Package ‘seqimpute’

March 27, 2024

Type Package

Title Imputation of Missing Data in Sequence Analysis

Version 2.0.0

Description Multiple imputation of missing data present in a dataset through the prediction based on either a random forest or a multinomial regression model. Covariates and time-dependent covariates can be included in the model. The prediction of the missing values is based on the method of Halpin (2012) <https://researchrepository.ul.ie/articles/report/Multiple_imputation_for_life-course_sequence_data/19839736>.

License GPL-2

Imports Amelia, cluster, dfidx, doRNG, doSNOW, dplyr, foreach, graphics, mlr, nnet, parallel, plyr, ranger, rms, stats, stringr, TraMineR, TraMineRextras, utils, mice

Suggests R.rsp, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder R.rsp

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

NeedsCompilation no

Author Kevin Emery [aut, cre],
Anthony Guinchard [aut],
Andre Berchtold [aut],
Kamyar Taher [aut]

Maintainer Kevin Emery <kevin.emery@unige.ch>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2024-03-27 13:00:02 UTC

R topics documented:

---

addcluster — *Function that adds the clustering result to a* `seqimp` *object obtained with the* `seqimpute` *function*

Description

Function that adds the clustering result to a `seqimp` object obtained with the `seqimpute` function

Usage

```
addcluster(impdata, clustering)
```

Arguments

`impdata` An object of class `seqimp` as created by the `seqimpute` function

`clustering` clustering made on the multiple imputed dataset. Can either be a dataframe or a matrix, where each row correspond to an observation and each column to a multiple imputed dataset

Value

Returns a `seqimp` object containing the cluster to which each sequence in each imputed dataset belongs. Specifically, a column named cluster is added to the imputed datasets.

fromseqimp *Transform an object of class* seqimp *into a dataframe or a* mids *object*

Description

The function converts a seqimp object into a specified format.

Usage

```
fromseqimp(data, format = "long", include = FALSE)
```

Arguments

data — An object of class seqimp as created by the function seqimpute

format — The format in which the seqimp object should be returned. It could be: "long", "stacked" and "mids". See the Details section for the interpretation.

include — logical that indicates if the original dataset with missing value should be included or not. This parameter does not apply if format="mids".

Details

The argument format specifies the object that should be returned by the function. It can take the following values

"long" produces a data set in which imputed data sets are stacked vertically. The following columns are added: 1) .imp referring to the imputation number, and 2) .id the row names of the original dataset

"stacked" the same as "long", but without the inclusion of the two columns .imp and .id

"mids" produces an object of class mids, which is the format used by the mice package.

Value

Transform a seqimp object into the desired format.

Author(s)

Kevin Emery

Examples

```
## Not run:
# Imputation with the MICT algorithm
imp <- seqimpute(data = gameadd, var = 1:4)

# The object imp is transformed to a dataframe, where completed datasets are
# stacked vertically
```

```
imp.stacked <- fromseqimp(data = imp,
    format = "stacked", include = FALSE)

## End(Not run)
```

`gameadd` *Example data set: Game addiction*

Description

Dataset containing variables on the gaming addiction of young people. The data consists of gaming addiction, coded as either ’no’ or ’yes’, measured over four consecutive years for 500 individuals, three covariates and one time-dependent covariate. The yearly states are recorded in columns 1 (`T1_abuse`) to 4 (`T4_abuse`).

The three covariates are

- `Gender` (female or male),
- `Age` (measured at time 1),
- `Track` (school or apprenticeship).

The time-varying covariate consists of the individual’s relationship to gambling at each of the four time points, appearing in columns `T1_gambling`, `T2_gambling`, `T3_gambling`, and `T4_gambling`. The states are either no, gambler or problematic gambler

Usage

```
data(gameadd)
```

Format

A data frame containing 500 rows, 4 states variable, 3 covariates and a time-dependent covariate.

`plot.seqimp` *Plot a* `seqimp` *object*

Description

Plot a `seqimp` object. The state distribution plot of the first `m` completed datasets is shown, possibly alongside the original dataset with missing data

Usage

```
## S3 method for class 'seqimp'
plot(x, m = 5, include = TRUE, ...)
```

Arguments

`x`	Object of class `seqimp`
`m`	Number of completed datasets to show
`include`	logical that indicates if the original dataset with missing value should be plotted or not
`...`	Arguments to be passed to the seqdplot function

Author(s)

Kevin Emery

`print.seqimp` *Print a* `seqimp` *object*

Description

Print a `seqimp` object

Usage

```
## S3 method for class 'seqimp'
print(x, ...)
```

Arguments

`x`	Object of class `seqimp`
`...`	additional arguments passed to other functions

Author(s)

Kevin Emery

`seqaddNA` *Generation of missing on longitudinal categorical data.*

Description

Generation of missing data under the form of gaps, which is the typical form of missing data with longitudinal data. It simulates MCAR or MAR missing data.

Usage

```
seqaddNA(
  data,
  var = NULL,
  states.high = NULL,
  propdata = 1,
  pstart.high = 0.1,
  pstart.low = 0.005,
  maxgap = 3,
  only.traj = FALSE
)
```

Arguments

`data`	a data frame containing sequences of a multinomial variable with missing data (coded as `NA`)
`var`	the list of columns containing the trajectories. Default is `NULL`, i.e. all the columns.
`states.high`	list of states that have a larger probability of triggering a subsequent missing data gap
`propdata`	proportion observations for which missing data is simulated
`pstart.high`	probability to start a missing data for the states specified with the `states.high` argument
`pstart.low`	probability to start a missing data for the other states
`maxgap`	maximum length of a missing data gap
`only.traj`	logical that specifies whether only the trajectories should be returned (`only.traj=TRUE`), or the whole data (`only.traj=FALSE`)

Value

Returns a data frame on which missing data were simulated

Author(s)

Kevin Emery

Examples

```
# Generate MCAR missing data on the mvad dataset
# from the TraMineR package

## Not run:
data(mvad, package = "TraMineR")
mvad.miss <- seqaddNA(mvad, var = 17:86)


# Generate missing data on mvad where joblessness is more likely to trigger
# a missing data gap
mvad.miss2 <- seqaddNA(mvad, var = 17:86,  states.high = "joblessness")

## End(Not run)
```

---

seqcomplete *Extract all the trajectories without missing value.*

Description

Extract all the trajectories without missing value.

Usage

```
seqcomplete(data, var = NULL)
```

Arguments

data	either a data frame containing sequences of a multinomial variable with missing data (coded as NA) or a state sequence object built with the TraMineR package
var	the list of columns containing the trajectories. Default is NULL, i.e. all the columns.

Value

Returns either a data frame or a state sequence object, depending the type of data that was provided to the function

Author(s)

Kevin Emery

Examples

```
# Game addiction dataset
data(gameadd)
# Extract the trajectories without any missing data
gameadd.complete <- seqcomplete(gameadd, var = 1:4)
```

`seqimpute` *seqimpute: Imputation of missing data in longitudinal categorical data*

Description

The seqimpute package implements the MICT and MICT-timing methods. These are multiple imputation methods for longitudinal data. The core idea of the algorithms is to fills gaps of missing data, which is the typical form of missing data in a longitudinal setting, recursively from their edges. The prediction is based on either a multinomial or a random forest regression model. Covariates and time-dependent covariates can be included in the model.

The MICT-timing algorithm is an extension of the MICT algorithm designed to address a key limitation of the latter: its assumption that position in the trajectory is irrelevant.

Usage

```
seqimpute(
  data,
  var = NULL,
  np = 1,
  nf = 1,
  m = 5,
  timing = FALSE,
  frame.radius = 0,
  covariates = NULL,
  time.covariates = NULL,
  regr = "multinom",
  npt = 1,
  nfi = 1,
  ParExec = FALSE,
  ncores = NULL,
  SetRNGSeed = FALSE,
  verbose = TRUE,
  available = TRUE,
  pastDistrib = FALSE,
  futureDistrib = FALSE,
  ...
)
```

Arguments

`data`	a data frame containing sequences of a categorical variable with missing data (coded as `NA`)
`var`	the list of columns containing the trajectories. Default is NULL, i.e. all the columns.
`np`	number of previous observations in the imputation model of the internal gaps.

`nf` number of future observations in the imputation model of the internal gaps.

`m` number of multiple imputations (default: `5`).

`timing` a logical value that specifies if the MICT algorithm (timing=FALSE) or the MICT-timing algorithm (timing=TRUE) should be used.

`frame.radius` parameter relative to the MICT-timing algorithm specifying the radius of the timeframe.

`covariates` the list of columns containing the covariates to include in the imputation process

`time.covariates`
the list of columns containing the time-varying covariates to include in the imputation process

`regr` a character specifying the imputation method. If `regr="multinom"`, multinomial models are used, while if `regr="rf"`, random forest models are used.

`npt` number of previous observations in the imputation model of the terminal gaps.

`nfi` number of future observations in the imputation model of the initial gaps.

`ParExec` logical. If `TRUE`, the multiple imputations are run in parallel. This allows faster run time depending of how many cores the processor has.

`ncores` integer. Number of cores to be used for the parallel computation. If no value is set for this parameter, the number of cores will be set to the maximum number of CPU cores minus 1.

`SetRNGSeed` an integer that is used to set the seed in the case of parallel computation. Note that setting `set.seed()` alone before the seqimpute function won't work in case of parallel computation.

`verbose` logical. If `TRUE`, seqimpute will print history and warnings on console. Use `verbose=FALSE` for silent computation.

`available` a logical value allowing the user to choose whether to consider the already imputed data in the predictive model (`available = TRUE`) or not (`available = FALSE`).

`pastDistrib` a logical indicating if the past distribution should be used as predictor in the imputation model.

`futureDistrib` a logical indicating if the future distribution should be used as predictor in the imputation model.

`...` Named arguments that are passed down to the imputation functions.

Details

The imputation process is divided into several steps, depending on the type of gaps of missing data. The order of imputation of the gaps are:

`Internal gap:` there is at least `np` observations before an internal gap and `nf` after the gap

`Initial gap:` gaps situated at the very beginning of a trajectory

`Terminal gap:` gaps situated at the very end of a trajectory

`Left-hand side specifically located gap (SLG):` gaps that have at least `nf` observations after the gap, but less than `np` observation before it

Right-hand side SLG: gaps that have at least np observations before the gap, but less than nf observation after it

Both-hand side SLG: gaps that have less than np observations before the gap, and less than nf observations after it

The primary difference between the MICT and MICT-timing algorithms lies in their approach to selecting patterns from other sequences for fitting the multinomial model. While the MICT algorithm considers all similar patterns regardless of their temporal placement, MICT-timing restricts pattern selection to those that are temporally closest to the missing value. This refinement ensures that the imputation process adequately accounts for temporal dynamics, resulting in more accurate imputed values.

Value

Returns an S3 object of class seqimp.

Author(s)

Kevin Emery <kevin.emery@unige.ch>, Andre Berchtold, Anthony Guinchard, and Kamyar Taher

References

HALPIN, Brendan (2012). Multiple imputation for life-course sequence data. Working Paper WP2012-01, Department of Sociology, University of Limerick. http://hdl.handle.net/10344/3639.

HALPIN, Brendan (2013). Imputing sequence data: Extensions to initial and terminal gaps, Stata's. Working Paper WP2013-01, Department of Sociology, University of Limerick. http://hdl.handle.net/10344/3620

Examples

```
# Default multiple imputation of the trajectories of game addiction with the
# MICT algorithm

## Not run:
set.seed(5)
imp1 <- seqimpute(data = gameadd, var = 1:4)


# Default multiple imputation with the MICT-timing algorithm
set.seed(3)
imp2 <- seqimpute(data = gameadd, var = 1:4, timing = TRUE)


# Inclusion in the MICt-timing imputation process of the three background
# characteristics (Gender, Age and Track), and the time-varying covariate
# about gambling


set.seed(4)
imp3 <- seqimpute(data = gameadd, var = 1:4, covariates = 5:7,
  time.covariates = 8:11)
```

```
# Parallel computation


imp4 <- seqimpute(data = gameadd, var = 1:4, covariates = 5:7,
  time.covariates = 8:11, ParExec = TRUE, ncores=5, SetRNGSeed = 2)

## End(Not run)
```

seqmissfplot *Plot the most common patterns of missing data.*

Description

Plot function that renders the most frequent patterns of missing data. This function is based on the seqfplot function.

Usage

```
seqmissfplot(data, var = NULL, with.complete = TRUE, ...)
```

Arguments

`data` a data.frame where missing data are coded as `NA` or a state sequence object built with seqdef function

`var` the list of columns containing the trajectories. Default is NULL, i.e. all the columns.

`with.complete` a logical stating if complete trajectories should be included or not in the plot

`...` parameters to be passed to the seqfplot function

Details

This plot function is based on the seqfplot function. To see which arguments can be changed, see the seqfplot help. In particular, the number of most frequent patterns to be plotted can be changed with the argument `idxs`. By default, the 10 most frequent patterns are plotted.

Author(s)

Kevin Emery

Examples

```
# Plot the 10 most common patterns of missing data

seqmissfplot(gameadd, var=1:4)

# Plot the 10 most common patterns of missing data discarding
# complete trajectories

seqmissfplot(gameadd, var=1:4, with.missing = FALSE)

# Plot only the 5 most common patterns of missing data discarding
# complete trajectories

seqmissfplot(gameadd, var=1:4, with.missing = FALSE, idxs = 1:5)
```

---

seqmissimplic *Identification and visualization of states that best characterize sequences with missing data*

Description

Function based on the seqimplic. Identification and visualization of the states that best characterize the sequence with missing data vs. the sequences without missing data at each position (time point). See the seqimplic help for more details on how it works.

Usage

```
seqmissimplic(data, var = NULL, ...)
```

Arguments

`data`	a data frame where missing data are coded as `NA` or a state sequence object built with seqdef function
`var`	the list of columns containing the trajectories. Default is NULL, i.e. all the columns.
`...`	parameters to be passed to the seqimplic function

Value

returns a `seqimplic` object that can be plotted and printed.

Author(s)

Kevin Emery

Examples

```
# For illustration purpose, we simulate missing data on the mvad dataset,
# available in the TraMineR package. The state "joblessness" state has a
# higher probability of triggering a missing gap

## Not run:
data(mvad, package = "TraMineR")
mvad.miss <- seqaddNA(mvad, var = 17:86, states.high = "joblessness")

# The states that best characterize sequences with missing data
implic <- seqmissimplic(mvad.miss, var = 17:86)

# Visualization of the results
plot(implic)

## End(Not run)
```

seqmissIplot *Plot all the patterns of missing data.*

Description

#' @description Plot function that renders all the patterns of missing data. This function is based on the seqIplot function.

Usage

```
seqmissIplot(data, var = NULL, with.complete = TRUE, ...)
```

Arguments

`data` a data.frame where missing data are coded as `NA` or a state sequence object built with seqdef function

`var` the list of columns containing the trajectories. Default is NULL, i.e. all the columns.

`with.complete` a logical stating if complete trajectories should be included or not in the plot

`...` parameters to be passed to the seqIplotfunction

Author(s)

Kevin Emery

Examples

```
# Plot all the patterns of missing data

seqmissIplot(gameadd, var=1:4)

# Plot all the patterns of missing data discarding
# complete trajectories

seqmissIplot(gameadd, var=1:4, with.missing = FALSE)
```

---

seqQuickLook *Summary of the types of gaps among a dataset*

Description

The `seqQuickLook()` function aimed at providing an overview of the number and size of the different types of gaps spread in the original dataset.

Usage

```
seqQuickLook(data, var = NULL, np = 1, nf = 1)
```

Arguments

`data`	a data.frame where missing data are coded as NA or a state sequence object built with seqdef function
`var`	the list of columns containing the trajectories. Default is NULL, i.e. all the columns.
`np`	number of previous observations in the imputation model of the internal gaps.
`nf`	number of future observations in the imputation model of the internal gaps.

Details

The distinction between internal and SLG gaps depends on the number of previous (`np`) and future (`nf`) observations that are set for the `MICT` and `MICT-timing` algorithms.

Value

Returns a `data.frame` object that summarizes, for each type of gaps (Internal Gaps, Initial Gaps, Terminal Gaps, LEFT-hand side SLG, RIGHT-hand side SLG, Both-hand side SLG), the minimum length, the maximum length, the total number of gaps and the total number of missing they contain.

Author(s)

Andre Berchtold and Kevin Emery

Examples

```
data(gameadd)

seqQuickLook(data = gameadd, var = 1:4, np = 1, nf = 1)
```

---

seqTrans *Spotting impossible transitions in longitudinal categorical data*

Description

The purpose of seqTrans is to spot impossible transitions in longitudinal categorical data.

Usage

```
seqTrans(data, var = NULL, trans)
```

Arguments

data a data frame containing sequences of a multinomial variable with missing data (coded as NA)

var the list of columns containing the trajectories. Default is NULL, i.e. all the columns.

trans character vector gathering the impossible transitions. For example: trans <- c("1->3","1->4","2->1","4->1","4->3")

Value

It returns a matrix where each row is the position of an impossible transition.

Author(s)

Andre Berchtold and Kevin Emery

Examples

```
data(gameadd)

seqTransList <- seqTrans(data = gameadd, var = 1:4, trans = c("yes->no"))
```

seqwithmiss *Extract all the trajectories with at least one missing value*

Description

Extract all the trajectories with at least one missing value

Usage

```
seqwithmiss(data, var = NULL)
```

Arguments

`data`	either a data frame containing sequences of a multinomial variable with missing data (coded as `NA`) or a state sequence object built with the TraMineR package
`var`	the list of columns containing the trajectories. Default is NULL, i.e. all the columns.

Value

Returns either a data frame or a state sequence object, depending the type of data that was provided to the function

Author(s)

Kevin Emery

Examples

```
# Game addiction dataset
data(gameadd)
# Extract the trajectories without any missing data
gameadd.withmiss <- seqwithmiss(gameadd, var = 1:4)
```

summary.seqimp *Summary of a* `seqimp` *object*

Description

Summary of a `seqimp` object

Usage

```
## S3 method for class 'seqimp'
summary(object, ...)
```

Arguments

`object`	Object of class `seqimp`
`...`	additional arguments passed to other functions

Author(s)

Kevin Emery

Index