Package 'miceExt'

February 9, 2018

Title Extension Package to 'mice'

Version 1.0.0

Maintainer Tobias Schumacher <tobias.schumacher1@rwth-aachen.de>

Description Extends and builds on the 'mice' package by
adding a functionality to perform multivariate predictive mean matching on
imputed data as well as new functionalities to perform predictive mean
matching on factor variables.

Depends R (>= 3.3), mice (>= 2.46.0)

Imports RANN (>= 2.5.1), RANN.L1 (>= 2.5)

License GPL-2

Author Tobias Schumacher [aut, cre],
Philipp Gaffert [aut],
Stef van Buuren [ctb],
Karin Groothuis-Oudshoorn [ctb]

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests testthat

NeedsCompilation no

Repository CRAN

Date/Publication 2018-02-09 11:12:24 UTC

R topics documented:

boys_data *Growth of Dutch boys*

Description

Modified sample of the boys data set that is included in mice.

Format

A dataset with 100 observations of the following 9 variables:

age Decimal age (0-21 years)

hgt Height (cm)

wgt Weight (kg)

bmi Body mass index

hc Head circumference (cm)

gen Genital Tanner stage (G1-G5)

phb Pubic hair (Tanner P1-P6)

tv Testicular volume (ml)

reg Region (north, east, west, south, city)

Details

The original data set boys that is included in mice is a random sample of 10% from the cross-sectional data used to construct the Dutch growth references 1997, and contains information height, weight, head circumference and puberty of 748 Dutch boys. From this data, a sample of 100 rows has been taken and modified such that in each row, the values in the columns hc, gen and phb are either blockwise NA or blockwise non-NA.

Source

Fredriks, A.M,, van Buuren, S., Burgmeijer, R.J., Meulmeester JF, Beuker, R.J., Brugman, E., Roede, M.J., Verloove-Vanhorick, S.P., Wit, J.M. (2000) Continuing positive secular growth change in The Netherlands 1955-1997. *Pediatric Research*, **47**, 316-323.

Fredriks, A.M., van Buuren, S., Wit, J.M., Verloove-Vanhorick, S.P. (2000). Body index measurements in 1996-7 compared with 1980. *Archives of Disease in Childhood*, **82**, 107-112.-734.
'@keywords datasets

See Also

boys

mammal_data *Mammal sleep data*

Description

Modified version of the mammal sleep data set that is included in mice.

Format

A dataset with 62 observations of the following 11 variables:

species Species of animal

bw Body weight (kg)

brw Brain weight (g)

sws Slow wave ("nondreaming") sleep (hrs/day)

ps Paradoxical ("dreaming") sleep (hrs/day)

ts Total sleep (hrs/day) (sum of slow wave and paradoxical sleep)

mls Maximum life span (years)

gt Gestation time (days)

pi Predation index (1-5), 1 = least likely to be preyed upon

sei Sleep exposure index (1-5), 1 = least exposed (e.g. animal sleeps in a well-protected den), 5 = most exposed

odi Overall danger index (1-5) based on the above two indices and other information, 1 = least danger (from other animals), 5 = most danger (from other animals)

Details

The original dataset was from a study by Allison and Cicchetti (1976) of 62 mammal species on the interrelationship between sleep, ecological, and constitutional variables, and was adapted in the mice-package. It contains missing values on five variables and has been modified such that for each row, the entries in the column tuples (sws,ps) and (mls,gt) are either pairwiseNA or pairwise non-NA.

Source

Allison, T., Cicchetti, D.V. (1976). Sleep in Mammals: Ecological and Constitutional Correlates. Science, 194(4266), 732-734.

See Also

mammalsleep

`mice.binarize` *Binarize Factor Columns in Data Frames*

Description

This function replaces factor columns in data frames in-place by a set of binary columns which represent the so-called one-hot encoding of this factor. More precisely, a column of a factor with `n` levels will be transformed into a set of `n` binary columns, each representing exactly one category of the original factor. Hence, the value `1` occurs in a column if and only if the original factor had the value corresponding to that column.
Further, this function also returns a predictor matrix that fits to the binarized data frame and, when used as input parameter in `mice()`, ensures that binary columns that relate to the same factor do not predict each other.

Usage

```
mice.binarize(data, include_ordered = TRUE, include_observed = FALSE,
  cols = NULL, pred_matrix = (1 - diag(1, ncol(data))))
```

Arguments

`data`	Matrix or data frame that contains factor columns which we want to convert into an equivalent set of binary columns.
`include_ordered`	Logical variable indicating whether we also want to transform ordered factors. Default is `TRUE`.
`include_observed`	Logical variable indicating whether we also want to transform factor columns in which all values are observed. Default is `FALSE`.
`cols`	Numerical vector corresponding to the indices or character vector corresponding to the names of factor columns which we want to transform. By default, its value is `NULL`, indicating that the algorithm automatically identifies all factor columns that are to be binarized. If however the user specifies its value, the function exclusively transforms the specified columns, ignoring the values of the other optional parameters.
`pred_matrix`	A custom predictor matrix relating to input `data`, which will get transformed into the format that fits to the binarized output data frame. The result of this transformation will be stored in the `pred_matrix`-element of the output and should then be used as the `predictorMatrix` parameter in `mice()` to ensure that binary columns relating to the same factor column in the original data do not predict each other, yielding cleaner imputation models. If not specified, the default is the massive imputation predictor matrix.

Value

List containing the following three elements:

data The binarized data frame.

par_list A list containing the original data frame as well as some parameters with further information on the transformation. This list is needed to retransform the (possibly imputed) data at later stager via the `mice.factorize()` function, and should not be edited by the user under any circumstance. Next to the original data, the most notable element of this list would be "dummy_cols", which itself is a list of the column tuples that correspond to the transformed factor column from the original data set, and therefore works perfectly as input for `mice.post.matching()` (cf. examples below).

pred_matrix Transformed version of input `pred_matrix` that should be used as the input argument `predictorMatrix` of `mice()`.

Author(s)

Tobias Schumacher, Philipp Gaffert

See Also

`mice.factorize`, `mice.post.matching`, `mice`

Examples

```
#-------------------------------------------------------------------------------
# first set of examples illustrating basic functionality
#-------------------------------------------------------------------------------

# binarize all factor columns in boys_data that contain NAs
boys_bin <- mice.binarize(boys_data)


# binarize only column 'gen' in boys_data
boys_bin <- mice.binarize(boys_data, cols = c("gen"))

# read out binarized data
boys_bin$data

## Not run:
#-------------------------------------------------------------------------------
# this example illustrates the combined functionalities of mice.binarize,
# mice.factorize and mice.post.matching on the dataset 'boys' from mice, which
# yields different imputations on the factor columns 'gen', 'phb' and 'reg'
# than mice() would output
#-------------------------------------------------------------------------------

# binarize all factor columns in boys_data that contain NAs
boys_bin <- mice.binarize(boys)

# run mice on binarized data, note that we need to use boys_bin$data to grab
```

```
# the actual binarized data and that we use the output predictor matrix
# boys_bin$pred_matrix which is recommended for obtaining better imputation
# models
mids_boys <- mice(boys_bin$data, predictorMatrix = boys_bin$pred_matrix)

# it is very likely that mice imputed multiple ones among one set of dummy
# variables, so we need to post-process
post_boys <- mice.post.matching(mids_boys, distmetric = "residual")

# now we can safely retransform to the original data, with non-binarized
# imputations
res_boys <- mice.factorize(post_boys$midsobj, boys_bin$par_list)

# analyze the distribution of imputed variables, e.g. of the column 'gen',
# using the mice version of with()
with(res_boys, table(gen))

## End(Not run)
```

`mice.factorize` *Transform Imputations of Binarized Data Into Their Corresponding Factors*

Description

This function acts as the counterpart to `mice.binarize`, as it effectively retransforms imputations of binarized data that `mice` has been run on and that has been post-processed via `mice.post.matching` after. The post-processing is usually necessary as `mice` is very likely to impute multiple ones among the dummy columns belonging to to a single factor entry. The resulting `mice::mids` object is not suited for further `mice.mids()` iterations or the use of `plot`, but works well as input to `with()`.

Usage

```
mice.factorize(obj, par_list)
```

Arguments

`obj`	`mice::mids` object resulting from a call of `mice.post.matching()` and whose underlying data frame results from a call of `mice::binarize()`.
`par_list`	List that has been returned in a previous call of `mice::binarize()` next to the underlying data of the argument `obj`.

Value

A `mice::mids` object in which data and imputations have been retransformed from their respective binarized versions in the input `obj`. As this isn't a proper result of a mice iteration and many of the attributes of `obj` cannot be transformed well, only the slots `data`, `nmis`, `where` and `imp`, which are needed in `with()` are not `NULL`. Hence, it does not work as input for `mice.mids()`.

Author(s)

Tobias Schumacher, Philipp Gaffert

See Also

mice.binarize, mice.post.matching, mice

Examples

```
## Not run:
#-------------------------------------------------------------------------------
# this example illustrates the combined functionalities of mice.binarize,
# mice.factorize and mice.post.matching on the dataset 'boys' from mice, which
# yields different imputations on the factor columns 'gen', 'phb' and 'reg'
# than mice() would output
#-------------------------------------------------------------------------------

# binarize all factor columns in boys_data that contain NAs
boys_bin <- mice.binarize(boys)

# run mice on binarized data, note that we need to use boys_bin$data to grab
# the actual binarized data and that we use the output predictor matrix
# boys_bin$pred_matrix which is recommended for obtaining better imputation
# models
mids_boys <- mice(boys_bin$data, predictorMatrix = boys_bin$pred_matrix)

# it is very likely that mice imputed multiple ones among one set of dummy
# variables, so we need to post-process
post_boys <- mice.post.matching(mids_boys, distmetric = "residual")

# now we can safely retransform to the original data, with non-binarized imputations
res_boys <- mice.factorize(post_boys$midsobj, boys_bin$par_list)

# analyze the distribution of imputed variables, e.g. of the column 'gen',
# using the mice version of with()
with(res_boys, table(gen))

## End(Not run)
```

`mice.post.matching` *Post-processing of Imputed Data by Multivariate Predictive Mean Matching*

Description

Performs multivariate predictive mean matching (PMM) on a set of columns that have been imputed on by the functionalities of the mice-package. Also offers functionality to match imputations against observed variables.

Usage

```
mice.post.matching(obj, cols = NULL, donors = 5L, weights_list = NULL,
  distmetric = "residual", matchtype = 1L, match_vars = NULL,
  ridge = 1e-05, eps = 1e-04, maxcor = 0.99)
```

Arguments

obj	mice::mids object that has been returned by a previous run of mice() or mice.mids() and whose imputations we want to post-process.
cols	Column tuple or list of column tuples that multivariate PMM has to be performed on. Each column tuple has to be represented by an atomic vector that can contain column names as character strings or column indices in numerical format. The column list must not contain any duplicates, and each column in the list has to be included in the visitSequence of the given mids object. The imputation method that was applied on each column has to be either pmm or norm. Univariate tuples are also allowed if they have been imputed on via mice.impute.norm(), or if we want to match them against on observed variable as specified in parameter match_vars. The default is cols = NULL, in which case this function automatically looks for tuples that have identical missing value patterns and imputes on those.
donors	Integer indicating the size of the donor pool among which a draw in the matching step is made. The default is donors = 5L. Setting donors = 1L always selects the closest match (nearest neighbor), but is not recommended.
weights_list	List of numeric vectors that allocates weights to the elements in each tuple in cols, giving us the possibility to punish or mitigate differences in certain columns of the data when computing the distances in the matching step to determine the donor pool. Hence, this list has to be the same length as cols, and each element has to be the same length as the corresponding column tuple. If we do not want to apply weights to a single tuple, we can write a single 0, 1 or NULL in the corresponding spot of the list. The default is weights_list = NULL, meaning that no weights should be applied to any columns at all.
distmetric	Character string that determines which mathematical metric we want to use to compute the distances between the multivariate y_obs and y_mis. Options are "euclidian", "manhattan", "mahalanobis" and "residual". The first three options refer to the distance metrics of the same name, the latter is a variant of the mahalanobis distance in which we consider the residual covariance cov(y_hat_obs - y_obs) of the predicted model. This distance has been proposed and recommended by Little (1988) and is also the default. Note that the first two options are faster though, as they do not involve any matrix computations.

`matchtype` Integer indicating the type of matching distance to be used in PMM. The default choice (`matchtype = 1L`) calculates the distance between the *predicted* values of `y_obs` and the *drawn* values of `y_mis` (called type-1 matching). Other choices are `matchtype = 0L` distance between predicted values) and `matchtype = 2L` (distance between drawn values).

`match_vars` Vector specifying for each tuple which additional variable has to be matched against. Can be an integer or character vector, either specifying column indices or column names. `match_vars` must be the same length as `cols` with `0`-elements or `""`-elements to disable this functionality for certain groups. Default is `match_vars = NULL`, which completely disables this functionality.

`ridge` The ridge penalty used in an internal call of `mice:::.norm.draw()` to prevent problems with multicollinearity. The default is `ridge = 1e-05`, which means that 0.01 percent of the diagonal is added to the cross-product. Larger ridges may result in more biased estimates. For highly noisy data (e.g. many junk variables), set `ridge = 1e-06` or even lower to reduce bias. For highly collinear data, set `ridge = 1e-04` or higher.

`eps` The minimum variance that we allow predictors to have when building a linear model to compute the `y_hat`-values. More precisely, this parameter is used in an internal call of `mice:::remove.lindep()`. The default is `eps = 1e-04`.

`maxcor` The maximum correlation that we allow predictors to have when building a linear model to compute the `y_hat`-values. More precisely, this parameter is used in an internal call of `mice:::remove.lindep()`.The default is `maxcor = 0.99`.

Details

The algorithm basically iterates over the `m` imputations of the input `mice::mids` object and each column tuple in `cols`, each time following the following two main steps:

Prediction First, we iterate over the columns of the current tuple, collecting the complete column y in which we want to impute, along with the matrix x of its predictors. Among the values of y, we identify the observed values y_{obs} and the missing values `y_{mis}` along with their corresponting predictors x_{obs} and `x_{mis}`, restricting ourselves to only those values whose predictors are complete, i.e. don't contain any missing values themselves. Note that among all the columns in the current tuple, there may be no common predictors in which case the function breaks.

From the observed values and their predictors, we build a Bayesian linear regression model and, depending on the specified matching type, use the model's coefficients and drawn regression weights β to compute the predicted means $\hat{y}_{obs}$ and $\hat{y}_{mis}$ which we then save in a matrix $\hat{Y}$.

Matching After iterating over the columns in the current tuple and building the matrix $\hat{Y}$, we match the multivariate predictions of the missing values in $\hat{Y}$ against the predictions of the observed values. More precisely, for each $\hat{y}_{mis}$, we perform a k-nearest-neighbor search among all values $\hat{y}_{obs}$, where k is the number of donors specified by the user, and then randomly sample one element of those k nearest neighbors which is then used to impute the missing value tuple in y. The distance metric that is used to determine the nearest neighbors is specified by the user as well, and in case of Euclidian or Manhattan distance its computation is very straightforward. If the Mahalanobis distance or the residual distance (as proposed by Little) has been selected, we first compute the corresponding covariance matrix and its (pseudo) inverse via its eigen

decomposition, and then use it to transform the values $\hat{y}_{mis}$ and $\hat{y}_{obs}$ that are fed into the nearest-neighbor search. If weights have been specified as well, they are also applied before the kNN-search.
If an additional observed variable to match against has been specified via `match_vars`, the set of predictors and missing values are partitioned by the values of the external variable first, and then the matching is performed within each pair of corresponding subsets of that partition.

Note that the imputed values are only stored as a result and, other than in the `mice`-algorithm, are not used to compute predictive means for any other missing value. We exclusively use the imputed values that are provided within the input `mids`-object.

Value

List containing the following two elements:

midsobj `mice::mids` object that differs from the input object only in the imputations that have been post-processed, and the `call` and `loggedEvents` attributes that have been updated. In particular, those post-processed imputations are not affecting the `chainMean` or `chainVar`-attributes, and hence, `plot()` will not consider them either.

cols List of column tuples that multivariate imputation has been performed on. It is equal to the input parameter `cols` if it has been specified by the user, otherwise those column tuples have been determined internally.

Author(s)

Tobias Schumacher, Philipp Gaffert

References

Little, R.J.A. (1988), Missing data adjustments in large surveys (with discussion), Journal of Business Economics and Statistics, 6, 287–301.

Van Buuren, S. (2012). Flexible Imputation of Missing Data. CRC/Chapman \& Hall, Boca Raton, FL.

Van Buuren, S., Groothuis-Oudshoorn, K. (2011). `mice`: Multivariate Imputation by Chained Equations in `R`. *Journal of Statistical Software*, **45**(3), 1-67. http://www.jstatsoft.org/v45/i03/

See Also

`mice`, `mids-class`, `mice.binarize`, `mice.factorize`

Examples

```
## Not run:
#-------------------------------------------------------------------------------
# example on modified 'mammalsleep' data set from mice, that has identical
# missing data patterns on the column tuples ('ps','sws') and ('mls','gt')
#-------------------------------------------------------------------------------

# run mice on data set 'mammal_data' and obtain a mids object to post-process
mids_mammal <- mice(mammal_data)
```

```
# run function, as cols has not been specified, it will automatically detect
# the column tuples with identical missing data patterns and then impute on
# these
post_mammal <- mice.post.matching(mids_mammal)

#read out which column tuples have been imputed on
post_mammal$cols

#look into imputations within resulting mice::mids object
post_mammal$midsobj$imp


#-------------------------------------------------------------------------------
# example on original 'mammalsleep' data set from mice, in which we
# want to post-process the imputations in column 'sws' by only imputing values
# from rows whose value in 'pi' matches the value of 'pi' in the row we impute
# on.
#-------------------------------------------------------------------------------

# run mice on data set 'mammal_data' and obtain a mids object to post-process
mids_mammal <- mice(mammalsleep)


# run function, specify 'sws' as the column to impute on, and specify 'pi' as
# the observed variable to consider in the matching.
post_mammal <- mice.post.matching(mids_mammal, cols = "sws", match_vars = "pi")


#look into imputations within resulting mice::mids object
post_mammal$midsobj$imp


#-------------------------------------------------------------------------------
# example working on 'boys_data', illustrating some more
#   advanced functionalities
#-------------------------------------------------------------------------------

# run mice() first
mids_boys <- mice(boys_data)

# in boys_data, the columns 'hgt' and 'bmi' have the same NA-pattern,
# and they have both been imputed on by PMM within mice()
# hence, we can perform our post-processing on those columns:
cols <- c("hgt","bmi")


# run post-processing, here with some non-default parameters:
# -> a reduced donor pool of only 3 donors is chosen
# -> weights_list: we want to punish bigger differnces in bmi within the
#    matching, so we apply weight of 3 to this column while leaving the
#    other column as is
```

```
# -> distmetric: within our matching process, we want to take the variances of
#    each column into account by normaizing the column values over their
#    variance. This is done by using the Mahalanobis-metric.

post_boys <- mice.post.matching(mids_boys,
                                cols = cols,
                                donors = 3L,
                                weights_list = c(1, 3),
                                distmetric = "mahalanobis")


#-------------------------------------------------------------------------------
# example that illustrates the combined functionalities of mice.binarize,
# mice.factorize and mice.post.matching on the dataset 'boys' from mice, to
# impute the factor columns 'gen', 'phb' and 'reg'.
#-------------------------------------------------------------------------------

# binarize all factor columns in boys_data that contain NAs
boys_bin <- mice.binarize(boys)

# run mice on binarized data, note that we need to use boys_bin$data to grab
# the actual binarized data and that we use the output predictor matrix
# boys_bin$pred_matrix which is recommended for obtaining better imputation
# models
mids_boys <- mice(boys_bin$data, predictorMatrix = boys_bin$pred_matrix)

# it is very likely that mice imputed multiple ones among one set of dummy
# variables, so we need to post-process
post_boys <- mice.post.matching(mids_boys, distmetric = "residual")

# now we can safely retransform to the original data, with non-binarized
# imputations
res_boys <- mice.factorize(post_boys$midsobj, boys_bin$par_list)

# analyze the distribution of imputed variables, e.g. of the column 'gen',
# using the mice version of with()
with(res_boys, table(gen))


#-------------------------------------------------------------------------------
# similar example to the previous working on 'boys_data' again, in which the
# factor columns  'gen' and 'phb' have the same missing data pattern
#-------------------------------------------------------------------------------

# binarize
boys_bin <- mice.binarize(boys_data, cols = c("gen", "phb"))

# run mice on binarized data
mids_boys <- mice(boys_bin$data, predictorMatrix = boys_bin$pred_matrix)

# again we want to post-process the binarized columns. As the binarized columns
# of 'gen' and 'phb' share the same missing data pattern, they would, by default,
# be imputed as one big tuple, as this is how they are detected internally.
```

```
# If we want to post-process the binary columns of 'gen' and 'phb' separately, we
# can use the element 'dummy_cols' from 'boys_bin$par_list' which was generated
# in the binarization, and feed it into the 'cols'-argument.
post_boys <- mice.post.matching(mids_boys, cols = boys_bin$par_list$dummy_cols)

# retransform to the original format
res_boys <- mice.factorize(post_boys$midsobj, boys_bin$par_list)

## End(Not run)
```

`miceExt` *miceExt: Extension Package to mice*

Description

This package extends and builds on the `mice` package by adding a functionality to perform multivariate predictive mean matching on imputed data as well as new functionalities to perform predictive mean matching on factor variables.

Details

The `mice` package, which was implemented and published by Stef van Buuren and Karin Groothuis-Oudshoorn in 2001 and has been further developed ever since, is one of most extensive and most commonly used implementations of multiple imputation within R. Despite its many years of refinement however, there are still some missing data problems that mice does not handle very well, and two of these have now been addressed within the implementation of this package.
First, mice does not provide any option to perform imputation on multiple columns at once, which can, for instance, result in nonsensical output imputations when there are causal relationships between the corresponding attributes, e.g. a 15-year-old person that has a driver's license.
Further, mice still struggles with imputing categorical data, as many internally used imputation methods either are not suited for this kind of data at all or do not necessarily converge to the optimal solution.
Overall, `miceExt` provides three functions, namely

1. `mice.post.matching()`,
2. `mice.binarize()`,
3. `mice.factorize()`,

out of which the first function post-processes results of the `mice()`-algorithm by performing multivariate predictive mean matching on a user-defined set of column tuples, and results in imputations that are always equal to already-observed values, which annihilates the chance of getting unrealistic output values.
The latter two functions tackle the second issue by even extending the functionality of `mice.post.matching()`. The function `mice.binarize()` transforms categorical attributes of a given data frame into a binary dummy representation, which results in an exclusively numerical data set that mice can handle well. Inconsistencies within the imputed dummy columns can then be handled by `mice.post.matching()`, and `mice.factorize()` finally serves the purpose of retransforming the imputed binary data into the corresponding original categories, resulting in a proper imputation of the given categorical data.

Author(s)

Tobias Schumacher, Philipp Gaffert, Stef van Buuren, Karin Groothuis-Oudshoorn

See Also

`mice.post.matching`, `mice.binarize`, `mice.factorize`, `mice`

Index