Package: impimp (via r-universe)

October 13, 2024

Type Package

Title Imprecise Imputation for Statistical Matching

Version 0.3.1

Date 2019-02-03

Description Imputing blockwise missing data by imprecise imputation, featuring a domain-based, variable-wise, and case-wise strategy. Furthermore, the estimation of lower and upper bounds for unconditional and conditional probabilities based on the obtained imprecise data is implemented. Additionally, two utility functions are supplied: one to check whether variables in a data set contain set-valued observations; and another to merge two already imprecisely imputed data. The method is described in a technical report by Endres, Fink and Augustin (2018, <doi:10.5282/ubm/epub.42423>).

License GPL-2 | GPL-3

LazyData TRUE

Encoding UTF-8

Imports stats

RoxygenNote 6.1.1

NeedsCompilation no

Author Paul Fink [aut, cre], Eva Endres [aut], Melissa Schmoll [ctb]

Maintainer Paul Fink <paul.fink@stat.uni-muenchen.de>

Date/Publication 2019-02-03 18:43:16 UTC

Repository https://paul-fink.r-universe.dev

RemoteUrl https://github.com/cran/impimp

RemoteRef HEAD

RemoteSha 976176f80b808f5b7c0e88830fd49c5c4fa7295f

Contents

`checkImprecision` *Imprecise Imputation*

Description

Check whether the variables of a data frame contain imprecise observations

Usage

```
checkImprecision(data)
```

Arguments

`data`	data.frame to test to apply the check onto.

Value

A named logical vector of length `ncol(data)`, where `TRUE` indicates that `"|"` is present in the values, which is used to indicate an imprecise observations.

Note

This check is only reliabe for `data`, inheriting class `"impimp"`. If `data` does not inherit class `"impimp"`, the check is tried, but additionaly the user is notified with a warning.

See Also

`impimp`

Examples

```
A <- data.frame(x1 = c(1,0), x2 = c(0,0),
                y1 = c(1,0), y2 = c(2,2))
B <- data.frame(x1 = c(1,1,0), x2 = c(0,0,0),
                z1 = c(0,1,1), z2 = c(0,1,2))
AimpB <- impimp(A, B, method = "variable_wise")
BimpA <- impimp(B, A, method = "variable_wise")
AB <- rbindimpimp(AimpB, BimpA)
```

```
checkImprecision(AB)


data(iris)
checkImprecision(iris) # emits a warning
```

`generateTupelData` *Tuple representation*

Description

Generating a tuple representation of a data.frame with imprecise observations

Usage

```
generateTupelData(data, constraints = NULL)
```

Arguments

`data`	a data.frame object, with potentially imprecise entries; see 'Note'.
`constraints`	a list of so-called logical constraints or fixed zeros. Each element must be an object of class `"impimp_event"`. See 'Details'.

Details

By specifying `constraints` one can exlude combinations of imputed values which are deemed impossible, so called 'logical constraints' or 'fixed zeros'.

Value

A list of length `NROW(data)` of data.frames for the observation within the original data.frame.

Each such data.frame contains the precise observations which are compatible with its imprecise representation.

Note

No sanity check is performed on whether `data` actually contains imprecise observations or is in the form for denoting imprecision throughoutly used in the impimp-package. A warning is triggered if it is not of class `"impimp"`.

See Also

`impimp`, `impimp_event` for sepcifying the constraints

Examples

```
A <- data.frame(x1 = c(1,0), x2 = c(0,0),
                y1 = c(1,0), y2 = c(2,2))
B <- data.frame(x1 = c(1,1,0), x2 = c(0,0,0),
                z1 = c(0,1,1), z2 = c(0,1,2))
AimpB <- impimp(A, B, method = "domain")

## no constraints
generateTupelData(AimpB)

## (y1,z1) = (0,0) as constraint
generateTupelData(AimpB, list(impimp_event(y1 = 0, z1 = 0)))


data(iris)
generateTupelData(iris) # emits a warning
```

`impest` *Imprecise Estimation*

Description

Estimate the probability of some events based on data obtained by imprecise imputation

Usage

```
impest(data, event, constraints = NULL)
```

Arguments

`data`	a data.frame obtained as result from an imprecise imputation e.g. by a call to `impimp`.
`event`	a list of objects of class `"impimp_event"`, specifiying the event of interest. See 'Details'.
`constraints`	a list of so-called logical constraints or fixed zeros. Each element must be an object of class `"impimp_event"`. See 'Details' .

Details

`event` should be a list of objects of class `"impmp_event"`, where the set union of impimp_events is the actual event of interest.

By specifying `constraints` one can exlude combinations of imputed values which are deemed impossible, so called 'logical constraints' or 'fixed zeros'. `constraints` should be a list of objects of class `"impimp_event"`.

An object of class `"impimp_event"` is obtained as a result of a call to `impimp_event`.

For both `event` and `constraints` holds that overlapping in the resulting events generated by the individual impimp_events does not have any side effects, besides a potential decrease in performance.

Value

A numeric vector of length 2, where the first component contains the lower and the second component the upper probability of the event of interest.

References

Endres, E., Fink, P. and Augustin, T. (2018), Imprecise Imputation: A Nonparametric Micro Approach Reflecting the Natural Uncertainty of Statistical Matching with Categorical Data, *Department of Statistics (LMU Munich): Technical Reports*, No. 214

See Also

`impimp`, `impimp_event` for sepcifying constraints and events; `impestcond` for the estimation of conditional probabilities

Examples

```
A <- data.frame(x1 = c(1,0), x2 = c(0,0),
                y1 = c(1,0), y2 = c(2,2))
B <- data.frame(x1 = c(1,1,0), x2 = c(0,0,0),
                z1 = c(0,1,1), z2 = c(0,1,2))
AimpB <- impimp(A, B, method = "variable_wise")
BimpA <- impimp(B, A, method = "variable_wise")
AB <- rbindimpimp(AimpB, BimpA)

## P(Z1=1, Z2=0)
myevent1 <- list(impimp_event(z1 = 1, z2 = 0))
impest(AB, event = myevent1)

## P[(Z1,Z2) in {(1,0),(0,1),(1,1)}]
myevent2 <- list(impimp_event(z1 = 1,z2 = 0),
                 impimp_event(z1 = c(0,1), z2 = 1))
impest(AB, event = myevent2)
```

`impestcond` *Conditional Imprecise Estimation*

Description

Estimate conditional probability of some events based on data obtained by imprecise imputation

Usage

```
impestcond(data, event, condition, constraints = NULL)
```

Arguments

`data`	a data.frame obtained as result from an imprecise imputation e.g. by a call to `impimp`.
`event`	a list of objects of class `"impimp_event"`, specifiying the event of interest. See 'Details'.
`condition`	a list of objects of class `"impimp_event"`, specifiying the event to condition on. See 'Details'.
`constraints`	a list of so-called logical constraints or fixed zeros. Each element must be an object of class `"impimp_event"`. See 'Details' .

Details

`event` and `condition` should each be a list of objects of class `"impmp_event"`, where within each list the set union of impimp_events is the actual event of interest or conditioning event, respectively.

By specifying `constraints` one can exlude combinations of imputed values which are deemed impossible, so called 'logical constraints' or 'fixed zeros'. `constraints` should be a list of objects of class `"impimp_event"`.

An object of class `"impimp_event"` is obtained as a result of a call to `impimp_event`.

For `event`, `condition` and `constraints` holds that overlapping in the resulting events generated by the individual impimp_events does not have any side effects, besides a potential decrease in performance.

Value

A numeric vector of length 2, where the first component contains the lower and the second component the upper conditional probability of the event of interest.

References

Dubois, D. and Prade, H. (1992), Evidence, knowledge, and belief functions, *International Journal of Approximate Reasoning* **6**(3), 295–319.

See Also

`impimp`, `impimp_event` for sepcifying constraints and events; `impest` for the estimation of unconditional probabilities

Examples

```
A <- data.frame(x1 = c(1,0), x2 = c(0,0),
                y1 = c(1,0), y2 = c(2,2))
B <- data.frame(x1 = c(1,1,0), x2 = c(0,0,0),
                z1 = c(0,1,1), z2 = c(0,1,2))
AimpB <- impimp(A, B, method = "domain")
BimpA <- impimp(B, A, method = "domain")
AB <- rbindimpimp(AimpB, BimpA)

myevent <- list(impimp_event(z1 = 1,z2 = 0),
```

```
                impimp_event(z1 = c(0,1), z2 = 1))
cond <- list(impimp_event(x1 = 1))

impestcond(AB, event = myevent, condition = cond)

constr <- list(impimp_event(y1 = 0, z1 = 0))
impestcond(AB, event = myevent, condition = cond,
           constraints = constr)
```

---

impimp *Imprecise Imputation for Statistical Matching*

Description

Impute a data frame imprecisely

Usage

```
impimp(recipient, donor, method = c("variable_wise", "case_wise",
  "domain"), matchvars = NULL, vardomains = NULL)

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

is.impimp(z)
```

Arguments

recipient	a data.frame acting as recipient; see details.
donor	a data.frame acting as donor; see details.
method	1-character string of the desired imputation method. The following values are possible, see details for an explanantion: "variable_wise" (default), "case_wise" and "domain".
matchvars	a character vector containing the variable names to be used as matching variables. If NULL (default) all variables, present in both donor and recipient are used as matching variables.
vardomains	a named list containing the possible values of all variable in donor that are not present in recipient. If set to NULL (default) the list is generated by first coercing all those variables to type factor and then storing their levels.
x	object of class 'impimp'
...	further arguments passed down to print.data.frame
z	object to test for class "impimp"

Details

As in the context of statistical matching the data.frames `recipient` and `donor` are assumed to contain an overlapping set of variables.

The missing values in `recipient` are subsituted with observed values in `donor` for approaches based on donation classes and otherwise with the set of all possible values for the variable in question.

For `method = "domain"` a missing value of a variable in `recipient` is imputed by the set of all possible values of that variable.

The other methods are based on donation classes which are formed based on the matching variables whose names are provided by `matchvars`. They need to be present in both `recipient` and `donor`: For `method = "variable_wise"` a missing value of a variable in `recipient` is imputed by the set of all observed values of that variable in `donor`. For `method = "case_wise"` the variables only present in `donor` are represented as tuples. A missing tuple in `recipient` is then imputed by the set of all observed tuples in `donor`.

Value

The data.frame resulting in an imprecise imputation of `donor` into `recipient`. It is also of class `"impimp"` and stores the imputation method in its attribute `"impmethod"`, the names of the variables of the resulting object containing imputed values in the attribute `"imputedvarnames"`, as well as the list of (guessed) levels of each underlying variable in `"varlevels"`.

Reserved characters

The variable names and observations in `recipient` and `donor` must not contain characters that are reserved for internal purpose. The actual characters that are internally used are stored in the options `options("impimp.obssep")` and `options("impimp.varssep")`. The former is used to separate the values of a set-valued observation, while the other is used for a concise tupel representation.

Note

This method does not require that all variables in `recipient` and `donor` are `factor` variables, however, the imputation methods apply coercion to factor, so purely numerical variables will be treated as factors eventually. It does assume (and test for it) that there are no missing values present in the matching variables.

References

Endres, E., Fink, P. and Augustin, T. (2018), Imprecise Imputation: A Nonparametric Micro Approach Reflecting the Natural Uncertainty of Statistical Matching with Categorical Data, *Department of Statistics (LMU Munich): Technical Reports*, No. 214. URL https://epub.ub.uni-muenchen.de/42423/.

See Also

for the estimation of probabilities `impest` and `impestcond`; `rbindimpimp` for joining two `impimp` objects

Examples

```
A <- data.frame(x1 = c(1,0), x2 = c(0,0),
                y1 = c(1,0), y2 = c(2,2))
B <- data.frame(x1 = c(1,1,0), x2 = c(0,0,0),
                z1 = c(0,1,1), z2 = c(0,1,2))
impimp(A, B, method = "variable_wise")

## Specifically setting the possible levels of 'z1'
impimp(A, B, method = "domain", vardomains = list(z1 = c(0:5)))
```

`impimp_event` *Imprecise Events*

Description

Helper function to allow the generation of a set of events as cartesian product.

Usage

```
impimp_event(..., isEventList = FALSE)

is.impimp_event(x)
```

Arguments

`...`	these arguments are of the form `varname = value`. For each component the varname should be a variable name from the underlying data.frame and value a vector of possible outcomes; may also be of length one.
`isEventList`	logical; if `TRUE` and `...` contains only a list object, this list is treated as if it was an event specification, see `....` Since this argument follows `...` its name cannot be abbreviated.
`x`	object to test for class `"impimp_event"`

Value

A object of class `"impimp_event"` as a list of lists, where each sublist contains one point in the cartesian product, spanned by the input values and variables.

Note

There is no plausibility check on whether the supplied varnames are actually contained in the data.frame for which the resulting impimp_event object is later used for.

See Also

`impest`, `impestcond`

Examples

```
## underlying data set: x1: 1:6, x2: 1:10

## subspace, requiring: x1 == 1 & ((x2 == 1 ) | (x2 == 2))
impimp_event(x1 = 1, x2 = c(1,2))

## subsapce containing all points whitin the Cartesian
## product of (x1 =) {1,2,3,6} x {5,8} (= x2)
# via  ... argument
impimp_event(x1 = c(1:3,6), x2 = c(5,8))
# via EVENTLIST
impimp_event(list(x1 = c(1:3,6), x2 = c(5,8)),
             isEventList = TRUE)
```

`rbindimpimp` *Combine impimp Objects*

Description

Combine two object of class `"impimp"` like `rbind` would do with data frames.

Usage

```
rbindimpimp(x, y)
```

Arguments

`x, y`	objects of class `"impimp"`. As such may contain variables in form of tuples, they are not required to have the same number of variables as returned from `ncol`. However, they are required to have the same underlying variables. If that condition is not satisfied an error is raised.

Details

The resulting object is constructed in such a way that minimizes the creation of 'tupled' variables. Only those variables are joined as tuples which are actually necessary to keep the data frame like consise representation of impimp objects.

The attributes `"impmethod"` and `"varlevels"` contain the set union of those of `x` and `y` on a global and per underlying variable basis, respectively.

Value

An object of class `"impimp"`, inheriting the attributes, specific to imimp objects, of `x` and `y`.

See Also

`impimp`

Examples

```
A <- data.frame(x1 = c(1,0), x2 = c(0,0),
                y1 = c(1,0), y2 = c(2,2))
B <- data.frame(x1 = c(1,1,0), x2 = c(0,0,0),
                z1 = c(0,1,1), z2 = c(0,1,2))
impA <- impimp(A, B, method = "case_wise")
impB <- impimp(B, A, method = "case_wise")
rbindimpimp(impA, impB)
```

Index