S. Müller (2009) Discrepancy analysis of complex objects using dissimilarities. In H. Briand, F. Guillet, G. Ritschard, and D. A. Zighed (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence. Berlin: Springer.

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7–18.

Batagelj, V. (1988) Generalized ward and related clustering problems. In H. Bock (Ed.), *Classification and related methods of data analysis*, Amsterdam: North-Holland, pp. 67–74.

See Also

`dissvar` to compute the pseudo variance from dissimilarities and for a basic introduction to concepts of pseudo variance analysis
`dissassoc` to test association between objects represented by their dissimilarities and a covariate.

disstree for an induction tree analyse of objects characterized by a dissimilarity matrix.
dissmfac to perform multi-factor analysis of variance from pairwise dissimilarities.

Examples

```
## Defining a state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])

## Building dissimilarities
mvad.lcs <- seqdist(mvad.seq, method="LCS")

## Compute distance to center according to group gcse5eq
dc <- disscenter(mvad.lcs, group=mvad$gcse5eq)

## Ploting distribution of dissimilarity  to center
boxplot(dc~mvad$gcse5eq, col="cyan")

## Retrieving index of the first medoids, one per group
dc <- disscenter(mvad.lcs, group=mvad$Grammar, medoids.index="first")
print(dc)

## Retrieving index of all medoids in each group
dc <- disscenter(mvad.lcs, group=mvad$Grammar, medoids.index="all")
print(dc)
```

dissmfac *Multi-factor ANOVA from a dissimilarity matrix*

Description

Perform a multi-factor analysis of variance from a dissimilarity matrix.

Usage

```
dissmfac(formula, data, R = 1000, gower = FALSE, squared = TRUE,
 permutation = "dissmatrix")
```

Arguments

formula	A regression-like formula. The left hand side should be a dissimilarity matrix or a dist object.
data	data to search for variables in formula
R	Number of permutations to assess significance
gower	Logical: Is the dissimilarity matrix already a Gower matrix?
squared	Logical: should we square the dissimilarity matrix?
permutation	if equal to `dissmatrix`, permutations are done on the dissimilarity matrix, else if equal to "model" permutations are done on the variable matrix. Depending on the number of observation, "model" can be quicker.

Details

This method is, in some way, a generalization of `dissassoc` that can account for several explanatory variables. This function compute the part of variance explained by a list of covariates using a decomposition of the discrepancy (variance) explained. This function is slower than `dissassoc` for one factor. More on that, the latter also perform a test of discrepancy homogeneity (equality of variance) using a generalization of the T statistic.

The function is based on the program written for scipy (Python) by Ondrej Libiger and Matt Zapala. See Zapala and Schork (2006) for a full reference.

Value

A `dissmultifactor` object with the following components:

`mfac`	The part of variance explained by each variable (comparing full model to model without the specified variable) and its significance using permutation test
`call`	Function call
`perms`	Permutation values as a `boot` object
`perm_method`	Permutation method used to compute significance

References

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Discrepancy analysis of complex objects using dissimilarities. In H. Briand, F. Guillet, G. Ritschard, and D. A. Zighed (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence. Berlin: Springer.

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009). Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.

Anderson, M. J. (2001). A new method for non-parametric multivariate analysis of variance. *Austral Ecology* 26, 32-46.

McArdle, B. H. et M. J. Anderson (2001). Fitting multivariate models to community data: A comment on distance-based redundancy analysis. *Ecology* 82(1), 290-297.

Zapala, M. A. et N. J. Schork (2006). Multivariate regression analysis of distance matrices for testing associations between gene expression patterns and related variables. *Proceedings of the National Academy of Sciences of the United States of America* 103(51), 19430-19435.

See Also

`dissvar` to compute the pseudo variance from dissimilarities and for a basic introduction to concepts of pseudo variance analysis.
`dissassoc` to test association between objects represented by their dissimilarities and a covariate.
`disstree` for an induction tree analyse of objects characterized by a dissimilarity matrix.
`disscenter` to compute the distance of each object to its group center from pairwise dissimilarities.

Examples

```
## Defining a state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])

## Building dissimilarities
mvad.lcs <- seqdist(mvad.seq, method="LCS")
print(dissmfac(mvad.lcs ~ male + Grammar + funemp +
        gcse5eq + fmpr + livboth, data=mvad, R=10))
```

`dissrep` *Extracting sets of representative objects using a dissimilarity matrix*

Description

The function extracts a set of representative objects that exhibits the key features of the whole data set, the goal being to get easy sounded interpretation of the latter. The user can set either the desired coverage level (the proportion of objects having a representative in their neighborhood) or the desired number of representatives.

Usage

```
dissrep(dist.matrix, criterion="density",
        score=NULL, decreasing=TRUE,
        trep=0.25, nrep=NULL, tsim=0.1, dmax=NULL)
```

Arguments

`dist.matrix`	a matrix containing the pairwise distances between objects.
`criterion`	the representativeness criterion for sorting the candidate list. One of `"freq"` (frequency), `"density"` (neighborhood density) or `"dist"` (centrality). An optional vector containing the scores for sorting the candidate objects may also be provided. See below and details.
`score`	an optional vector containing the representativeness scores used for sorting the objects in the candidate list. The length of the vector must be equal to the number of rows/columns in the distance matrix, i.e the number of objects.
`decreasing`	if a score vector is provided, indicates wheter the objects in the candidate list must be sorted in ascending or decreasing order of this score. The first object in the candidate list is supposed to be the most representative.
`trep`	controls the size of the representative set by setting the desired coverage level, i.e the proportion of objects having a representative in their neighborhood. Neighborhood diameter is defined by `tsim`.
`nrep`	number of representatives. If `NULL` (default), `trep` argument is used to control the size of the representative set.
`tsim`	threshold for setting the redundancy and neighborhood diameter. Defined as a percentage of the maximum (theoretical) distance. Defaults to 0.1 (10%).

`dmax`	maximum theoretical distance. Redundancy and neighborhood diameters are defined as a proportion of this maximum theoretical distance. If `NULL`, it is derived from the distance matrix.

Details

The representative set is obtained by an heuristic that first builds a sorted list of candidates using a representativeness score and then eliminates redundancy. The available criterions for sorting the candidate list are: *sequence frequency*, *neighborhood density*, *centrality*. Other user defined sorting criterions can be provided using the `score` argument.

The *frequency* criterion uses the frequencies as representativeness score. The frequency of an object in the data is computed as the number of other objects with whom the dissimilarity is equal to 0. The more frequent an object the more representative it is supposed to be. Hence, objects are sorted in decreasing frequency order. Indeed, this criterion is the neighborhood (see below) criterion with the neighborhood diameter set to 0.

The *neighborhood density* criterion uses the number — the density — of objects in the neighborhood of each candidate. This requires indeed to set the neighborhood diameter. We suggest to set it as a given proportion of the maximal (theoretical) distance between two objects. Candidates are sorted in decreasing density order.

The *centrality* criterion uses the sum of distances to all other objects, i.e. the centrality as a representativeness criterion. The smallest the sum, the most representative the candidate.

For more details, see *Gabadinho et al., 2009*.

Value

An object of class `diss.rep`. This is a vector containing the indexes of the representative objects with the following additional attributes:

`Scores`	a vector with the representative score of each object given the chosen criterion.
`Distances`	a matrix with the distance of each object to its nearest representative.
`Statistics`	contains several quality measures for each representative in the set: number of objects attributed to the representative, number of object in the representatives neighborhood, mean distance to the representative.
`Quality`	overall quality measure.

Print and summary methods are available.

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Summarizing Sets of Categorical Sequences, In International Conference on Knowledge Discovery and Information Retrieval, Madeira, 6-8 October, INSTICC.

See Also

`seqrep`, `plot.stslist.rep`

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)

## Computing the distance matrix
costs <- seqsubm(biofam.seq, method="TRATE")
biofam.om <- seqdist(biofam.seq, method="OM", sm=costs)

## Representative set using the neighborhood density criterion
biofam.rep <- dissrep(biofam.om)
biofam.rep
summary(biofam.rep)
```

`disstree` *Dissimilarity Tree*

Description

Tree structured discrepancy analysis of non-measurable objects described by their pairwise dissimilarities.

Usage

```
disstree(formula, data= NULL, minSize = 0.05, maxdepth = 5,
   R = 1000, pval = 0.01)
```

Arguments

`formula`	A formula where the left hand side is a dissimilarity matrix and the right hand specifies the candidate partitioning variables to partition the population
`data`	a data frame where arguments in formula will be searched
`minSize`	minimum number of cases in a node, in percentage if less than 1.
`maxdepth`	maximum depth of the tree
`R`	Number of permutations used to assess the significance of the split.
`pval`	Maximum p-value, in percent

Details

The procedure iteratively splits the data. At each step, the procedure selects the variable and split that explains the biggest part of the discrepancy, i.e. the split for which we get the highest pseudo R2. The significance of the retained split is assessed through a permutation test.

Value

An object of class `disstree` that contains the following components:

`root`	A node object (see below), root of the tree
`adjustment`	A `dissassoc` object
`formula`	The formula used to generate the tree
`split`	Selected predictor, NULL for terminal nodes
`vardis`	Node discrepancy, see `dissvar`
`children`	Child nodes, NULL for terminal nodes
`ind`	Index of individuals in this node
`depth`	Depth of the node, starting from root node
`label`	Node label
`R2`	R squared of the split, NULL for terminal nodes

References

Studer, M., G. Ritschard, A. Gabadinho, and N. S. Müller (2009) Discrepancy analysis of complex objects using dissimilarities. In H. Briand, F. Guillet, G. Ritschard, and D. A. Zighed (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence. Berlin: Springer.

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.

Batagelj, V. (1988) Generalized ward and related clustering problems. In H. Bock (Ed.), *Classification and related methods of data analysis*, Amsterdam: Norht-Holland, pp. 67-74.

Anderson, M. J. (2001) A new method for non-parametric multivariate analysis of variance. *Austral Ecology* **26**, 32-46.

Piccarreta, R. et F. C. Billari (2007) Clustering work and family trajectories by using a divisive algorithm. *Journal of the Royal Statistical Society A* **170**(4), 1061–1078.

See Also

`seqtree2dot` to generate graphic representation of disstree objects when analyzing state sequences.
`disstree2dot` is a more general interface to generate such representation.
`dissvar` to compute discrepancy using dissimilarities and for a basic introduction to discrepancy analysis.
`dissassoc` to test association between objects represented by their dissimilarities and a covariate.
`dissmfac` to perform multi-factor analysis of variance from pairwise dissimilarities.
`disscenter` to compute the distance of each object to its group center from pairwise dissimilarities.

Examples

```
data(mvad)

## Defining a state sequence object
mvad.seq <- seqdef(mvad[, 17:86])

## Computing dissimilarities
mvad.lcs <- seqdist(mvad.seq, method="LCS")
dt <- disstree(mvad.lcs~ male + Grammar + funemp + gcse5eq + fmpr + livboth,
    data=mvad, R = 10)
print(dt)

## Using simplified interface to generate a file for GraphViz
seqtree2dot(dt, "mvadseqtree", seqdata=mvad.seq, type="d",
        border=NA, withlegend=FALSE, axes=FALSE, ylab="", yaxis=FALSE)

## Generating a file for GraphViz
disstree2dot(dt, "mvadtree", imagefunc=seqdplot, imagedata=mvad.seq,
        ## Additional parameters passed to seqdplot
        withlegend=FALSE, axes=FALSE, ylab="")

## Second method, using a specific function
myplotfunction <- function(individuals, seqs, mds,...) {
        par(font.sub=2, mar=c(3,0,6,0), mgp=c(0,0,0))

        ## using mds to order sequence in seqiplot
        mds <- cmdscale(seqdist(seqs[individuals,], method="LCS"),k=1)
        seqiplot(seqs[individuals,], sortv=mds,...)
        }

## Generating a file for GraphViz
## If imagedata is not set, index of individuals are sent to imagefunc
disstree2dot(dt, "mvadtree", imagefunc=myplotfunction, title.cex=3,
        ## additional parameters passed to myplotfunction
        seqs=mvad.seq, mds=mvad.mds,
        ## additional parameters passed to seqiplot (through myplotfunction)
        withlegend=FALSE, axes=FALSE,tlim=0,space=0, ylab="", border=NA)

## To run GraphViz (dot) from R and generate an "svg" file
## shell("dot -Tsvg -O mvadtree.dot")
```

`disstree2dot` *Graphical representation of a dissimilarity tree*

Description

Generate a "dot" file and associated images files that can be used in GraphViz to get a graphical representation of the tree.

Usage

```
disstree2dot(tree, filename, digits = 3,
  imagefunc = NULL, imagedata = NULL, imgLeafOnly = FALSE,
  devicefunc = "jpeg", imageext = "jpg", device.arg = list(),
  use.title = TRUE, label.loc = "main", node.loc = "main",
  split.loc = "sub", title.cex = 1, ...)
```

Arguments

`tree`	The tree to be plotted
`filename`	A filename, without extension, that will be used to generate image and dot files
`digits`	Number of significant digits to plot
`imagefunc`	A function to plot the individuals in a node, see details
`imagedata`	a `data.frame` that will be passed to imagefunc, see details
`imgLeafOnly`	Logical: If `TRUE`, only terminal node will be plotted
`devicefunc`	A device function used, typically jpeg
`imageext`	extension for image files.
`device.arg`	Argument passed to `devicefunc`
`use.title`	Logical: If `TRUE`, node information will be printed using `title` command, see details
`label.loc`	Location of the node label, see `title` for possible values
`node.loc`	Node content location, see `title` for possible values
`split.loc`	Split information location, see `title` for possible values
`title.cex`	`cex` applied to all title call (see `use.title`
`...`	other parameters that will be passed to `imagefunc`

Details

This function generates a "dot" file that can be used in GraphViz. It also generates one image per node through a call to `imagefunc` passing the selected lines of `imagedata` if present or a list of index (of individuals belonging to a node) if not.

if `use.title` is `TRUE`, `imagefunc` should take care to leave enough space for title informations.

This function is intended to be generic. See `seqtree2dot` for a much simpler version for states sequences objects.

Value

Nothing but generate a file in the current working directory (see `setwd`).

See Also

`disstree` for example

`disstreeleaf` *Terminal node appartenance*

Description

Return a factor with the terminal node appartenance of each cases.

Usage

```
disstreeleaf(tree)
```

Arguments

`tree`	The tree

See Also

`disstree` for examples

`dissvar` *Dissimilarity based discrepancy*

Description

Compute the discrepancy from the pairwise dissimilarities between objects. The discrepancy is a measure of dispersion of the set of objects.

Usage

```
dissvar(diss)
```

Arguments

`diss`	A dissimilarity matrix or a `dist` object (see `dist`)

Details

The discrepancy is an extension of the concept of variance to other kind of objects for which we have a dissimilarity measure. The discrepancy s^2 is defined as:

$$s^2 = \frac{1}{2n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}$$

Mathematical ground: In the Euclidean case, the sum of squares can be expressed as:

$$SS = \sum_{i=1}^{n} (y_i - \bar{y})^2 = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{n} (y_i - y_j)^2$$

The concept of discrepancy generalizes the equation by allowing to replace the term $(y_i - y_j)^2$ with any measure of dissimilarity d_{ij}.

Value

The pseudo variance.

References

Studer, M., G. Ritschard, A. Gabadinho, and N. S. Müller (2009) Discrepancy analysis of complex objects using dissimilarities. In H. Briand, F. Guillet, G. Ritschard, and D. A. Zighed (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence. Berlin: Springer.

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.

Batagelj, V. (1988) Generalized ward and related clustering problems. In H. Bock (Ed.), *Classification and related methods of data analysis*, Amsterdam: North-Holland, pp. 67-74.

Anderson, M. J. (2001) A new method for non-parametric multivariate analysis of variance. *Austral Ecology* **26**, 32-46.

See Also

`dissassoc` to test association between objects represented by their dissimilarities and a covariate.
`disstree` for an induction tree analyse of objects characterized by a dissimilarity matrix.
`disscenter` to compute the distance of each object to its group center from pairwise dissimilarities.
`dissmfac` to perform multi-factor analysis of variance from pairwise dissimilarities.

Examples

```
## Defining a state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])

## Building dissimilarities
mvad.lcs <- seqdist(mvad.seq, method="LCS")

## Pseudo variance of the sequences
print(dissvar(mvad.lcs))
```

`ex1` *Example data set with missing values and weights*

Description

Example data set used to demonstrate the handling of missing values and weights.

The state column (variable) names are '[P1]' ... '[P13]'

The alphabet is made of four possible states: A, B, C and D.

The data set contains also the 'weights' covariate which contains case weights. The sum of weights is 60.

Usage

```
data(ex1)
```

Format

A data frame with 6 rows, 13 state variables, 1 covariate.

Source

The brain of the TraMineR package maintainer.

---

famform — *Example data set: sequences of family formation*

Description

This data set contains 5 sequences of family formation histories, used by Elzinga to introduce several metrics for computing distances between sequences. These sequences don't contain information about the duration spent in each state, they contain only distinct successive states. This data set is used in TraMineR's manual to check some results obtained by comparing them with those presented by Elzinga.

Usage

```
data(famform)
```

Format

A data frame with 5 rows and 1 variable.

Details

the sequences are in the 'STS' format and stored in character strings where states are separated with '-'.

Source

Elzinga (2008)

References

Elzinga, Cees H. (2008). Sequence analysis: Metric representations of categorical time series. *Sociological Methods and Research*, forthcoming.

`mvad` *Example data set: Transition from school to work*

Description

The data comes from a study by McVicar and Anyadike-Danes on transition from school to work. The data consist of static background characteristics and a time series sequence of 72 monthly labour market activities for each of 712 individuals in a cohort survey. The individuals were followed up from July 1993 to June 1999.

States are:

employment (EM)
FE = further education (FE)
HE = higher education (HE)
joblessness (JL)
school (SC)
training (TR)

The data set contains also ids and sample weights as well as the following binary covariates:

male
catholic
Belfast, N.Eastern, Southern, S.Eastern, Western (location of school, one of five Education and Library Board areas in Northern Ireland)
Grammar (type of secondary education, 1=grammar school)
funemp (father's employment status at time of survey, 1=father unemployed)
gcse5eq (qualifications gained by the end of compulsory education, 1=5+ GCSEs at grades A-C, or equivalent)
fmpr (SOC code of father's current or most recent job, 1=SOC1 (professional, managerial or related))
livboth (living arrangements at time of first sweep of survey (June 1995), 1=living with both parents)

Usage

```
data(mvad)
```

Format

A data frame containing 712 rows, 72 state variables, 1 id variable and 13 covariates.

Source

McVicar and Anyadike-Danes (2002)

References

McVicar, Duncan and Anyadike-Danes, Michael (2002). Predicting Successful and Unsuccessful Transitions from School to Work by Using Sequence Methods, *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 165, 2, pp. 317–334.

`plot.stslist` *Plot method for state sequence objects*

Description

This is the plot method for state sequence objects of class *stslist* created by the seqdef function. It produces a sequence index plot.

Usage

```
## S3 method for class 'stslist':
plot(x, tlim=NULL, sortv=NULL,
        cpal=NULL, missing.color=NULL,
        ylab, yaxis=TRUE, xaxis=TRUE, xtlab=NULL, cex.plot=1, ...)
```

Arguments

`x`	a state sequence object created with the `seqdef` function.
`tlim`	indexes of the sequences to be plotted (default value is 1:10), for instance 20:50 to plot sequences 20 to 50, c(2,8,12,25) to plot sequences 2,8,12 and 25 in `seqdata`. If set to 0, all sequences in `seqdata` are plotted.
`sortv`	name of an optional variable used to sort the sequences before plotting.
`cpal`	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the 'seqdata' sequence object is used (see `seqdef`).
`missing.color`	alternative color for representing missing values inside the sequences. By default, this color is taken from the "missing.color" attribute of the sequence object being plotted.
`ylab`	An optional label for the y axis. If set to NA, no label is drawn.
`yaxis`	Controls whether the y axis is plotted or not. When set to `TRUE`, sequence indexes are displayed.
`xaxis`	if `TRUE` (default), the x (time) axis is plotted.
`xtlab`	optional labels for the x axis ticks labels. If unspecified, the column names of the 'seqdata' sequence object are used (see `seqdef`).

`cex.plot`	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 it.
`...`	arguments to be passed to the plot function or other graphical parameters.

Details

This is the default plot method for state sequence objects (produced by the `seqdef` function), i.e. objects of class *stslist*. It produces a sequence index plot, where individual sequences are rendered with stacked bars depicting the statuses over time.

This method is called by the generic `seqplot` function (if `type="i"`) that produces more sophisticated plots, allowing grouping and automatic display of the states legend. The `seqiplot` function is a shortcut for calling `seqplot` with `type="i"`.

The interest of sequence index plots has for instance been stressed by *Scherer (2001)*, *Brzinsky-Fay et al. (2006)* and *Gauthier (2007)*. Notice that such index plots for thousands of sequences result in very heavy graphic files if they are stored in PDF or POSTSCRIPT format. To reduce the size, we suggest saving the figures in bitmap format by using for instance `png` instead of `postscript` or `pdf`.

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)

## Plot of the 10 most frequent sequences
## with bar width proportional to the frequency
plot(biofam.seq)

## Plotting the all data set
## with no borders
plot(biofam.seq, tlim=0, space=0, border=NA)
```

`plot.stslist.freq` *Plot method for sequence frequency tables*

Description

Plot method for output produced by the seqmeant function, i.e objects of class *stslist.freq*.

Usage

```
## S3 method for class 'stslist.freq':
plot(x, cpal = NULL, missing.color = NULL, pbarw = TRUE,
  ylab = NULL, yaxis = TRUE, xaxis = TRUE,
  xtlab = NULL, cex.plot = 1, ...)
```

Arguments

`x`	an object of class `stslist.freq` as produced by the `seqtab` function.
`cpal`	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the `x` object is used.
`missing.color`	alternative color for representing missing values inside the sequences. By default, this color is taken from the `missing.color` attribute of the object being plotted.
`pbarw`	if pbarw=TRUE (default), the width of the bars are proportional to the sequence frequency in the dataset.
`ylab`	an optional label for the y axis. If set to NA, no label is drawn.
`yaxis`	if TRUE or "cum", the y axis is plotted with a label showing the cumulated percentage frequency of the displayed sequences. If "pct", the percentage value for each sequence is displayed.
`xaxis`	if TRUE (default) the xaxis is plotted.
`xtlab`	optional labels for the x axis ticks. If unspecified, the `names` attribute of the `x` object is used.
`cex.plot`	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.
`...`	further graphical parameters. For example `border=NA` to remove the bars borders, `space=0` to remove space between sequences. For more details about the graphical parameter arguments, see `barplot` and `par`.

Details

This is the plot method for the output produced by the `seqtab` function, i.e. objects of class *stslist.freq*. It produces a plot showing the sequences sorted bottom up according to their frequency in the data set.

This method is called by the generic `seqplot` function (if `type="f"`) that produces more sophisticated plots, allowing grouping and automatic display of the states legend. The `seqfplot` function is a shortcut for calling `seqplot` with `type="f"`.

Examples

```
## Loading the 'actcal' example data set
data(actcal)

## Defining a sequence object with data in columns 13 to 24
## (activity status from january to december 2000)
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal, 13:24, labels=actcal.lab)

## 10 most frequent sequences in the data
actcal.freq <- seqtab(actcal.seq, tlim=10)
```

```
## Plotting the object
plot(actcal.freq, main="Sequence frequencies - actcal data set")

## Plotting all the distinct sequences without borders
## and space between sequences
actcal.freq2 <- seqtab(actcal.seq, tlim=0)
plot(actcal.freq2, main="Sequence frequencies - actcal data set",
  border=NA, space=0)
```

`plot.stslist.meant` *Plot method for objects produced by the seqmeant function*

Description

This is the plot method for objects of class stslist.meant produced by the seqmeant function.

Usage

```
## S3 method for class 'stslist.meant':
plot(x, cpal = NULL, ylab = NULL, yaxis = TRUE, xaxis = TRUE,
        xtlab = NULL, cex.plot = 1, ylim = NULL, ...)
```

Arguments

`x`	an object of class `stslist.meant` as produced by the `seqmeant` function.
`cpal`	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the 'seqdata' sequence object is used (see `seqdef`).
`ylab`	an optional label for the y axis. If set to NA, no label is drawn.
`yaxis`	controls whether the y axis is plotted. Default to TRUE.
`xaxis`	if TRUE (default) the xaxis is plotted.
`xtlab`	optional labels for the x axis ticks. If unspecified, the `names` attribute of the `x` object is used.
`cex.plot`	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.
`ylim`	an optional vector setting the limits for the y axis. If `NULL` (default), limits are set to (0, max. sequence length).
`...`	further graphical parameters. For more details about the graphical parameter arguments, see `barplot` and `par`.

Details

This is the plot method for the output produced by the `seqmeant` function, i.e. objects of class *stslist.meant*. It produces a plot showing the mean times spent in each state of the alphabet.

This method is called by the generic `seqplot` function (if `type="mt"`) that produces more sophisticated plots, allowing grouping and automatic display of the states legend. The `seqmtplot` function is a shortcut for calling `seqplot` with `type="mt"`.

Examples

```
## Loading the mvad data set and creating a sequence object
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)

## Computing the mean times
mvad.meant <- seqmeant(mvad.seq)

## Plotting
plot(mvad.meant, main="Mean durations in each state of the alphabet")

## Changing the y axis limits
plot(mvad.meant, main="Mean durations in each state of the alphabet",
 ylim=c(0,40))
```

`plot.stslist.modst` *Plot method for modal state sequences*

Description

Plot method for output produced by the seqmodst function, i.e objects of class stslist.modst.

Usage

```
## S3 method for class 'stslist.modst':
plot(x, cpal = NULL, ylab = NULL, yaxis = TRUE, xaxis = TRUE,
  xtlab = NULL, cex.plot = 1, ...)
```

Arguments

`x`	an object of class `stslist.modst` as produced by the `seqmodst` function.
`cpal`	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the `x` object is used.
`ylab`	an optional label for the y axis. If set to NA, no label is drawn.
`yaxis`	if TRUE (default) the y axis is plotted.

`xaxis`	if TRUE (default) the x axis is plotted.
`xtlab`	optional labels for the x axis ticks. If unspecified, the `names` attribute of the `x` object is used.
`cex.plot`	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.
`...`	further graphical parameters. For more details about the graphical parameter arguments, see `barplot` and `par`.

Details

This is the plot method for the output produced by the `seqmodst` function, i.e. objects of class *stslist.modst*. It produces a plot showing the sequence of modal states with bar width proportional to the state frequencies.

This method is called by the generic `seqplot` function (if `type="ms"`) that produces more sophisticated plots, allowing grouping and automatic display of the states legend. The `seqmsplot` function is a shortcut for calling `seqplot` with `type="ms"`.

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)

## Modal state sequence
biofam.modst <- seqmodst(biofam.seq)
plot(biofam.modst)
```

`plot.stslist.rep` *Plot method for representative sequence sets*

Description

This is the plot method for output produced by the seqrep function, i.e objects of class *stslist.rep*. It produces a representative sequence plot.

Usage

```
## S3 method for class 'stslist.rep':
plot(x, cpal = NULL, pbarw = TRUE, dmax = NULL,
ylab = NULL, xaxis = TRUE, xtlab = NULL, cex.plot = 1, ...)
```

Arguments

`x`	an object of class *stslist.rep* as produced by the `seqrep` function.
`cpal`	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the `x` object is used.
`pbarw`	when `TRUE`, the bar heights are set proportional to the number of represented sequences.
`dmax`	maximal theoretical distance, used for the x axis limits.
`ylab`	an optional label for the y axis. If set to NA, no label is drawn.
`xaxis`	controls whether a x axis is plotted.
`xtlab`	optional labels for the x axis ticks labels. If unspecified, the column names of the object being plotted.
`cex.plot`	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.
`...`	further graphical parameters. For more details about the graphical parameter arguments, see `barplot` and `par`.

Details

This is the plot method for the output produced by the `seqrep` function, i.e. objects of class *stslist.rep*. It produces a plot where the representative sequences are displayed as horizontal bars with width proportional to the number of sequences assigned to them. Sequences are plotted bottom-up according to their representativeness score.
Above the plot, two parallel series of symbols associated to each representative are displayed horizontally on a scale ranging from 0 to the maximal theoretical distance D_{max}. The location of the symbol associated to the representative r_i indicates on axis A the (pseudo) variance (V_i) within the subset of sequences assigned to r_i and on the axis B the mean distance MD_i to the representative.

This method is called by the generic `seqplot` function (if `type="r"`) that produces more sophisticated plots with group splits and automatic display of the color legend. The `seqrplot` function is a shortcut for calling `seqplot` with `type="r"`.

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Summarizing Sets of Categorical Sequences, In *International Conference on Knowledge Discovery and Information Retrieval*, Madeira, 6-8 October, INSTICC.

Examples

```
## Loading the mvad data set and creating a sequence object
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)
```

```
## Computing optimal matching distances
submat   <- seqsubm(mvad.seq, method= "TRATE")
dist.om1 <- seqdist(mvad.seq, method="OM", indel=1, sm=submat)

## Extracting a representative set using the sequence frequency
## as a representativeness criterion
mvad.rep <- seqrep(mvad.seq, dist.matrix=dist.om1)

## Plotting the representative set
plot(mvad.rep)
```

`plot.stslist.statd` *Plot method for objects produced by the seqstatd function*

Description

This is the plot method for output produced by the `seqstatd` function, i.e objects of class *stslist.statd*.

Usage

```
## S3 method for class 'stslist.statd':
plot(x, type = "d", cpal = NULL, ylab = NULL,
  yaxis = TRUE, xaxis = TRUE, xtlab = NULL, cex.plot = 1, space=0, ...)
```

Arguments

`x`	an object of class `stslist.statd` as produced by the `seqstatd` function.
`type`	if `"d"` (default), a state distribution plot is produced. If `"Ht"` an entropy index plot is produced.
`cpal`	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of states in the alphabet. By default, the 'cpal' attribute of the `x` object is used.
`ylab`	an optional label for the y axis. If set to NA, no label is drawn.
`yaxis`	if TRUE or "cum", the y axis is plotted with a label showing the cumulated percentage frequency of the displayed sequences. If "pct", the percentage value for each sequence is displayed.
`xaxis`	if TRUE (default) the xaxis is plotted.
`xtlab`	optional labels for the x axis ticks. If unspecified, the `names` attribute of the input object is used.
`cex.plot`	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.
`space`	the space between the stacked bars. Default to 0, i.e. no space.
`...`	further graphical parameters. For example `border=NA` to remove the bars borders, `space=0` to remove space between sequences. For more details about the graphical parameter arguments, see `barplot` and `par`.

Details

This is the plot method for the output produced by the `seqstatd` function, i.e. objects of class *stslist.statd*. If `type="d"`it produces a state distribution plot presenting the sequence of the states frequencies for each time point, as computed by the `seqstatd` function. If `type="Ht"`, the series of state distribution entropies is plotted.

This method is called by the generic `seqplot` function (if `type="d"` or `type="Ht"`) that produces more sophisticated plots, allowing grouping and automatic display of the states legend. The `seqdplot` and `seqHtplot` functions are shortcuts for calling `seqplot` with `type="d"` or `type="Ht"` respectively.

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 distribution
biofam.statd <- seqstatd(biofam.seq)

## State distribution plot (default type="d" option)
plot(biofam.statd)

## Entropy index plot
plot(biofam.statd, type="Ht")
```

`plot.subseqelist` *Plot frequencies of subsequences*

Description

Plot frequencies of subsequences.

Usage

```
## S3 method for class 'subseqelist':
plot(x, freq=NULL,cex=1,...)
```

Arguments

`x`	The subsequences to plot (a `subseqelist` object
`freq`	The frequencies to plot, support if NULL
`cex`	Font size. See `par`.
`...`	arguments passed to `boxplot`

See Also

seqefsub

Examples

```
## loading data
data(actcal.tse)

## creating sequences
actcal.seqe <- seqecreate(actcal.tse)

## Looking for frequent subsequences
fsubseq <- seqefsub(actcal.seqe,pMinSupport=0.01)

## Frequence of first ten subsequences
plot(fsubseq[1:10], cex=2)
plot(fsubseq[1:10])
```

---

plot.subseqelistchisq

Plotting discriminant subsequences

Description

Plot the result of seqecmpgroup

Usage

```
## S3 method for class 'subseqelistchisq':
plot(x, ylim = "uniform", rows = NA, cols = NA,
         residlevels = c(0.05,0.01),
         cpal = brewer.pal(1 + 2 * length(residlevels), "RdBu"),
         legendcol = NULL, legend.cex = 1, ptype="freq", ...)
```

Arguments

x	The subsequences to plot (a subseqelist object).
ylim	if "uniform" all axis have same limits.
rows	Number of graphic rows
cols	Number of graphic columns
residlevels	Significance levels used to colorize the Pearson residual
cpal	Color palette used to color the results
legendcol	When TRUE the legend is printed vertically, when FALSE it is printed horizontally. If NULL (default) the best position will be chosen.
legend.cex	Scale parameters for text legend
ptype	If set to "resid", Pearson residuals are plotted instead of frequencies
...	Additional parameters passed to barplot

Value

nothing

See Also

`seqecmpgroup`

`read.tda.mdist` *Read a distance matrix produced by TDA.*

Description

This function reads a distance matrix produced by TDA into an R object. When computing OM distances in TDA, the output is a 'half' matrix stored in a text file as a vector.

Usage

```
read.tda.mdist(file)
```

Arguments

`file`	the path to the file containing TDA output.

Value

a R matrix containing the distances.

`seqcomp` *Compare two state sequences*

Description

Compare two state sequences and return TRUE if they are equal and FALSE otherwise

Usage

```
 seqcomp(x, y)
```

Arguments

`x`	a state sequence object containing a single sequence (typically the row of a main sequence object, see `seqdef`)
`y`	a state sequence object containing a single sequence (typically the row of a main sequence object, see `seqdef`)

Value

TRUE if sequences are identical, FALSE otherwise

See Also

`seqfind`, `seqfind`, `seqfind`

Examples

```
data(mvad)
mvad.shortlab <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad.seq <- seqdef(mvad, states=mvad.shortlab, 15:86)

## Comparing sequences 1 and 2 in mvad.seq
seqcomp(mvad.seq[1,],mvad.seq[2,])

## Comparing sequences 176 and 211 in mvad.seq
seqcomp(mvad.seq[176,],mvad.seq[211,])
```

`seqconc` *Concatenate vectors of states or events into a character string*

Description

Concatenate vectors of states or events into a character string. In the string, each state is separated by 'sep'. The void elements in the input sequences are eliminated.

Usage

```
seqconc(data, var=NULL, sep="-", vname="Sequence", void=NA)
```

Arguments

`data`	a dataframe or matrix containing sequence data.
`var`	the list of columns containing the sequences. Defaut to NULL, ie all the columns. Whether the sequences are in the compressed (character strings) or extended format is automatically detected by counting the number of columns.
`sep`	the character used as separator. By default, "-".
`vname`	an optional name for the variable containing the sequences. By default, "Sequence".
`void`	the code used for void elements appearing in the sequences (see *Gabadinho et al. (2008)* for more details on missing values and void elements in sequences). Default to NA.

Value

a vector of character strings, one for each row in the input data.

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2008). Mining Sequence Data in R with TraMineR: A user's guide. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

See Also

seqdecomp.

Examples

```
data(actcal)
actcal.string <- seqconc(actcal,13:24)
head(actcal.string)
```

---

seqdecomp *Convert a character string into a vector of states or events*

Description

For the moment, each character in the string will be considered to be one state or event = this function will not give accurate results if the character string representing the sequence contains events or states coded with more than one character.

Usage

```
seqdecomp(data, var=NULL, sep='-', miss="NA", vnames=NULL)
```

Arguments

`data`	a dataframe or matrix containing sequence data.
`var`	the list of columns containing the sequences. Defaut to NULL, ie all the columns. Whether the sequences are in the compressed (character strings) or extended format is automatically detected by counting the number of columns.
`sep`	the between states/events separator used in the input data set. Default to '-'.
`miss`	the symbol for missing values (if any) used in the input data set. Default to 'NA'.
`vnames`	optional names for the column/variables of the output data set. Default to NULL.

See Also

seqconc.

Examples

```
## Converts 'seq' into a vector of states of length 10
seq <- "A-A-A-A-B-B-B-C-C-C"
seqdecomp(seq)
```

`seqdef`	*Create a state sequence object*

Description

Create a state sequence object with attributes such as alphabet, color palette and state labels. Most TraMineR functions for state sequences require such a state sequence object as input argument. There are specific methods for plotting, summarizing and printing state sequence objects.

Usage

```
seqdef(data, var=NULL, informat="STS", stsep=NULL,
        alphabet=NULL, states=NULL, id=NULL, weights=NULL, start=1,
        left=NA, right="DEL", gaps=NA, missing=NA, void="%", nr="*",
        cnames=NULL, cpal=NULL, missing.color="darkgrey",
        labels=NULL, ...)
```

Arguments

`data`	a data frame or matrix containing sequence data.
`var`	the list of columns containing the sequences. Defaut to `NULL`, ie all the columns. Whether the sequences are in the compressed (successive states in a character string) or extended format is automatically detected.
`informat`	format of the original data. Default is 'STS'. Avalaible formats are: STS, SPS, SPELL. See TraMineR user's manual (*Gabadinho et al., 2008*) for a description of the formats.
`stsep`	the character used as separator in the original data if input format is successive states in a character string. If `NULL` (default value), the `seqfcheck` function is called for detecting automatically a separator among "-" and ":". Other separators must be specified explicitely.
`alphabet`	optional vector containing the alphabet (the list of all possible states). Use this option if some states in the alphabet don't appear in the data or if you want to reorder the states. The specified vector MUST contain AT LEAST all the states appearing in the data. It may possibly contain additional states not appearing in the data. If NULL, the alphabet is set to the distinct states appearing in the data as returned by the `seqstatl` function.
`states`	an optional vector containing the labels for the states. Must have a length equal to the number of states in the data, and the labels must be ordered accordingly with the values returned by the `seqstatl` function.
`id`	optional argument for setting the rownames of the sequence object. If NULL (default), the rownames are taken from the input data. If set to "auto", sequences are number 1 to number of sequences. A vector containing the rownames of length equal to number of sequences may be specified as well.
`weights`	optional numerical vector containing weights, which may be used by some functions to compute weighted statistics. EXPERIMENTAL.

`start`	starting time. For instance, if your sequences begin at age 15, you can specify 15. At this stage, used only for labelling column names.
`left`	the behavior for missing values appearing before the first (leftmost) valid state in each sequence. See *Gabadinho et al. (2008)* for more details on the options for handling missing values when defining sequence objects. By default, left missing values are treated as 'real' missing values and converted to the internal missing value code defined by the `nr` option. Other options are "DEL" to delete the positions containing missing values or a state code (belonging to the alphabet or not) to replace the missing values.
`right`	the behavior for missing values appearing after the last (rightmost) valid state in each sequence. Same options as for the `left` argument.
`gaps`	the behavior for missing values appearing inside the sequences, i.e. after the first (leftmost) valid state and before the last (rightmost) valid state of each sequence. Same options as for the `left` argument.
`missing`	the code used for missing values in the input data. When specified, all cells containing this value will be replaced by NA's, the internal R code for missing values. If 'missing' is not specified, cells containing NA's are considered to be missing values.
`void`	the internal code used by TraMineR for representing void elements in the sequences. Default is "`%`".
`nr`	the internal code used by TraMineR for representing real missing elements in the sequences. Default is "`*`".
`cnames`	optional names for the columns composing the sequence data. Those names will be used by default in the graphics as axis labels. If NULL (default), names are taken from the original column names in the data.
`cpal`	an optional color palette for representing the states in the graphics. If `NULL` (default), a color palette is created by calling the `brewer.pal` function of the `RColorBrewer` package. If number of states is less or equal than 8, the `"Accent"` palette is used. If number of states is between 8 and 12, the `"Set3"` palette is used. If the number of states in the data is greater than 12, you have to specify your own palette. The list of available colors is displayed by the `colors` function. You can also use alternatively some other palettes from the RColorBrewer package.
`missing.color`	alternative color for representing missing values inside the sequences. Defaults to "darkgrey".
`labels`	optional state labels used for the color legend of TraMineR's graphics. If `NULL` (default), the state names in the alphabet are used as state labels as well.
`...`	options passed to the `seqformat` function for handling input data that is not in STS format.

Details

Applying subscripts to sequence objects (eg. `seq[,1:5] or seq[1:10,]`) returns a state sequence object with some attributes preserved (alphabet, missing) and some others (start, column names) adapted to the selected column or row subset. If only one column is specified, a factor is returned.

Value

An object of class `stslist`. There are `print`, `plot` and `summary` methods for such objects. State sequence objects are required as argument to other functions such as plotting functions (seqdplot, seqiplot or seqfplot), functions to compute distances (seqdist), etc...

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2008). Mining Sequence Data in `R` with `TraMineR`: A user's guide. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

See Also

`plot.stslist` to plot state sequence objects, `seqplot` for high level plots of state sequence objects, `seqecreate` to create an event sequence object, `seqformat` for options to handle several longitudinal data formats.

Examples

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24,
        labels=c("> 37 hours", "19-36 hours", "1-18 hours", "no work"))

## Displaying the first 10 rows of the sequence object
actcal.seq[1:10,]

## Displaying the first 10 rows of the sequence object
## in SPS format
print(actcal.seq[1:10,], format="SPS")

## Plotting the first 10 sequences
plot(actcal.seq)

## Re-ordering the alphabet
actcal.seq <- seqdef(actcal,13:24,alphabet=c("B","A","D","C"))
alphabet(actcal.seq)

## Adding a state not appearing in the data to the
## alphabet
actcal.seq <- seqdef(actcal,13:24,alphabet=c("A","B","C","D","E"))
alphabet(actcal.seq)

## Adding a state not appearing in the data to the
## alphabet and changing the states labels
actcal.seq <- seqdef(actcal,13:24,
  alphabet=c("A","B","C","D","E"),
  states=c("FT","PT","LT","NO","TR"))
alphabet(actcal.seq)
actcal.seq[1:10,]
```

```
## ============================
## Example with missings values
## ============================
data(ex1)

## With right="DEL" default value
seqdef(ex1,1:13)

## Eliminating 'left' missing values
seqdef(ex1,1:13, left="DEL")

## Eliminating 'left' missing values and gaps
seqdef(ex1,1:13, left="DEL", gaps="DEL")

## ====================
## Example with weights
## ====================
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## weighted sequence frequencies
seqtab(ex1.seq)
```

`seqdiff` *Decompose the difference between groups of sequences*

Description

Decompose the difference between groups of sequences

Usage

```
seqdiff(seqdata, group, cmprange = c(0, 1),
        seqdist_arg=list(method="LCS",norm=TRUE))
```

Arguments

`seqdata`	The sequence to analyse
`group`	The group variable
`cmprange`	The range used to compare subsequences
`seqdist_arg`	argument passed directly to seqdist as a list

Details

Analyses at each timestamp the sequence discrepancy within a sliding time window (of range defined by `cmprange`) that is explained by the `group` variable. The method computes a distance matrix, using `seqdist` at each timestamp and then derives the explained discrepancy with `dissassoc`.

There are print and plot methods for the result returned.

Value

A `seqdiff` object, with the following items:

`stat`	A `data.frame` with three statistics (PseudoF, PseudoR2 and PseudoT) for each timestamp of the sequence, see `dissassoc`
`variance`	A `data.frame` with, at each time stamp, the discrepancy within each group defined by the `group` variable and for the whole population.

References

Studer, M., G. Ritschard, A. Gabadinho, and N. S. Müller (2009) Discrepancy analysis of complex objects using dissimilarities. In H. Briand, F. Guillet, G. Ritschard, and D. A. Zighed (Eds.), *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence. Berlin: Springer.

Studer, M., G. Ritschard, A. Gabadinho and N. S. Müller (2009) Analyse de dissimilarités par arbre d'induction. In EGC 2009, *Revue des Nouvelles Technologies de l'Information*, Vol. E-15, pp. 7-18.

See Also

`dissassoc` to analyse the association with the whole sequence

Examples

```
## Defining a state sequence object
data(mvad)
mvad.seq <- seqdef(mvad[, 17:86])

## Building dissimilarities
mvad.diff <- seqdiff(mvad.seq, group=mvad$gcse5eq)
print(mvad.diff)
plot(mvad.diff)
plot(mvad.diff, stat="Variance")
```

`seqdim` *Returns the dimension of a set of sequences*

Description

Returns the number of sequences (rows) and the maximum length of a set of sequences.

Usage

```
seqdim(seqdata)
```

Arguments

`seqdata`	a set of sequences.

Details

The function will first search for separators '-' or ':' in the sequences in order to detect wether they are in the compressed or extended format.

Value

a vector with the number of sequences and the maximum sequence length.

`seqdist` *Distances between sequences*

Description

Compute pairwise distances between sequences or distances to a reference sequence. Several metrics are available: optimal matching (OM) and other metrics such as the longest common prefix (LCP), the longest common suffix (RLCP), the longest common subsequence (LCS), the Hamming distance (HAM) and the Dynamic Hamming Distance (DHD).

Usage

```
seqdist(seqdata, method, refseq=NULL, norm=FALSE,
        indel=1, sm, with.miss = FALSE, full.matrix = TRUE)
```

Arguments

`seqdata`	a state sequence object defined with the `seqdef` function.
`method`	a character string indicating the metric to be used. One of "OM" (Optimal Matching), "LCP" (Longest Common Prefix), "RLCP" (reversed LCP, i.e. Longest Common Suffix), "LCS" (Longest Common Subsequence), "HAM" (Hamming distance), "DHD" (Dynamic Hamming distance).
`refseq`	Optional reference sequence to compute the distances from. Can be the index of a sequence in the state sequence object or 0 for the most frequent sequence, or an external sequence passed as a sequence object with 1 row.
`norm`	if TRUE, the computed OM, LCP, RLCP or LCS distances are normalized to account for differences in sequence lengths. Default is FALSE. See details
`indel`	the insertion/deletion cost (OM method). Default is 1. Ignored with non OM metrics.
`sm`	substitution-cost matrix (OM, HAM and DHD method). Default is NA. Ignored with LCP, RLCP and LCS metrics.
`with.miss`	must be set to TRUE when sequences contain non deleted gaps (missing values). See details.

`full.matrix`	If TRUE (default), the full distance matrix is returned. This is for compatibility with earlier versions of the `seqdist` function. If FALSE, an object of class `dist` is returned, that is, a vector containing only values from the upper triangle of the distance matrix. Since the distance matrix is symmetrical, no information is lost with this representation while size is divided by 2. Objects of class dist can be passed directly as arguments to most clustering functions. Ignored when refseq is set.

Details

The seqdist function returns a matrix of distances between sequences or a vector of distances to a reference sequence. The available metrics (see 'method' option) are optimal matching ("OM"), longest common prefix ("LCP"), longest common suffix ("RLCP"), longest common subsequence ("LCS"), Hamming distance ("HAM") and Dynamic Hamming Distance ("DHD"). The Hamming distance is OM without indels and the Dynamic Hamming Distance is HAM with specific substitution costs at each position as proposed by *Lesnard (2006).* Note that HAM and DHD apply only to sequences of equal length.

For OM, HAM and DHD, a user specified substitution cost matrix can be provided with the `sm` argument. For DHD, this should be a series of matrices grouped in a 3-dimensional matrix with the third index referring to the position in the sequence. When `sm` is not specified, a constant substitution cost of 1 used with HAM, and *Lesnard (2006)*'s proposal for DHD.

Distances can optionally be normalized by means of the `norm` argument. If set to TRUE, Elzinga's normalization (similarity divided by geometrical mean of the two sequence lengths) is applied to LCP, RLCP and LCS distances, while Abbott's normalization (distance divided by length of the longer sequence) is used for OM, HAM and DHD. For more details, see *Elzinga (2008)* and *Gabadinho et al. (2009).*

When sequences contain gaps and the `gaps=NA` option was passed to `seqdef`, i.e. when there are non deleted missing values, the `with.miss` argument should be set to TRUE. If left to FALSE the function stops when it encounters a gap. This is to make the user aware that there are gaps in his sequences. If "OM" method is selected, `seqdist` expects a substitution cost matrix with a row and a column entry for the missing state (symbol defined with the `nr` option of `seqdef`). This will be the case for substitution cost matrices returned by `seqsubm`. More details on how to compute distances with sequences containing gaps are given in *Gabadinho et al. (2009).*

Value

When refseq is specified, a vector with distances between the sequences in the data sequence object and the reference sequence is returned. When refseq is `NULL` (default), the whole matrix of pairwise distances between sequences is returned.

References

Elzinga, Cees H. (2008). Sequence analysis: Metric representations of categorical time series. *Sociological Methods and Research*, In revision.

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in `R` with `TraMineR`: A user's guide for version 1.1. Department of Econometrics and Laboratory of Demography, University of Geneva

Lesnard, L. (2006) Optimal Matching and Social Sciences. *Série des Documents de Travail du CREST*, Institut National de la Statistique et des Etudes Economiques, Paris.

See Also

seqsubm, seqdef.

Examples

```
## optimal matching distances with substitution cost matrix
## using transition rates
data(biofam)
biofam.seq <- seqdef(biofam, 10:25)
costs <- seqsubm(biofam.seq, method="TRATE")
biofam.om <- seqdist(biofam.seq, method="OM", indel=3, sm=costs)

## normalized LCP distances
biofam.lcp <- seqdist(biofam.seq, method="LCP", norm=TRUE)

## normalized LCS distances to the most frequent sequence in the data set
biofam.lcs <- seqdist(biofam.seq, method="LCS", refseq=0, norm=TRUE)

## histogram of the normalized LCS distances
hist(biofam.lcs)

## =====================
## Example with missings
## =====================
data(ex1)
ex1.seq <- seqdef(ex1,1:13)

subm <- seqsubm(ex1.seq, method="TRATE", with.miss=TRUE)
ex1.om <- seqdist(ex1.seq, method="OM", sm=subm, with.miss=TRUE)
```

seqdistmc *Multichannel distances between sequences*

Description

Compute multichannel pairwise distances between sequences. Several metrics are available: optimal matching (OM), the longest common subsequence (LCS), the Hamming distance (HAM) and the Dynamic Hamming Distance (DHD).

Usage

```
seqdistmc(channels, method, norm=FALSE, indel=1, sm=NULL,
         with.miss=FALSE, full.matrix=TRUE, link="sum", cval=2,
         miss.cost=2, cweight=NULL)
```

Arguments

`channels`	A list of state sequence objects defined with the `seqdef` function, each state sequence object corresponding to a "channel".
`method`	a character string indicating the metric to be used. One of "OM" (Optimal Matching), "LCS" (Longest Common Subsequence), "HAM" (Hamming distance), "DHD" (Dynamic Hamming distance).
`norm`	if TRUE, the computed distances are normalized to account for differences in sequence lengths. Default is FALSE. See details.
`indel`	A vector with an insertion/deletion cost for each channel (OM method).
`sm`	A list with a substitution-cost matrix for each channel (OM, HAM and DHD method) or a list of method names for generating the substitution-costs (see `seqsubm`).
`with.miss`	Must be set to TRUE when sequences contain non deleted gaps (missing values) or when channels are of different length. See details.
`full.matrix`	If TRUE (default), the full distance matrix is returned. If FALSE, an object of class `dist` is returned.
`link`	One of "sum" or "mean". Method to compute the "link" between channels. Default is to sum the substitution costs.
`cval`	Substitution cost for "CONSTANT" matrix, see `seqsubm`.
`miss.cost`	Missing values substitution cost, see `seqsubm`.
`cweight`	A vector of channel weights. Default is 1 (same weight for each channel).

Details

The `seqdistmc` function returns a matrix of multichannel distances between sequences. The available metrics (see 'method' option) are optimal matching ("OM"), longest common subsequence ("LCS"), Hamming distance ("HAM") and Dynamic Hamming Distance ("DHD"). See `seqdist` for more information about distances between sequences.

The `seqdistmc` function computes a multichannel distance in two steps following the strategy proposed by *Pollock (2007)*. First it builds a new sequence object derived from the combination of the sequences of each channel. Second, it derives the substitution cost matrix by summing (or averaging) the costs of substitution across channels. It then calls `seqdist` to compute the final matrix.

Normalization may be useful when dealing with sequences that are not all of the same length. For details on the applied normalization, see `seqdist`.

Value

A matrix of pairwise distances between sequences is returned.

References

Pollock, Gary (2007) Holistic trajectories: a study of combined employment, housing and family careers by using multiple-sequence analysis. *Journal of the Royal Statistical Society: Series A* **170**, Part 1, 167–183.

See Also

seqsubm, seqdef, seqdist.

Examples

```
data(biofam)

## Building one channel per type of event left, children or married
bf <- as.matrix(biofam[, 10:25])
children <-  bf==4 | bf==5 | bf==6
married <- bf == 2 | bf== 3 | bf==6
left <- bf==1 | bf==3 | bf==5 | bf==6

## Building sequence objects
child.seq <- seqdef(children)
marr.seq <- seqdef(married)
left.seq <- seqdef(left)

## Using transition rates to compute substitution costs on each channel
mcdist <- seqdistmc(channels=list(child.seq, marr.seq, left.seq),
        method="OM", sm =list("TRATE", "TRATE", "TRATE"))

## Using a weight of 2 for children channel and specifying substitution-cost
smatrix <- list()
smatrix[[1]] <- seqsubm(child.seq, method="CONSTANT")
smatrix[[2]] <- seqsubm(marr.seq, method="CONSTANT")
smatrix[[3]] <- seqsubm(left.seq, method="TRATE")
mcdist2 <- seqdistmc(channels=list(child.seq, marr.seq, left.seq),
        method="OM", sm =smatrix, cweight=c(2,1,1))
```

---

seqdss *Extract distinct states sequence from a sequence object*

Description

Extract distinct states sequence from a sequence object. Returns a sequence object containing the distinct states sequences, ie the durations are not taken into account. The DSS contained in 'D-D-D-D-A-A-A-A-A-A-A-D' is 'D-A-D'. Durations can be extracted with the 'seqdur' function.

Usage

```
seqdss(seqdata, with.miss=FALSE)
```

Arguments

`seqdata`	a sequence object as defined by the seqdef function.
`with.miss`	if set to TRUE, missing statuses (gaps in sequences) also appear in the DSS. See seqdef on options for handling missing values when creating sequence objects.

Value

a sequence object containing the distinct state sequence (DSS) for each sequence in the object given as argument.

See Also

seqdur.

Examples

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Retrieving the DSS
actcal.dss <- seqdss(actcal.seq)

## Displaying the DSS for the first 10 sequences
actcal.dss[1:10,]
```

---

seqdur *Extracts states durations from a sequence object.*

Description

Extracts states durations from a sequence object. Returns a matrix containing the states durations for the sequences. The states durations in 'D-D-D-D-A-A-A-A-A-A-A-D' are 4,7,1. Distinct states can be extracted with the seqdss function.

Usage

```
seqdur(seqdata, with.miss=FALSE)
```

Arguments

seqdata	a sequence object as defined by the seqdef function.
with.miss	if set to TRUE, durations are also computed for missing statuses (gaps in sequences). See seqdef on options for handling missing values when creating sequence objects.

Value

a matrix containing the states durations for each distinct state in each sequence.

See Also

seqdss.

Examples

```
## Creating a sequence object with the columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Retrieving the DSS
actcal.dur <- seqdur(actcal.seq)

## Displaying the durations for the first 10 sequences
actcal.dur[1:10,]
```

---

seqeapplysub *Checking if event sequences contain given subsequences*

Description

Checks occurrences of the subsequences `subseq` among the event sequences and returns the result according to the selected `method`.

Usage

```
seqeapplysub(subseq, method = "count", constraint = NULL, rules=FALSE)
```

Arguments

`subseq`	list of subsequences (an event subsequence object) such as created by `seqefsub`
`method`	type of result, should be one of "count", "presence" or "age"
`constraint`	Time constraints overriding those used to compute subseq. See `seqeconstraint`
`rules`	If set to TRUE, instead of checking occurences of the subsequences among the event sequences, check the occurence of the subsequences inside the subsequences (internally used by `seqerules`)

Details

There are three methods implemented: ’count’ counts the number of occurrence of each given subsequence in each event sequence; ’presence’ returns 1 if the subsequence is present, 0 otherwise; ’age’ returns the age of appearance of each subsequence in each event sequence. In case of multiple possibilities, the age of the first occurrence is returned. When the subsequence is not in the sequence, -1 is returned.

Value

The return value is a matrix where each row corresponds to a sequence (row names are set accordingly) and each column corresponds to a subsequence (col names are set accordingly). The cells of the matrix contain the requested values (count, presence-absence indicator or age).

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in `R` with `TraMineR`: A user's guide for version 1.1. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

See Also

`seqecreate` for more information on event sequence object and *Gabadinho et al. (2009)* on how to use the event sequence analysis module.

Examples

```
## Loading data
data(actcal.tse)

## Creating the event sequence object
actcal.seqe <- seqecreate(actcal.tse)

## Printing sequences
actcal.seqe[1:10]

## Looking for frequent subsequences
fsubseq <- seqefsub(actcal.seqe,pMinSupport=0.01)

## Counting the number of occurrences of each subsequence
msubcount <- seqeapplysub(fsubseq,method="count")
## First lines...
msubcount[1:10,1:10]
## Presence-absence of each subsequence
msubpres <- seqeapplysub(fsubseq,method="presence")
## First lines...
msubpres[1:10,1:10]

## Age at first appearance of each subsequence
msubage <- seqeapplysub(fsubseq,method="age")

## First lines...
msubage[1:10,1:10]
```

`seqecmpgroup` *Identifying discriminating subsequences*

Description

Identify and order the most discriminating subsequences according to a given statistical test.

Usage

```
seqecmpgroup(subseq, group, method="chisq", pvalue.limit=NULL)
```

Arguments

`subseq`	A `subseqelist` object (list of subsequences) such as produced by `seqefsub`
`group`	Variable or factor defining the membership to the groups to discriminate
`method`	The required test, one of `bonferroni` or `chisq`
`pvalue.limit`	Can be used to filter the results. Only subsequences with a p-value lower than the value set for this parameter will be selected. If NULL all subsequences are returned (regardless their p-values).

Details

The following test functions are implemented `chisq` Pearson Independence Chi squared test. `bonferroni` Pearson Independence Chi squared test with Bonferroni correction.

Value

An objet of type `subseqelistchisq` (subtype of `subseqelist`) with the following elements

`subseq`	Sorted list of found discriminating subsequences
`seqe`	The event sequence object on which the tests were computed
`constraint`	time constraints used for searching the subsequences (see `seqeconstraint`)
`labels`	levels (value labels) of the target group variable
`type`	Type of test used
`data`	A data frame with columns support, index (original order of the subsequence) and a pair of frequency and Pearson residual columns for each group

See Also

See Also `plot.subseqelistchisq` to plot the results

Examples

```
data(actcal.tse)
actcal.seqe <- seqecreate(actcal.tse)

##Searching for frequent subsequences, that is, appearing at least 20 times
fsubseq <- seqefsub(actcal.seqe, pMinSupport=0.01)

##searching for susbsequences discriminating the most men and women
data(actcal)
discr <- seqecmpgroup(fsubseq, group=actcal$sex, method="bonferroni")
##Printing discriminating subsequences
print(discr)
##Plotting the six most discriminating subsequences
plot(discr[1:6])
```

seqeconstraint *Setting time constraint*

Description

Function used to set time constraints in event sequence methods (seqe..) such as seqefsub for searching frequent subsequences or seqeapplysub for checking occurrences of subsequences.

Usage

```
seqeconstraint(maxGap = -1, windowSize = -1,
  ageMin = -1, ageMax = -1, ageMaxEnd = -1, countMethod = 1)
```

Arguments

`maxGap`	The maximum time gap between to events
`windowSize`	The maximum time span accepted for subsequences
`ageMin`	Minimal start time position allowed for subsequences. Ignored when equal to -1 (default).
`ageMax`	Maximal start time position allowed for subsequences. Ignored when equal to -1 (default).
`ageMaxEnd`	Maximal end time position allowed for subsequences. Ignored when equal to -1 (default).
`countMethod`	By default, subsequences are counted only one time by sequence. If set to 2, each occurence of the subsequence in a sequence is counted.

Details

`maxGap`, `windowSize`, `ageMin`, `ageMax` and `ageMaxEnd`. If so, two events should not be separated by more than `maxGap` and the whole subsequence should not exceed a `windowSize` time span. The other parameters specify the start and end age of the subsequence, it should start between `ageMin` and `ageMax` and finish before `ageMaxEnd`. Parameters `ageMin`, `ageMax` and `ageMaxEnd` are interpreted as the number of positions (time units) from the beginning of the sequence.

Value

A constraint object containing one item per constraint type.

See Also

seqefsub, seqeapplysub

seqecontain *Check if sequence contains events*

Description

Check if a sequence or a subsequence contains given events

Usage

```
seqecontain(seq, eventList, exclude = FALSE)
```

Arguments

`seq`	A event sequence object (`seqelist`) or a an event subsequence object (`subseqelist`)
`eventList`	A list of events
`exclude`	if TRUE the search is exclusive and returns FALSE for any subsequence containing an event that is not in `eventList`

Details

Checks, for each provided event sequence, if it contains one of the events in `eventList`. If `exclude` is TRUE, `seqecontain` looks if all events of the subsequence are in `eventList`.

Value

A logical vector.

See Also

`seqecreate` for creating event sequence objects and `seqefsub` for creating event subsequence objects.

Examples

```
data(actcal.tse)
actcal.seqe <- seqecreate(actcal.tse)

##Searching for frequent subsequences, that is appearing at least 20 times
fsubseq <- seqefsub(actcal.seqe,minSupport=20)

##looking for subsequence with FullTime
seqecontain(fsubseq,c("FullTime"))
```

`seqecreate` *Create event sequence objects.*

Description

Create an event sequence object from the given input.

Usage

```
seqecreate(data = NULL, id = NULL, timestamp = NULL, event = NULL,
  endEvent = NULL, tevent =  "transition", use.labels=TRUE)
```

Arguments

`data`	A state sequence object (see `seqdef`) or a data frame
`id`	Concerned sequence id's (integer), that is the 'id' column of the TSE format (ignored if data argument is provided).
`timestamp`	Time (double) at which events occur, that is the 'timestamp' column of the TSE format (ignored if data argument is provided).
`event`	Events that occurred at the specified time stamps, that is the 'event' column of the TSE format (ignored if data argument is provided).
`endEvent`	If specified this event will be considered as a flag for the end of observation time (total length of event sequences).
`tevent`	If `data` is a state sequence object either a transition matrix or a method to generate it (see `seqetm`)
`use.labels`	If TRUE, transitions names are built from state labels rather than from the sequence alphabet.

Details

There are several ways to create an event sequence object. The first one is by providing the events in TSE format (see `seqformat`), i.e. by providing three paired lists: id, timestamp and event, such that each triplet (id, timestamp, event) defines the event that occurs at time timestamp for case id. Several events at the same time for a same id are allowed. The lists can be provided with the arguments `id`, `timestamp` and `event`. An alternative is by providing a data frame as `data` argument in which case the function takes the required information from the "id", "timestamp" and "event" columns of that data frame.

The other way is to pass a state sequence object (as `data` argument) and to perfom an automatic state-to-event conversion. The simplest way to make a conversion is by means of a predefined method (see `seqetm`), such as "transition" (one distinct event per possible transition), "state" (one event when entering a new state) and "period" (a pair of events, one start-state event and one end-state event for each found transition). For a more customized conversion, you can specify a transition matrix in the same way as in `seqformat`. Function `seqetm` can help you in creating your transition matrix.

The resulting event sequence object can then be used in other 'seqe' methods, such as `seqefsub` or `seqeapplysub`.

See Also

seqformat for converting between sequence formats, seqefsub for searching frequent subsequences, seqecmpgroup to search for discriminant subsequences, seqeapplysub for counting subsequence occurrences and more, seqelength about length (observation time) of event sequences, seqdef to create a state sequence object.

Examples

```
##Starting with states sequences
##Loading data
data(biofam)
## Creating state sequences
biofam.seq <- seqdef(biofam,10:25,informat='STS')
## Creating event sequences from biofam
biofam.seqe <- seqecreate(biofam.seq)

## Loading data
data(actcal.tse)
## Creating sequences
actcal.seqe <- seqecreate(id=actcal.tse$id, timestamp=actcal.tse$time,
        event=actcal.tse$event)
##printing sequences
actcal.seqe[1:10]
## Using the data argument
actcal.seqe <- seqecreate(data=actcal.tse)
```

---

seqefsub *Searching for frequent subsequences*

Description

Returns the list of frequent subsequences satisfying the specified minimum support. Several time constraints can be set to restrict the search to specific time periods or subsequences durations.

Usage

```
seqefsub(seq, strsubseq = NULL, minSupport = NULL,
         pMinSupport = NULL, constraint = seqeconstraint(), maxK = -1)
```

Arguments

`seq`	A list of event sequences
`strsubseq`	Can be used to look for specific subsequences. See details.
`minSupport`	The minimum support (in number of sequences)
`pMinSupport`	The minimum support (in percentage, will be rounded)
`constraint`	Time constraint object, i.e the result of a call to seqeconstraint
`maxK`	The maximum number of events allowed in a subsequence

Details

There are two usages of this function. The first is for searching subsequences satisfying a support condition. The support is counted per sequence and not per occurrence, i.e. when a sequence contains twice a same subsequence it is counted only once. The support can be set through `pMinSupport` as a percentage (between 0 and 1 and it will be rounded), or through minSupport as a number of sequences. Time constraints can also be imposed with the `constraint` argument, which must be the outcome of a call to the `seqeconstraint` function).

The second possibility is for searching sequences that contain specified subsequences. This is done by passing the list of subsequences with the `strsubseq` argument. The subsequences must be formatted as the one used to display subsequences (see `str.seqelist`). Each group of events should be enclosed in parentheses () and separated with commas, and the succession of events should be denoted by a '-' that indicates a time gap. For instance "(FullTime)-(PartTime, Children)" stands for the subsequence "FullTime" followed by the group of the two simultaneously occurring events "PartTime" and "Children".

Information about the sequences that contain the subsequences can then be obtained with the `seqeapplysub` function.

Subsets of the returned `subseqelist` can be accessed with the [] operator (see example). There are print and plot methods for `subsequelist`.

Value

A `subseqelist` object which contain at least the following objects:

`seqe`	The list of sequences in which the subsequences were searched (a `seqelist` event sequence object).
`subseq`	A list of subsequences (a `seqelist` event sequence object).
`data`	A data frame containing details (support, frequency, ...) about the subsequences
`constraint`	The constraint object used when searching the subsequences.
`type`	The type of search: 'frequent' or 'user'

See Also

See `plot.subseqelist` to plot the result. See `seqecreate` for creating event sequences. See `seqeapplysub` to count the number of occurrences of frequent subsequences in each sequence. See `is.seqelist` about `seqelist`.

Examples

```
data(actcal.tse)
actcal.seqe <- seqecreate(actcal.tse)

##Searching for frequent subsequences, that is, appearing at least 20 times
fsubseq <- seqefsub(actcal.seqe, minSupport=20)
##The same using a percentage
fsubseq <- seqefsub(actcal.seqe, pMinSupport=0.01)
##Getting a string representation of subsequences
##Ten first subsequences
fsubseq[1:10]
```

```
##Using time constraints
##Looking for subsequence starting in summer (between june and september)
fsubseq <- seqefsub(actcal.seqe, minSupport=10,
  constraint=seqeconstraint(ageMin=6, ageMax=9))
fsubseq[1:10]

##Looking for subsequence contained in summer (between june and september)
fsubseq <- seqefsub(actcal.seqe, minSupport=10,
  constraint=seqeconstraint(ageMin=6, ageMax=9, ageMaxEnd=9))
fsubseq[1:10]

##Looking for subsequence enclosed in a 6 month period
## and with a maximum gap of 2 month
fsubseq <- seqefsub(actcal.seqe, minSupport=10,
  constraint=seqeconstraint(maxGap=2, windowSize=6))
fsubseq[1:10]
```

`seqeid` *Retrieve id of an event sequence object.*

Description

Retrieve id of an event sequence or a list of event sequence object.

Usage

```
seqeid(s)
```

Arguments

`s`	A sequence or a list of sequence

Examples

```
data(actcal.tse)
actcal.seqe <- seqecreate(actcal.tse)
seqeid(actcal.seqe)
```

`seqelength` *Length of event sequences*

Description

The length of an event sequence is its time span, i.e. the total time of observation. This information is optional but may be useful to perform for instance a survival analysis. `seqelength` retrieves the length the given sequences. `seqesetlength` sets the length of the sequences.

Usage

```
seqelength(s)
seqesetlength(s,len)
```

Arguments

`s`	An event sequence object (`seqelist`).
`len`	A list of sequence lengths.

Value

`seqelength` returns a numeric vector with the length of each sequences.

Examples

```
data(actcal.tse)
actcal.seqe <- seqecreate(actcal.tse)
##time to end is added
sl <- numeric()
sl[1:2000] <- 12
##All sequences with same length
seqesetlength(actcal.seqe, sl)
actcal.seqe[1:10]
##Retrieve length
seqelength(actcal.seqe)
```

`seqetm` *Create a transition-definition matrix*

Description

This function automatically creates a transition-definition matrix from a state sequence object to transform the state sequences into time stamped event sequences (in TSE format).

Usage

```
seqetm(seq, method = "transition", use.labels = TRUE,
  sep = ">", bp = "", ep = "end")
```

Arguments

`seq`	State sequence object from which transition events will be determined
`method`	The method to use. One of "transition", "period" or "state".
`use.labels`	If TRUE, transition names are built from state labels rather than from the alphabet.
`sep`	Separator to be used between the from-state and to-state that define the transition ("transition" method).
`bp`	Prefix for beginning of period event names ("period" method)
`ep`	Prefix for end of period event names ("period" method)

Details

One of three methods can be selected with the `method` argument:

'transition' generates a single (from-state > to-state) event for each found transition and a distinct start-state event for each different sequence start;

'period' generates a pair of events (end-state-event, start-state-event) for each found transition, a start-state event for the beginning of the sequence and an end-state event for the end of the sequence; names used for end-state and start-state names can be controlled with the `bp` and `ep` arguments;

'state' generates only the to-state event of each found transition (useful for analysing state sequences with methods for event sequences);

Value

The transition-definition matrix.

See Also

`seqformat` for converting to TSE format, `seqecreate` for creating an event sequence object, `seqdef` for creating a state sequence object.

Examples

```
## Creating a state sequence object from columns 13 to 24
## in the 'actcal' example data set
data(actcal)
actcal.seq <- seqdef(actcal,13:24,
        labels=c("FullTime", "PartTime", "LowPartTime", "NoWork"))
## Creating a transition matrix, one event per transition
seqetm(actcal.seq,method = "transition")

## Creating a transition matrix, single to-state events
seqetm(actcal.seq,method = "state")
```

```
## Creating a transition matrix, two events per transition
seqetm(actcal.seq,method = "period")

## changing the prefix of period start event.
seqetm(actcal.seq,method = "period", bp="begin")
```

`seqfind` *Find the occurrences of sequence(s) x in the set of sequences y*

Description

Finds the occurrences of sequence(s) x in the set of sequences y. The function returns the indexes of sequence x in the y sequence object.

Usage

```
 seqfind(x, y)
```

Arguments

`x`	a sequence object containing one or more sequences.
`y`	a sequence object.

Value

index(es) of the occurence of sequence(s) x in the set of sequences y.

See Also

.

Examples

```
data(mvad)
mvad.shortlab <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad.seq <- seqdef(mvad, states=mvad.shortlab, 15:86)

## Finding occurrences of sequence 176 in mvad.seq
seqfind(mvad.seq[176,],mvad.seq)

## Finding occurrences of sequence 1 to 8 in mvad.seq
seqfind(mvad.seq[1:8,],mvad.seq)
```

seqformat *Translation between sequence formats*

Description

Translate a sequence data set from one format to another.

Usage

```
seqformat(data, var=NULL, id=NULL,
        from, to, compressed=FALSE,
        nrep=NULL, tevent, stsep=NULL, covar=NULL,
        SPS.in=list(xfix="()", sdsep=","),
        SPS.out=list(xfix="()", sdsep=","),
        begin=NULL, end=NULL, status=NULL,
        process=TRUE, pdata=NULL, pvar=NULL,
        limit=100, overwrite=TRUE,
        fillblanks=NULL, tmin=NULL, tmax=NULL)
```

Arguments

`data`	a data frame or matrix containing sequence data.
`var`	the list of columns containing the sequences. Defaut to `NULL`, ie all the columns. Whether the sequences are in the compressed (character strings) or extended format is automatically detected by counting the number of columns.
`id`	column containing the identification numbers for the sequences. When using `SPELL` format as input, this identification number is mandatory, in order to identify all spells belonging to each individual in the data set.
`from`	format of the original data. Avalaible formats are: STS, SPS, SPELL. If `data` is a sequence object, format is automatically set to `STS`.
`to`	format of the output data. Avalaible formats are: STS, SPS, SRS, TSE
`compressed`	if TRUE and output format is one of STS, SPS or DSS, the output sequences are compressed into character strings
`nrep`	number of previous states replicated, for the 'SRS' format
`tevent`	when converting to time-stamped-event (TSE) format, a matrix of size 'ns' * 'ns' where 'ns' is the number of distinct states appearing in the sequences must be given. In this matrix, the cell a,b contains all events associated with a transition from state a to state b.
`stsep`	the character used as separator in the original data if input format is a vector of character strings. If `NULL` (default value), the `seqfcheck` function is called for detecting automatically a separator among "-" and ":". Other separators must be specified explicitely.
`covar`	the list of columns containing associated covariates to be included in the output data frame. If to='SRS' is choosed, the covariates are replicated accross each row. Default to NULL.

`SPS.in`	a list with the characters used as prefix/suffix and state/duration separator for each state duration couple if input data contains sequences in SPS format. Set the xfix element of the list to "" if there are no pre-suf-fix.
`SPS.out`	a list with the characters used as prefix/suffix and state/duration separator to be used for each state duration couple if output is in SPS format. Set the xfix element of the list to "" if there are no pre-suf-fix.
`begin`	when converting from SPELL, the column with the beginning of the spell
`end`	when converting from SPELL, the column with the end of the spell
`status`	when converting from SPELL, the column with the status
`process`	when converting from SPELL, create sequences on a process time axis. If set to false, create sequences on a calendar time axis.
`pdata`	when converting from SPELL and `process=TRUE`, either NULL, "auto" or the name of the data frame containing the individual 'birth' time, that is, the entering time from which the process time will be computed. If set to `NULL` (default), the starting and ending time of each spell are supposed to be ages. If set to auto, ages are computed using the starting time of the first spell of each individual as her/his birth date. If external birth dates are provided, the data must contain two columns: an id to match the birth time with SPELL data and a 'birth' time.
`pvar`	names or numbers of the columns containing the individual identification number and the 'birth' time in pdata.
`limit`	when converting from SPELL, size of the resulting dataframe when creating age sequences (by default goes from age 1 to age 100)
`overwrite`	when converting from SPELL, if overwrite is set to TRUE, the most recent episode overwrites the older one if they overlap each other. If set to false, the most recent episode starts from the end of the previous one.
`fillblanks`	when converting from SPELL, if fillblanks is not NULL, gaps between episodes are filled with any character given as argument.
`tmin`	when converting from SPELL, if sequences are to be defined on a calendar time axis is, defines the starting time of the axis. If set to NULL, the minimum time is taken from the 'begin' column in the data.
`tmax`	when converting from SPELL, if year sequences are wanted, defines the ending year of the dataframe. If set to NULL, it is guessed from the data (not very accurately).

Details

The 'seqformat' function is used to convert data from one format to another. The input data is first converted into the STS format and then converted to the output format. Depending on input and output formats, some information can be lost due to the steps in the conversion process. The output is a matrix, NOT a sequence object to be passed to TraMineR functions for plotting and mining sequences (use the `seqdef` function therefore). See *Gabadinho et al. (2009)* and *Ritschard et al. (2009)* for more details on longitudinal data formats and translation between them.

Value

a data frame

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Mining Sequence Data in R with TraMineR: A user's guide. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

Ritschard, G., A. Gabadinho, M. Studer and N. S. Müller. Converting between various sequence representations. in Ras, Z. & Dardzinska, A. (ed.) *Advances in Data Management*, Springer, 2009, 223, 155-175

See Also

seqdef

Examples

```
## Converting sequences into SPS format
data(actcal)
actcal.SPS.A <- seqformat(actcal,13:24, from="STS", to="SPS")
head(actcal.SPS.A)

## SPS (compressed) format with no prefix/suffix "/" as state/duration separator
actcal.SPS.B <- seqformat(actcal,13:24,
        from="STS", to="SPS", compressed=TRUE,
        SPS.out=list(xfix="", sdsep="/"))
head(actcal.SPS.B)

## Converting sequences into DSS (compressed) format
actcal.DSS <- seqformat(actcal,13:24,
        from="STS", to="DSS", compressed=TRUE)
head(actcal.DSS)
```

---

seqfpos *Search for the first occurrence of a given element in a sequence*

Description

Returns a vector containing the position of the first occurrence of the given element in each of the sequences in the data set.

Usage

```
seqfpos(seqdata, state)
```

Arguments

`seqdata`	a sequence object (see `seqdef` function).
`state`	the state element to search in the sequences

Details

the state to search for has to be passed as a character string, and must be one of the state returned by the `alphabet` function. If the state is not contained in a sequence, NA is returned for this sequence.

Examples

```
data(biofam)
biofam.seq <- seqdef(biofam,10:25)

## Searching for the first occurrence of state 1
## in the biofam data set.
seqfpos(biofam.seq,"1")
```

`seqgen` *Random sequences generation*

Description

Generates random sequences.

Usage

```
seqgen(n, length, alphabet, p)
```

Arguments

`n`	number of sequences to generate
`length`	sequences length
`alphabet`	the alphabet from which the sequences are generated
`p`	an optional vector of probabilities for the states in the alphabet. Must be of the same length as the alphabet. If not specified, equal probabilities are used.

Details

Each sequence is generated by choosing a set of random numbers (with min=1 and max=length of the alphabet) using the `runif` function. When the probability distribution is not specified, the uniform probability distribution giving same probability to each state is used to generate the sequences.

Value

a sequence object.

Examples

```
seq <- seqgen(1000,10,1:4,c(0.2,0.1,0.3,0.4))
seqstatd(seqdef(seq))
```

seqient *Within sequences entropy*

Description

Within sequences entropy

Usage

```
 seqient(seqdata, norm=TRUE, with.miss=FALSE)
```

Arguments

`seqdata`	a sequence object as returned by the the `seqdef` function.
`norm`	by default (TRUE), entropy is normalized, ie divided by the maximum entropy. The maximum entropy is computed as the entropy of the alphabet, ie an hypothetic sequence having all the states in the alphabet with equal length. Note that if for example the sequence length is uneven and the number of states in the alphabet is even, the theoretical maximum cannot be observed in the data.
`with.miss`	if set to TRUE, missing status (gaps in sequences) is handled as an additional state when computing the state distribution in the sequence.

Details

The seqient function returns the Shannon entropy of each sequence in `seqdata`. The entropy of a sequence is computed using the formula

$$h(\pi_1, \ldots, \pi_s) = -\sum_{i=1}^{s} \pi_i \log_2 \pi_i$$

where s is the size of the alphabet and π_i the proportion of occurrences of the ith state in the considered sequence. The entropy can be interpreted as the ‘uncertainty’ of predicting the states in a given sequence. If all states in the sequence are the same, the entropy is equal to 0. The maximum entropy for a sequence of length 12 with an alphabet of 4 states is 1.386294 and is attained when each of the four states appears 3 times.

Another measure of entropy is available: `seqstatd` returns the entropy of the distribution of states for each time unit.

Value

a vector whose number of elements is the number of sequences in `seqdata`, containing the entropy value of each sequence.

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Muller (2009). Mining Sequence Data in `R` with `TraMineR`: A user's guide. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

See Also

`seqstatd`.

Examples

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Summarize and plots an histogram
## of the within sequence entropy
actcal.ient <- seqient(actcal.seq)
summary(actcal.ient)
hist(actcal.ient)
```

`seqistatd` *States frequency for each individual sequence*

Description

Returns the state frequencies for each sequence in the sequence object.

Usage

```
 seqistatd(seqdata, with.miss=FALSE)
```

Arguments

`seqdata`	a sequence object (see `seqdef` function).
`with.miss`	if set to TRUE, cumulated durations are also computed for the missing status (gaps in the sequences). See seqdef on options for handling missing values when creating sequence objects.

Examples

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)
seqistatd(actcal.seq[1:10,])
```

seqlegend — *Plot a legend for the states in a sequence object*

Description

Plots a legend for the states in a sequence object. Useful if several graphics are plotted together and only one legend is necessary. Unless specified by the user, the *cpal* and *labels* attributes of the sequence object are used for the colors and text appearing in the legend (see `seqdef`).

Usage

```
seqlegend(seqdata, cpal=NULL, ltext=NULL,
        position="topleft", fontsize=1, ...)
```

Arguments

`seqdata`	a sequence object as returned by the the `seqdef` function.
`cpal`	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of distinct states. By default, the 'cpal' attribute of the 'seqdata' sequence object is used (see `seqdef`).
`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 number of distinct states. If unspecified, the 'labels' attributes of the 'seqdata' sequence object is used (see `seqdef`).
`position`	the position of the legend in the graphic area. For accepted values, see `legend`. Defaults to `"topleft"`.
`fontsize`	size of the font for the labels. A value less than 1 decreases the font size, a value greater than 1 increases the font size. Defaults to 1.
`...`	optional arguments passed to the `legend` function.

Examples

```
## Loading the 'actcal' example data set
## and defining a sequence object with
## (activity statuses from jan. to dec. 2000)
## the data in columns 13 to 24
data(actcal)
actcal.seq <- seqdef(actcal,13:24,
        labels=c("> 37 hours", "19-36 hours", "1-18 hours", "no work"))

## Plotting the sequences frequency,
## the states distribution
## and the legend
par(mfrow=c(2,2))
seqiplot(actcal.seq, tlim=0, withlegend=FALSE, border=NA, space=0)
seqfplot(actcal.seq, pbarw=TRUE, withlegend=FALSE)
seqdplot(actcal.seq, withlegend=FALSE)
seqlegend(actcal.seq)
```

seqlength *Sequence length*

Description

Returns the length of sequences.

Usage

```
seqlength(seqdata)
```

Arguments

seqdata	a sequence object created with the seqdef function.

Details

The length of a sequence is computed by eliminating the missing values at the end (right) and counting the number of states or events. The seqlength function returns a vector containing the length of each sequence in the sequence object given as argument.

Examples

```
## Loading the 'famform' example data set
data(famform)

## Defining a sequence object with the 'famform' data set
ff.seq <- seqdef(famform)

## Retrieving the length of the first 10 sequences
## in the ff.seq sequence object
seqlength(ff.seq)
```

seqLLCP *Compute the length of the longest common prefix of two sequences*

Description

Returns the length of the longest common prefix of two sequences. This attribute is described in *Elzinga (2008).*

Usage

```
seqLLCP(seq1, seq2)
```

Arguments

`seq1`	a sequence from a sequence object.
`seq2`	a sequence from a sequence object.

Value

an integer being the length of the longest common prefix of the two sequences.

References

Elzinga, Cees H. (2008). Sequence analysis: Metric representations of categorical time series. *Sociological Methods and Research*, forthcoming.

See Also

`seqdist`

Examples

```
data(famform)
famform.seq <- seqdef(famform)

## The LCP's length between sequences 1 and 2
## in the famform sequence object is 2
seqLLCP(famform.seq[1,],famform.seq[2,])
```

`seqLLCS` *Compute the length of the longest common subsequence of two sequences*

Description

Returns the length of the longest common subsequence of two sequences. This attribute is described in *Elzinga (2008).*

Usage

```
seqLLCS(seq1, seq2)
```

Arguments

`seq1`	a sequence from a sequence object
`seq2`	a sequence from a sequence object

Value

an integer being the length of the longest common subsequence of the two sequences.

References

Elzinga, Cees H. (2008). Sequence analysis: Metric representations of categorical time series. *Sociological Methods and Research*, forthcoming.

See Also

seqdist

Examples

```
LCS.ex <- c("S-U-S-M-S-U", "U-S-SC-MC", "S-U-M-S-SC-UC-MC")
LCS.ex <- seqdef(LCS.ex)
seqLLCS(LCS.ex[1,],LCS.ex[3,])
```

---

seqlogp *Computing the logarithm of sequences probabilities*

Description

Compute the logarithm of probability of each sequence using a state transition model. The probability of a sequence is equal to the product of each state probability of the sequence. There are several method to compute a state probability.

Usage

```
seqlogp(seqdata, prob="trate", time.varying=TRUE, begin="freq", weighted=TRUE)
```

Arguments

`seqdata`	The sequence to compute the probabilities.
`prob`	The name of the probability model used. The probability can be either based on transition rates ("trate") or on state frequencies ("freq"). This can also be an `array` specifying the transition probabilities at each t (see details).
`time.varying`	Logical. If `TRUE`, the probabilities are (either transition or frequencies) are computed separately for each time t
`begin`	Model used to compute the probability of the first state. Either "freq" to use the observed frequencies on the first period or a vector specifying the probability of each states appearing in `seqdata`.
`weighted`	Logical. If `TRUE`, uses the weights specified in `seqdata` when computing the observed transition rates.

Details

The sequence likelihood $P(s)$ is defined as the product of the probability with which each of its observed successive state is supposed to occur at its position. Let $s = s_1 s_2 \cdots s_\ell$ be a sequence of length ℓ. Then

$$P(s) = P(s_1, 1) \cdot P(s_2, 2) \cdots P(s_\ell, \ell)$$

with $P(s_t, t)$ the probability to observe state s_t at position t.

The question is how to determinate the state probabilities $P(s_t, t)$. Several methods are available and can be set using the `prob` argument.

One commonly used method for computing them is to postulate a Markov model, which can be of various order. We can consider probabilities derived from the first order Markov model, that is each $P(s_t, t), t > 1$ is set to the transition rate $p(s_t|s_{t-1})$. This is available in `seqlogp` by setting `prob="trate"`.
The transition rates may be considered constant over time/positions (`time.varying=FALSE`), that is estimated across sequences from the observations at positions t and $t-1$ for all t together. Time varying transition rates may also be considered (`time.varying=TRUE`), in which case they are computed separately for each position, that is estimated across sequences from the observations at positions t and $t-1$ for each t, yielding an array of transition matrices. The user may also specify his own transition rates array or matrix.

Another method is to use the frequency of a state at each position to set $P(s_t, t)$ (`prob="freq"`). In the latter case, the probability of a sequence is independant of the probability of its transition. Here again, the frequencies can be computed all together (`time.varying=FALSE`) or separately for each position t (`time.varying=TRUE`). For $t = 1$, we set $P(s_1, 1)$ to the observed frequency of the state s_1 at position 1. Alternatively, the `begin` argument allows to specify the probability of the first state.

The likelihood $P(s)$ being generally very small, `seqlogp` return $-\log P(s)$. The latter quantity is minimal when $P(s)$ is equal to 1.

Value

A vector containing the logarithm of each sequence probability.

Examples

```
## Creating the sequence objects using weigths
data(biofam)
biofam.seq <-  seqdef(biofam, 10:25, weights=biofam$wp00tbgs)

## Computing sequence probabilities
biofam.prob <- seqlogp(biofam.seq)
## Comparing the probability of each cohort
cohort <- biofam$birthyr>1940
boxplot(biofam.prob~cohort)
```

seqmeant — *Mean durations in each state*

Description

Compute the mean durations spent in each state of the alphabet for the set of sequences given as input.

Usage

```
seqmeant(seqdata, weighted = TRUE)
```

Arguments

`seqdata`	a sequence object as defined by the `seqdef` function.
`weighted`	if `TRUE`, the weights (weights attribute) attached to the sequence object are used for computing weighted mean durations.

Value

An object of class *stslist.meant*. There are `print` and `plot` methods for such objects.

See Also

`plot.stslist.meant` for basic plots of *stslist.meant* objects and `seqplot` with `type="mt"` argument for more sophisticated plots of the mean durations allowing grouping and legend.

Examples

```
## Defining a sequence object with columns 13 to 24
## in the actcal example data set
data(actcal)
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal,13:24,labels=actcal.lab)

## Computing the mean durations
seqmeant(actcal.seq)
```

seqmodst — *Sequence of modal states*

Description

Sequence made of the modal state at each position.

Usage

```
seqmodst(seqdata, dist = FALSE, ...)
```

Arguments

`seqdata`	a state sequence object as defined by the `seqdef` function.
`dist`	experimental
`...`	experimental

Details

In case of multiple modal states at a given position, the first one is taken. Hence, the result may vary with the alphabet order.

Value

an object of class *stslist.modst*. This is actually a state sequence object (containing a single state sequence) with additional attributes, among which the `Frequencies` attribute containing the transversal frequency of each state in the sequence. There are print and plot methods for such objects. More sophisticated plots can be produced with the `seqplot` function.

See Also

`plot.stslist.modst` for default plot method, `seqplot` for higher level 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.seq <- seqdef(biofam, 10:25, labels=biofam.lab)

## Modal state sequence
seqmodst(biofam.seq)
```

seqmpos — *Number of matching positions between two sequences.*

Description

Returns the number of common elements, ie same states appearing at the same position in the two sequences.

Usage

```
seqmpos(seq1, seq2, with.miss=FALSE)
```

Arguments

`seq1`	a sequence from a sequence object.
`seq2`	a sequence from a sequence object.
`with.miss`	if `TRUE`, gaps appearing at the same position in both sequences are also considedered as common elements

.

See Also

.

Examples

```
data(famform)
famform.seq <- seqdef(famform)

seqmpos(famform.seq[1,],famform.seq[2,])
seqmpos(famform.seq[2,],famform.seq[4,])

## Example with gaps in sequences
a <- c(NA,"A",NA,"B","C")
b <- c(NA,"C",NA,"B","C")

ex1.seq <- seqdef(rbind(a,b))

seqmpos(ex1.seq[1,], ex1.seq[2,])
seqmpos(ex1.seq[1,], ex1.seq[2,], with.miss=TRUE)
```

seqnum *Translate a sequence object's alphabet into numerical alphabet, ranging 0-(nbstates-1).*

Description

If the alphabet (the list of possible states or events in a set of sequences) is composed of characters, this function converts the sequence data using a numerical alphabet. The first state (for exemple 'A') is coded with the value '0', the second state (for exemple 'B') is coded with the value '1', etc... The function returns a sequence object containing the original sequences coded with the new numerical alphabet.

Usage

```
seqnum(seqdata, with.miss=FALSE)
```

Arguments

`seqdata`	a sequence object as defined by the `seqdef` function.
`with.miss`	if TRUE, missing elements in the sequences are turned into numerical values as well. The code for missing values in the sequences is retrieved as the 'nr' attribute of `seqdata`.

Examples

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## The first 10 sequences in the actcal.seq
## sequence object
actcal.seq[1:10,]
alphabet(actcal.seq)

## The first 10 sequences in the actcal.seq
## sequence object with numerical alphabet
seqnum(actcal.seq[1:10,])

## states A,B,C,D are now coded 0,1,2,3
alphabet(seqnum(actcal.seq))
```

seqplot	*Plot functions for state sequence objects*

Description

High level plot functions for state sequence objects that can produce state distribution, frequency, index, transversal entropy, sequence of modes, meant time, and representative plots.

Usage

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

seqdplot(seqdata, group=NULL, title=NULL, ...)
seqfplot(seqdata, group=NULL, title=NULL, ...)
seqiplot(seqdata, group=NULL, title=NULL, ...)
seqHtplot(seqdata, group=NULL, title=NULL, ...)
seqmsplot(seqdata, group=NULL, title=NULL, ...)
seqmtplot(seqdata, group=NULL, title=NULL, ...)
seqrplot(seqdata, group = NULL, title = NULL, ...)
```

Arguments

`seqdata`	a state sequence object created with the `seqdef` function.
`group`	Plots one plot for each level of the factor given as argument.
`type`	the type of the plot. Available types are `"d"` for state distribution plots, `"f"` for sequence frequency plots, `"Ht"` for entropy index plots, `"i"` for sequence index plots, `"ms"` for plotting the sequence of modal states, `"mt"` for mean times plots and `"r"` for representative sequence plots.
`title`	title for the graphic. Default to NULL.
`cpal`	alternative color palette to use for the states. If user specified, a vector of colors with number of elements equal to the number of distinct states. By default, the 'cpal' attribute of the 'seqdata' sequence object is used (see `seqdef`).
`missing.color`	alternative color for representing missing values inside the sequences. By default, this color is taken from the "missing.color" attribute of the sequence object being plotted.
`ylab`	an optional label for the y axis. If set to NA, no label is drawn.
`yaxis`	controls whether a y axis is plotted. If left to `'NULL'`, the value is set according to the plot type, i.e. `FALSE` for `type="i"` and `'TRUE'` for all other types. When set to `'TRUE'`, sequence indexes are displayed for `"i"`, mean time values for `"mt"` and percentages for `"d"` and `"f"`.

`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.
`xtlab`	optional labels for the x axis ticks labels. If unspecified, the column names of the 'seqdata' sequence object are used (see `seqdef`).
`cex.plot`	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.
`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 value is 'right'. Obsolete option 'TRUE' is identical 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' attributes 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 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. If layout is used, the standard 'par(mfrow=....)' for arranging plots will not work anymore. When `withlegend` is FALSE and `group` is NULL, layout is automatically deactivated and 'par(mfrow=....)' will work.
`legend.prop`	sets the 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,cols`	optional arguments to arrange plots when use.layout=TRUE.
`...`	arguments to be passed to the function called to produce the appropriate statistics and the associated plot method (see details), or other graphical parameters.

Details

`seqplot` is the generic function for high level plots of state sequence objects with group splits and automatic display of the color legend. Many different types of plots can be produced by means of the `type` argument. Except for sequence index plots, `seqplot` first calls the specific function producing the required statistics and then the plot method for objects produced by this function (see below). For sequence index plots, the state sequence object itself is plotted by calling the `plot.stslist` method. When splitting by groups and/or displaying the color legend, the `layout` function is used for arranging the plots.

The `seqdplot, seqfplot, seqiplot, seqHtplot, seqmsplot, seqmtplot` and `seqrplot` functions are aliases for calling `seqplot` with `type` argument set respectively to `"d", "f", "i", "Ht", "ms", "mt"` or `"r"`.

State distribution plot (`type="d"`) represent the sequence of the transversal state frequencies by position (time point) computed by the `seqstatd` function.

Sequence frequency plots (`type="f"`) display the most frequent sequences, each one with an horizontal stack bar of its successive states. Sequences are ordered bottom-up according to the relative frequencies computed by the `seqtab` function. The `plot.stslist.freq` plot method is

called for producing the plot.
The `tlim` optional argument may be specified for selecting the number of sequences to be plotted (default is 10, i.e. the ten most frequent sequences). The width of the bars representing the sequences is by default proportional to the sequences frequencies, but this can be disabled with the `pbarw=FALSE` optional argument. If weights have been specified when creating `seqdata`, weighted frequencies will be returned by `seqtab` since the default option is `weighted=TRUE`. See examples below, the `seqtab` and `plot.stslist.freq` manual pages for a complete list of optional arguments and *Müller et al., 2008)* for a description of sequence frequency plots.

In *sequence index plots* (`type="i"`), the requested individual sequences are rendered with horizontal stacked bars depicting the states over successive positions (time). Optional arguments are `tlim` for specifying the indexes of the sequences to be plotted (defaults to the first ten sequences, i.e `tlim=1:10`). For plotting nicely a (big) whole set use `tlim=0` together with additional graphical parameter `border=NA` and `space=0` to suppress bar borders and space between bars. The `sortv` argument can be used to pass a vector of numerical values for sorting the sequences. See `plot.stslist` for a complete list of optional arguments.

The interest of sequence index plots has for instance been stressed by *Scherer (2001)* and *Brzinsky-Fay et al. (2006)*. Notice that index plots for thousands of sequences result in very heavy PDF or POSTSCRIPT graphic files. Dramatic file size reduction may be achieved by saving the figures in bitmap format with using for instance the `png` graphic device instead of `postscript` or `pdf`.

The *entropy index plot* (`type="Ht"`) displays the evolution over positions of the transversal entropies (*Billari, 2001*). Transversal entropies are computed by calling `seqstatd` function and then plotted by calling the `plot.stslist.statd` plot method.

The *modal state sequence plot* (`type="ms"`) displays the sequence of the modal states with each mode proportional to its frequency at the given position. The `seqmodst` function is called which returns the sequence and the result is plotted by calling the `plot.stslist.modst` plot method.

The *mean time plot* (`type="mt"`) displays the mean time spent in each state of the alphabet as computed by the `link{seqmeant}` function. The `plot.stslist.meant` plot method is used to plot the resulting statistics.

The *representative sequence plot* (`type="r"`) displays a reduced, non redundant set of representative sequences extracted from the provided state sequence object and sorted according to a representativeness criterion. The `seqrep` function is called to extract the representative set which is then plotted by calling the `plot.stslist.rep` method. A distance matrix is required that is passed with the `dist.matrix` argument or by calling the `seqdist` function if `dist.matrix=NULL`. The `criterion` argument sets the representativeness criterion used to sort the sequences. See examples below, the `seqrep` and `plot.stslist.rep` manual pages for a complete list of optional arguments and *Gabadinho et al. (2009)* for more details on the extraction of representative sets.

References

Billari, F. C. (2001). The analysis of early life courses: Complex description of the transition to adulthood. *Journal of Population Research* **18**(2), 119-142.

Brzinsky-Fay C., U. Kohler, M. Luniak (2006). Sequence Analysis with Stata. *The Stata Journal*, **6**(4), 435-460.

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Summarizing Sets of Categorical Sequences. In *International Conference on Knowledge Discovery and Information Retrieval, Madeira, 6-8 October*. INSTICC.

Müller, N. S., A. Gabadinho, G. Ritschard and M. Studer (2008). Extracting knowledge from life courses: Clustering and visualization. In *Data Warehousing and Knowledge Discovery, 10th International Conference DaWaK 2008, Turin, Italy, September 2-5*, LNCS 5182, Berlin: Springer, 176-185.

Scherer S (2001). Early Career Patterns: A Comparison of Great Britain and West Germany. *European Sociological Review*, **17**(2), 119-144.

Examples

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

## 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)

## actcal data set
data(actcal)
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal,13:24,labels=actcal.lab)

## ex1 using weights
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## ========================
## Sequence frequency plots
## ========================

## Plot of the 10 most frequent sequences
seqplot(biofam.seq, type="f")

## Grouped by sex
seqfplot(actcal.seq, group=actcal$sex, tlim=10)

## Unweighted vs weighted frequencies
seqfplot(ex1.seq, weighted=FALSE)
seqfplot(ex1.seq, weighted=TRUE)

## =====================
## Modal states sequence
## =====================
seqplot(biofam.seq, type="ms")
## same as
seqmsplot(biofam.seq)

## ====================
## Representative plots
## ====================
## Computing a distance matrix
## with OM metric
costs <- seqsubm(biofam.seq, method="TRATE")
biofam.om <- seqdist(biofam.seq, method="OM", sm=costs)

## Plot of the representative sets grouped by sex
## using the default frequency criterion
seqrplot(biofam.seq, dist.matrix=biofam.om, group=biofam$sex)

## Plot of the representative sets grouped by sex
## using the default frequency criterion
seqrplot(biofam.seq, group=biofam$sex, dist.matrix=biofam.om)

## Plot of the representative sets grouped by sex
## using the "density" criterion
seqrplot(biofam.seq, group=biofam$sex, criterion="density", dist.matrix=biofam.om)

## ====================
## Sequence index plots
## ====================

## First ten sequences
seqiplot(biofam.seq)

## All sequences sorted by age in 2000
## grouped by sex
## using 'border=NA' and 'space=0' options to have a nicer plot
seqiplot(actcal.seq, group=actcal$sex, tlim=0, border=NA, space=0,
        sortv=actcal$age00)

## ===================
## Entropy index plots
## ===================
seqplot(biofam.seq, type="Ht", group=biofam$sex)

## =======================
## State distribution plot
## =======================

## Grouped by sex
seqplot(actcal.seq, type="d", group=actcal$sex)

## Sequence index plot (first 10 seq.)
## for the actcal data set
## grouped by sex
seqplot(actcal.seq, type="i", group=actcal$sex)

## ===============
## Meant time plot
## ===============

## actcal data set, grouped by sex
```

```
seqplot(actcal.seq, type="mt", group=actcal$sex)

## biofam data set, grouped by sex
seqmtplot(biofam.seq, group=biofam$sex)
```

`seqpm` *Find patterns in sequences*

Description

Search for a pattern (subsequence) into sequences.

Usage

```
seqpm(seqdata, pattern)
```

Arguments

`seqdata`	a sequence object as defined by the `seqdef` function.
`pattern`	a character string representing the pattern (subsequence) to search for, without sperator between the states.

Details

This function search a pattern (a character string) into a set of sequences and returns a list containing the results. The elements of the list are 'Nbmatch', containing the number of occurences of pattern and 'MatchesIndex', containing the indexes (row numbers) of the sequences that match the pattern (see exemples below).

Value

a list with two elements (see details).

Examples

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## search for pattern "DAAD"
## (no work-full time work-full time work-no work)
## results are stored in the 'daad' object
daad <- seqpm(actcal.seq,"DAAD")

## Looking at the sequences
## containing the pattern
actcal.seq[daad$MIndex,]

## search for pattern "AD"
## (full time work-no work)
seqpm(actcal.seq,"AD")
```

`seqrep` *Extracting sets of representative sequences*

Description

The function attempts to find an optimal set of representative sequences that exhibits the key features of the whole sequence data set, the goal being to get easy sounded interpretation of the latter.

Usage

```
seqrep(seqdata, criterion="density", score=NULL, decreasing=TRUE,
        trep=0.25, nrep=NULL,
        tsim=0.1, dmax=NULL, dist.matrix=NULL, ...)
```

Arguments

`seqdata`	a state sequence object as defined by the `seqdef` function.
`criterion`	the representativeness criterion for sorting the candidate list. One of `"freq"` (sequence frequency), `"density"` (neighborhood density), `"mscore"` (mean state frequency), `"dist"` (centrality) and `"prob"` (sequence likelihood). See details.
`score`	an optional vector containing the representativeness scores used to sort the sequences in the candidate list. The length of the vector must be equal to the number of sequences in the sequence object.
`decreasing`	if a score vector is provided, indicates whether the objects in the candidate list must be sorted in ascending or descending order of this score. Default is TRUE, i.e. descending. The first object in the candidate list is then supposed to be the most representative.
`trep`	coverage threshold, i.e. minimum proportion of sequences that should have a representative in their neighborhood (neighborhood diameter is defined by `tsim`).
`nrep`	number of representative sequences. If `NULL` (default), the size of the representative set is controlled by `trep`.
`tsim`	neighborhood diameter as a percentage of the maximum (theoretical) distance. Defaults to 0.1 (10%). This diameter serves for evaluating redundancy.
`dmax`	maximum theoretical distance. The neighborhood diameter is defined as a proportion of this maximum distance. If `NULL`, it is derived from the distance matrix.
`dist.matrix`	a matrix containing the pairwise distances between sequences in `seqdata`. If `NULL`, the matrix is computed by calling the `seqdist` function. In that case, optional arguments to be passed to the `seqdist` function (see ... hereafter) should also be provided.

`...`	optional arguments to be passed to the `seqdist` function, mainly `dist.method` specifying the metric for computing the distance matrix, `norm` for normalizing the distances, `indel` and `sm` for indel and substitution costs when Optimal Matching metric is chosen. See `seqdist` manual page for details.

Details

The representative set is obtained by an heuristic that first builds a sorted list of candidates using a representativeness

score and then eliminates redundancy. The available criterions for sorting the candidate list are: *sequence frequency*, *neighborhood density*, *mean state frequency*, *centrality* and *sequence likelihood*.

The *sequence frequency* criterion uses the sequence frequencies as representativeness score. The more frequent a sequence the more representative it is supposed to be. Hence, sequences are sorted in decreasing frequency order.

The *neighborhood density* criterion uses the number — the density — of sequences in the neighborhood of each candidate sequence. This requires indeed to set the neighborhood diameter `tsim`. We suggest to set it as a given proportion of the maximal theoretical distance between two sequences. Sequences are sorted in decreasing density order.

The *mean state frequency* criterion is the mean value of the transversal frequencies of the successive states. Let $s = s_1 s_2 \cdots s_\ell$ be a sequence of length ℓ and $(f_{s_1}, f_{s_2}, \ldots, f_{s_\ell})$ the frequencies of the states at (time-)position $(t_1, t_2, \ldots t_\ell)$. The mean state frequency is the sum of the state frequencies divided by the sequence length

$$MSF(s) = \frac{1}{\ell} \sum_{i=1}^{\ell} f_{s_i}$$

The lower and upper boundaries of MSF are 0 and 1. MSF is equal to 1 when all the sequences in the set are the same, i.e. when there is a single distinct sequence. The most representative sequence is the one with the highest score.

The *centrality* criterion uses the sum of distances to all other sequences as a representativeness criterion. The smallest the sum, the most representative the sequence.

The *sequence likelihood* $P(s)$ is defined as the product of the probability with which each of its observed successive state is supposed to occur at its position. Let $s = s_1 s_2 \cdots s_\ell$ be a sequence of length ℓ. Then

$$P(s) = P(s_1, 1) \cdot P(s_2, 2) \cdots P(s_\ell, \ell)$$

with $P(s_t, t)$ the probability to observe state s_t at position t.
The question is how to determinate the state probabilities $P(s_t, t)$. One commonly used method for computing them is to postulate a Markov model, which can be of various order. The implemented criterion considers the probabilities derived from the first order Markov model, that is each $P(s_t, t)$, $t > 1$ is set to the transition rate $p(s_t|s_{t-1})$ estimated across sequences from the observations at positions t and $t-1$. For $t = 1$, we set $P(s_1, 1)$ to the observed frequency of the state s_1 at position 1.

The likelihood $P(s)$ being generally very small, we use $-\log P(s)$ as sorting criterion. The latter quantity is

minimal when $P(s)$ is equal to 1, which leads to sort the sequences in ascending order of their score.

For more details, see Gabadinho et al., 2009.

Value

An object of class `stslist.rep`. This is actually a state sequence object (containing a list of state sequences) with the following additional attributes:

`Scores`	a vector with the representative score of each sequence in the original set given the chosen criterion.
`Distances`	a matrix with the distance of each sequence to its nearest representative.
`Statistics`	contains several quality measures for each representative sequence in the set: number of sequences attributed to the representative, number of sequence in the representatives neighborhood, mean distance to the representative.
`Quality`	overall quality measure.

Print, plot and summary methods are available. More elaborated plots are produced by the `seqplot` function using the

`type="r"` argument, or the `seqrplot` alias.

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2009). Summarizing Sets of Categorical Sequences, In International Conference on Knowledge Discovery and Information Retrieval, Madeira, 6-8 October, INSTICC.

See Also

`seqplot`, `plot.stslist.rep`

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)

## Computing the distance matrix
costs <- seqsubm(biofam.seq, method="TRATE")
biofam.om <- seqdist(biofam.seq, method="OM", sm=costs)

## Representative set using the neighborhood density criterion
biofam.rep <- seqrep(biofam.seq, dist.matrix=biofam.om, criterion="density")
biofam.rep
summary(biofam.rep)
plot(biofam.rep)
```

seqsep *Adds separators to sequences stored as character string*

Description

Adds separators to sequences stored as character string.

Usage

```
 seqsep(seqdata, sl=1, sep="-")
```

Arguments

`seqdata`	a dataframe or matrix containing sequence data, as vectors of states or events.
`sl`	the length of the states (the number of characters used to represent them). Default to 1.
`sep`	the character used as separator. Set by default to "-".

See Also

`seqdecomp`.

Examples

```
seqsep("ABAAAAAAD")
```

seqST *Sequences turbulence*

Description

Computes the turbulence for each sequence in a sequence data set, using the measure proposed by Elzinga.

Usage

```
 seqST(seqdata)
```

Arguments

`seqdata`	a sequence object as returned by the the `seqdef` function.

Details

Sequence turbulence is a measure proposed by *Elzinga (2007)*. It is based on the number $\phi(x)$ of distinct subsequences that can be extracted from the distinct state sequence and the variance of the consecutive times t_i spent in the distinct states. For a sequence x, the formula is

$$T(x) = \log_2(\phi(x)\,\frac{s^2_{t,max}(x)+1}{s^2_t(x)+1})$$

where s^2_t is the variance of the state-duration for the x sequence and $s^2_{t,max}$ is the maximum value that this variance can take given the total duration of the sequence. This maximum is computed as follow

$$s^2_{t,max} = (n-1)(1-\bar{t})$$

where $\bar{t}$ is the mean consecutive time spent in the distinct states, i.e. the sequence duration divided by the number of distinct states in the sequence.

Value

a vector whose number of elements is the number of sequences in `seqdata`, containing the turbulence value of each sequence.

References

Elzinga, Cees H. and Liefbroer, Aart C. (2007). De-standardization of Family-Life Trajectories of Young Adults: A Cross-National Comparison Using Sequence Analysis. *European Journal of Population*, 23, 225-250.

See Also

`seqient` for computing the within sequence entropy.

Examples

```
## Loading the 'actcal' example data set
data(actcal)

## Defining a sequence object with data in columns 13 to 24
## (activity status from january to december 2000)
actcal.seq <- seqdef(actcal,13:24, informat='STS')

## Computing the sequences turbulence
turb <- seqST(actcal.seq)

## Histogram for the turbulence
hist(turb)
```

seqstatd — *Sequence of transversal state distributions and their entropies*

Description

Returns the state frequencies, the number of valid states and the entropy of the state distribution at each position in the sequence.

Usage

```
seqstatd(seqdata, weighted=TRUE, with.missing=FALSE, norm=TRUE)
```

Arguments

`seqdata`	a state sequence object as defined by the `seqdef` function.
`weighted`	if TRUE, distributions account for the weights assigned to the state sequence object (see `seqdef`). Set to FALSE if you want ignore the weights.
`with.missing`	If FALSE (default value), returned distributions ignore missing values.
`norm`	if TRUE (default value), entropy is normalized, ie divided by the entropy of the alphabet. Set to FALSE if you want the entropy without normalization.

Details

In addition to the state distribution at position in the sequence, the `seqstatd` function provides also for each time point the number of valid states and the Shannon entropy of the observed state distribution. Letting p_i denote the proportion of cases in state i at the considered time point, the entropy is

$$h(p_1, \ldots, p_s) = -\sum_{i=1}^{s} p_i \log_2(p_i)$$

where s is the size of the alphabet. The entropy is 0 when all cases are in the same state and is maximal when the same proportion of cases are in each state. The entropy can be seen as a measure of the diversity of states observed at the considered time point. An application of such a measure (but with aggregated transversal data) can be seen in *Billari (2001)* and *Fussell (2005).*

References

Billari, F. C. (2001). The analysis of early life courses: complex descriptions of the transition to adulthood. *Journal of Population Research* 18 (2), 119-24.

Fussell, E. (2005). Measuring the early adult life course in Mexico: An application of the entropy index. In R. Macmillan (Ed.), *The Structure of the Life Course: Standardized? Individualized? Differentiated?*, Advances in Life Course Research, Vol. 9, pp. 91-122. Amsterdam: Elsevier.

See Also

`plot.stslist.statd` the plot method for objects of class `stslist.statd`, `seqdplot` and `seqHtplot` for higher level plots.

Examples

```
data(biofam)
biofam.seq <- seqdef(biofam,10:25)
sd <- seqstatd(biofam.seq)
## Plotting the state distribution
plot(sd, type="d")

## Plotting the entropy indexes
plot(sd, type="Ht")

## ====================
## example with weights
## ====================
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## Unweighted
seqstatd(ex1.seq, weighted=FALSE)

seqstatd(ex1.seq, weighted=TRUE)
```

`seqstatf` *State frequencies in the all sequence data set*

Description

Frequency of each state of the alphabet in the all sequence data set.

Usage

```
seqstatf(seqdata, weighted = TRUE)
```

Arguments

`seqdata`	a sequence object as defined by the `seqdef` function.
`weighted`	if TRUE, frequencies account for the weights assigned to the state sequence object (see `seqdef`). Set to FALSE if you want ignore the weights. If no weights were assigned during the creation of the sequence object, `weighted=TRUE` will yield the same result as `weighted=FALSE` since each sequence is allowed a weight of 1.

Details

The `seqstatf` function computes the (weighted) raw and percentage frequency of each state of the alphabet in `seqdata`, i.e the (weighted) sum of the occurences of a state in `seqdata`.

Value

a data.frame with as many rows as the number of states in the alphabet and two columns, one for the raw frequencies (Freq) and one for the percentage frequencies.

See Also

`seqstatd` for the state distribution by time point (position), `seqistatd` for the state distribution within each sequence.

Examples

```
## Creating a sequence object from the actcal data set
data(actcal)
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal, 13:24, labels=actcal.lab)

## States frequencies
seqstatf(actcal.seq)

## Example with weights
data(ex1)
ex1.seq <- seqdef(ex1, 1:13, weights=ex1$weights)

## Unweighted
seqstatf(ex1.seq, weighted=FALSE)

## Weighted
seqstatf(ex1.seq, weighted=TRUE)
```

`seqstatl` *List of distinct states or events (alphabet) in a sequence data set.*

Description

Returns a list containing distinct states or events found in a data frame or matrix containing sequence data, the alphabet.

Usage

```
seqstatl(data, var=NULL, format='STS')
```

Arguments

`data`	a data frame or matrix containing sequence data.
`var`	the list of columns containing the sequences. Default is NULL, i.e. all the columns. Whether the sequences are in the compressed (character strings) or extended format is automatically detected from the number of columns.

`format`	the format of the sequence data set. One of 'STS', 'SPS', 'DSS'. Default is 'STS'. The `seqstatl` function uses the `seqformat` function to translate between formats when necessary.

See Also

`seqformat`

Examples

```
data(actcal)
seqstatl(actcal,13:24)
```

`seqsubm` *Create a substitution-cost matrix*

Description

The substitution-cost matrix is used when computing distances between sequences by the method of optimal matching. The function creates the substitution matrix using either a constant or the transition rates computed from the sequence data or other methods to be implemented in the future.

Usage

```
seqsubm(seqdata, method, cval=2, with.miss=FALSE,
        miss.cost=2, time.varying=FALSE, weighted=TRUE)
```

Arguments

`seqdata`	a sequence object as returned by the seqdef function.
`method`	method to compute transition rates. At this time, the methods available are constant value (method="CONSTANT") or substitution costs using transition rates (method="TRATE")
`cval`	the constant substitution cost if method "CONSTANT" is choosen. Otherwise, do not specify.
`with.miss`	if `TRUE`, an additional entry is added in the matrix for the missing states. Hence, a new "missing" state is added to the list of "valid" states. Use this if you want to compute distances with missing values inside the sequences. See *Gabadinho, 2008* for more details on the options for handling missing values when computing distances between sequences.
`miss.cost`	the substitution cost for the missing state.
`time.varying`	Logical. If TRUE return an `array` containing a distinct matrix for each time unit. The time is the third dimension (subscript).
`weighted`	Logical. If TRUE compute transition rates using weights specified in `seqdata`.

Details

The substitution-cost matrix has dimension $ns*ns$, where ns is the number of states in the alphabet of the sequence object. The element (i, j) of the matrix is the cost of substituting state i whith state j. In the "constant" method, the substitution costs are the same for all the states, with a default value of 2. An alternate value can be provided by the user. When the "transition rates" method is choosen, the transition rates between all states are computed using the seqtrate function. The substitution cost between states i and j is obtained with the formula

$$SC(i, j) = 2 - P(i, j) - P(j, i)$$

where $P(i, j)$ is the transition rate between states i and j.

References

Gabadinho, A., G. Ritschard, M. Studer and N. S. Müller (2008). Mining Sequence Data in `R` with `TraMineR`: A user's guide. *Department of Econometrics and Laboratory of Demography, University of Geneva*.

See Also

`seqtrate` `seqdef`.

Examples

```
## Defining a sequence object with columns 10 to 25
## in the 'biofam' example data set
data(biofam)
biofam.seq <- seqdef(biofam,10:25)

## Optimal matching using transition rates based substitution-cost matrix
## and insertion/deletion costs of 3
trcost <- seqsubm(biofam.seq, method="TRATE")
biofam.om <- seqdist(biofam.seq,method="OM",indel=3,sm=trcost)

## Optimal matching using constant value (2) substitution-cost matrix
## and insertion/deletion costs of 3
ccost <- seqsubm(biofam.seq, method="CONSTANT", cval=2)
biofam.om.c2 <- seqdist(biofam.seq, method="OM",indel=3,sm=ccost)

## Displaying the distance matrix for the first 10 sequences
biofam.om.c2[1:10,1:10]

## =================================
## Example with weights and missings
## =================================
data(ex1)
ex1.seq <- seqdef(ex1,1:13, weights=ex1$weights)

## Unweighted
subm <- seqsubm(ex1.seq, method="TRATE", with.miss=TRUE, weighted=FALSE)
ex1.om <- seqdist(ex1.seq, method="OM", sm=subm, with.miss=TRUE)
```

```
## Weighted
subm.w <- seqsubm(ex1.seq, method="TRATE", with.miss=TRUE, weighted=TRUE)
ex1.omw <- seqdist(ex1.seq, method="OM", sm=subm.w, with.miss=TRUE)

ex1.om == ex1.omw
```

---

seqsubsn — *Number of distinct subsequences in a sequence.*

Description

Computes the number of distinct subsequences in a sequence using Elzinga's algorithm.

Usage

```
seqsubsn(seqdata, DSS=TRUE)
```

Arguments

`seqdata`	a sequence object as defined by the `seqdef` function.
`DSS`	if TRUE, the Distinct State Sequences (DSS, see `seqdss`) are first extracted, eg. the DSS contained in 'D-D-D-D-A-A-A-A-A-A-A-D' is 'D-A-D', and the number of distinct subsequences in the DSS is computed. If FALSE, the number of distinct subsequences is computed from sequences as they appear in the input sequence object. Hence the number of distinct subsequences is in most cases much higher with the DSS=FALSE option.

Value

a vector containing the number of distinct subsequences for each sequence in the input sequence object.

See Also

`seqdss`.

Examples

```
data(actcal)
actcal.seq <- seqdef(actcal,13:24)

## Number of subsequences with DSS=TRUE
seqsubsn(actcal.seq[1:10,])

## Number of subsequences with DSS=FALSE
seqsubsn(actcal.seq[1:10,],DSS=FALSE)
```

seqtab — *Frequency table of the sequences*

Description

Computes the frequency table of the sequences (count and percent of each sequence).

Usage

```
seqtab(seqdata, tlim=10, weighted=TRUE, format="SPS")
```

Arguments

`seqdata`	a sequence object as defined by the `seqdef` function.
`tlim`	if tlim>0, return frequencies only for the 'tlim' most frequent sequences. Default to 10.
`weighted`	if TRUE, frequencies account for the weights assigned to the state sequence object (see `seqdef`). Set to FALSE if you want ignore the weights. If no weights were assigned during the creation of the sequence object, `weighted=TRUE` will yield the same result as `weighted=FALSE` since each sequence is allowed a weight of 1.
`format`	format used for displaying sequences as rownames in the output table. Default is SPS format, which yields shorter and more readable sequence representations. Alternatively, "STS" may be specified.

Value

An object of class `stslist.freq`. This is actually a state sequence object (containing a list of state sequences) with added attributes, among others the `freq` attribute containing the frequency table. There are `print` and `plot` methods for such objects. More sophisticated plots can be produced with the `seqplot` function.

See Also

`seqplot`, `seqplot`.

Examples

```
## Creating a sequence object from the actcal data set
data(actcal)
actcal.lab <- c("> 37 hours", "19-36 hours", "1-18 hours", "no work")
actcal.seq <- seqdef(actcal, 13:24, labels=actcal.lab)

## 10 most frequent sequences in the data
seqtab(actcal.seq, tlim=10)

## With tlim=0, we get all distinct sequences in the data set
## sorted according to their frequency
seqtab(actcal.seq, tlim=0)

## Example with weights
## from biofam data set using weigths
data(ex1)
ex1.seq <-  seqdef(ex1, 1:13, weights=ex1$weights)

## Unweighted frequencies
seqtab(ex1.seq, weighted=FALSE)

## Weighted frequencies
seqtab(ex1.seq, weighted=TRUE)
```

`seqtrate` *Compute transition rates between states*

Description

Returns a matrix with transition rates between states, computed from a set of sequences.

Usage

```
seqtrate(seqdata, statl=NULL, time.varying=FALSE, weighted=TRUE)
```

Arguments

`seqdata`	a sequence object as defined by the `seqdef` function.
`statl`	a list of states or events for which the transition rates will be computed. If omitted (default), transition rates are computed between the distinct states in seqdata (obtained with the `alphabet` function).
`time.varying`	Logical. If TRUE return an `array` containing a distinct matrix for each time unit. The time is the third dimension (subscript).
`weighted`	Logical. If TRUE compute transition rates using weights specified in `seqdata`.

Details

Transition rates are the probabilities of transition from one state to another observed in the sequence data. Substitution costs based on transition rates can be used when computing distances between sequences with the optimal matching method (see seqdist).

Value

a matrix of dimension $ns * ns$, where ns is the number of states in the alphabet of the sequence object.

See Also

seqdist seqsubm alphabet.

Examples

```
  ## Loading the 'actcal' example data set
  data(actcal)

  ## Defining a sequence object with data in columns 13 to 24
  ## (activity status from january to december 2000)
  actcal.seq <- seqdef(actcal,13:24,informat='STS')

  ## Computing transition rates
  seqtrate(actcal.seq)

  ## Computing transition rates between states "A" and "B" only
  seqtrate(actcal.seq, c("A","B"))

## ====================
## Example with weights
## ====================
data(ex1)
ex1.seq <- seqdef(ex1,1:13, weights=ex1$weights)

seqtrate(ex1.seq, weighted=FALSE)
seqtrate(ex1.seq, weighted=TRUE)
```

---

seqtree2dot — *Graphical representation of a dissimilarity tree*

Description

This function offers shortcuts to generate a "dot" file and associated images files that can be used in GraphViz to get a graphical representation of the tree.

Usage

```
seqtree2dot(tree, filename, seqdata, imgLeafOnly=FALSE, sortv=NULL, ...)
```

Arguments

`tree`	A tree object to be plotted as defined by `disstree`
`filename`	A filename, without extension, that will be used to generate image and dot files
`seqdata`	a sequence object as defined by the the `seqdef` function.
`sortv`	The name of an optional variable used to sort the data before plotting, see `plot.stslist`.
`imgLeafOnly`	If `TRUE`, only terminal node will be plotted
`...`	other parameters that will be passed to `plot.stslist`

Details

This function generates a "dot" file and one image file per node. For each node, it calls `plot.stslist` passing the selected lines of `seqdata` as argument. `seqtree2dot` is a shortcut for sequences objects using the plot function `plot.stslist`.

Value

Nothing but generates a file in the current working directory (see `setwd`).

See Also

`disstree` for examples

`TraMineR.checkupdates` *Check for updates*

Description

Check if the installed version of TraMineR is up-to-date. This function only prints a message and does not need any argument. It connects to the TraMineR webserver (http://mephisto.unige.ch/TraMineR).

Usage

```
TraMineR.checkupdates()
```

Value

Return your current version number of TraMineR and the latest stable and development version number if more recent versions are available.

Index