Package ‘seqimpute’

January 15, 2025

Type Package

Title Imputation of Missing Data in Sequence Analysis

Version 2.2.0

Description Multiple imputation of missing data in a dataset using MICT or MICT-timing methods. The core idea of the algorithms is to fill gaps of missing data, which is the typical form of missing data in a longitudinal setting, recursively from their edges. Prediction is based on either a multinomial or random forest regression model. Covariates and time-dependent covariates can be included in the model.

License GPL-2

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

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

VignetteBuilder R.rsp

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

URL https://github.com/emerykevin/seqimpute

BugReports https://github.com/emerykevin/seqimpute/issues

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 2025-01-15 16:10:02 UTC

Contents

`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 in sequence based on a Markovian approach.

Usage

```
seqaddNA(
  data,
  var = NULL,
  states.high = NULL,
  propdata = 1,
  pstart.high = 0.1,
  pstart.low = 0.005,
  pcont = 0.66,
  maxgap = 3,
  maxprop = 0.75,
  only.traj = FALSE
)
```

Arguments

`data` A data frame containing sequences of a categorical (multinomial) variable, where missing data are coded as `NA`.

`var` A vector specifying the columns of the dataset that contain the trajectories. Default is `NULL`, meaning all columns are used.

`states.high` A list of states with a higher probability of initiating a subsequent missing data gap.

`propdata` Proportion of trajectories for which missing data is simulated, as a decimal between 0 and 1.

`pstart.high` Probability of starting a missing data gap for the states specified in the `states.high` argument.

`pstart.low` Probability of starting a missing data gap for all other states.

`pcont` Probability of a missing data gap to continue.

`maxgap` Maximum length of a missing data gap.

`maxprop` Maximum proportion of missing data allowed in a sequence, as a decimal between 0 and 1.

`only.traj` Logical, if `TRUE`, only the trajectories (specified in `var`) are returned. If `FALSE`, the entire data frame is returned.

Details

The first time point of a trajectory has a `pstart.low` probability to be missing. For the next time points, the probability to be missing depends on the previous time point. There are four cases:

1. If the previous time point is missing and the maximum length of a missing gap, which is specified by the argument `maxgap`, is reached, the time point is set as observed.

2. If the previous time point is missing, but the maximum length of a gap is not reached, there is a `pcont` probability that this time point is missing.

3. If the previous time point is observed and the previous time point belongs to the list of states specified by `pstart.high`, the probability to be missing is `pstart.high`.

4. If the previous time point is observed but the previous time point does not belong to the list of states specified by `pstart.high`, the probability to be missing is `pstart.low`.

If the proportion of missing data in a given trajectory exceeds the proportion specified by `maxprop`, the missing data simulation is repeated for the sequence.

Value

A data frame with simulated missing data.

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,
  end.impute = TRUE,
  verbose = TRUE,
  available = TRUE,
  pastDistrib = FALSE,
  futureDistrib = FALSE,
  ...
)
```

Arguments

`data` Either a data frame containing sequences of a categorical variable, where missing data are coded as `NA`, or a state sequence object created using the seqdef function. If using a state sequence object, any "void" elements will also be treated as missing. See the `end.impute` argument if you wish to skip imputing values at the end of the sequences.

`var` A specifying the columns of the dataset that contain the trajectories. Default is `NULL`, meaning all columns are used.

`np` Number of prior states to include in the imputation model for internal gaps.

`nf` Number of subsequent states to include in the imputation model for internal gaps.

`m` Number of multiple imputations to perform (default: `5`).

`timing` Logical, specifies the imputation algorithm to use. If `FALSE`, the MICT algorithm is applied; if `TRUE`, the MICT-timing algorithm is used.

`frame.radius` Integer, relevant only for the MICT-timing algorithm, specifying the radius of the timeframe.

`covariates` List of the columns of the dataset containing covariates to be included in the imputation model.

`time.covariates`
List of the columns of the dataset with time-varying covariates to include in the imputation model.

`regr` Character specifying the imputation method. Options include `"multinom"` for multinomial models and `"rf"` for random forest models.

`npt` Number of prior observations in the imputation model for terminal gaps (i.e., gaps at the end of sequences).

`nfi` Number of future observations in the imputation model for initial gaps (i.e., gaps at the beginning of sequences).

`ParExec` Logical, indicating whether to run multiple imputations in parallel. Setting to `TRUE` can improve computation time depending on available cores.

`ncores` Integer, specifying the number of cores to use for parallel computation. If unset, defaults to the maximum number of CPU cores minus one.

`SetRNGSeed` Integer, to set the random seed for reproducibility in parallel computations. Note that setting `set.seed()` alone does not ensure reproducibility in parallel mode.

`end.impute` Logical. If `FALSE`, missing data at the end of sequences will not be imputed.

`verbose` Logical, if `TRUE`, displays progress and warnings in the console. Use `FALSE` for silent computation.

`available` Logical, specifies whether to consider already imputed data in the predictive model. If `TRUE`, previous imputations are used; if `FALSE`, only original data are considered.

`pastDistrib` Logical, if `TRUE`, includes the past distribution as a predictor in the imputation model.

`futureDistrib` Logical, if `TRUE`, includes the future distribution as a 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, imping in more accurate imputed values.

Value

An object of class `seqimp`, which is a list with the following elements:

`data` A `data.frame` containing the original (incomplete) data.

`imp` A list of `m` `data.frame` corresponding to the imputed datasets.

`m` The number of imputations.

`method` A character vector specifying whether MICT or MICT-timing was used.

`np` Number of prior states included in the imputation model.

`nf` Number of subsequent states included in the imputation model.

`regr` A character vector specifying whether multinomial or random forest imputation models were applied.

`call` The call that created the object.

Author(s)

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

References

Halpin, B. (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, B. (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

Emery, K., Studer, M., & Berchtold, A. (2024). Comparison of imputation methods for univariate categorical longitudinal data. Quality & Quantity, 1-25. https://link.springer.com/article/10.1007/s11135-024-02028-z

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

This function plots the most frequent patterns of missing data, based on the seqfplot function.

Usage

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

Arguments

`data`	Either a data frame containing sequences of a categorical variable, where missing data are coded as `NA`, or a state sequence object created using the seqdef function.
`var`	A vector specifying the columns of the dataset that contain the trajectories. Default is `NULL`, meaning all columns are used.
`with.complete`	Logical, if `TRUE`, complete trajectories will be included in the plot.
`void.miss`	Logical, if `TRUE`, treats void elements as missing values. Applies only to state sequence objects created with seqdef. Note that the default behavior of `seqdef` is to treat missing data at the end of sequences as void elements.
`...`	Additional parameters passed to the seqfplot function.

Details

This plot function is based on the seqfplot function, allowing users to visualize patterns of missing data within sequences. For details on additional customizable arguments, see the seqfplot documentation.

By default, this function plots the 10 most frequent patterns. The number of patterns to be plotted can be adjusted using the `idxs` argument in `seqfplot`.

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

This function identifies and visualizes states that best characterize sequences with missing data at each position (time point), comparing them to sequences without missing data at each position (time point). It is based on the seqimplic function. For more information on the methodology, see the `seqimplic` documentation.

Usage

```
seqmissimplic(data, var = NULL, void.miss = TRUE, ...)
```

Arguments

`data` Either a data frame containing sequences of a categorical variable, where missing data are coded as `NA`, or a state sequence object created using the seqdef function.

`var` A vector specifying the columns of the dataset that contain the trajectories. Default is `NULL`, meaning all columns are used.

`void.miss` Logical, if `TRUE`, treats void elements as missing values. This argument applies only to state sequence objects created with seqdef. Note that the default behavior of seqdef is to treat missing data at the end of sequences as void elements.

`...` 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

This function plots all patterns of missing data within sequences, based on the seqIplot function.

Usage

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

Arguments

`data` Either a data frame containing sequences of a categorical variable, where missing data are coded as `NA`, or a state sequence object created using the seqdef function.

`var` A vector specifying the columns of the dataset that contain the trajectories. Default is `NULL`, meaning all columns are used.

`with.complete` Logical, if `TRUE`, complete trajectories will be included in the plot.

`void.miss` Logical, if `TRUE`, treats void elements as missing values. Applies only to state sequence objects created with seqdef. Note that the default behavior of `seqdef` is to treat missing data at the end of sequences as void elements.

`...` Additional parameters passed to the seqIplot function.

Details

This function uses seqIplot to visualize all patterns of missing data within sequences. For further customization options, refer to the seqIplot documentation.

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