Package ‘NADIA’

November 10, 2020

Type Package

Title NA Data Imputation Algorithms

Version 0.4.0

Author Jan Borowski, Piotr Fic

Maintainer Jan Borowski <janborowka7@gmail.com>

Description
Package crate uniform interface for several advanced imputations missing data methods. Every available method can be used as a part of mlr3 pipelines whats allow easy tuning and performance evaluation. Most of the used function work separately on train and test sets (imputation is trained on the training set and impute train data, after that imputation is again trained on test set and impute test data). This approach makes training imputation parameters unprofitable.

License GPL

Depends mlr3, mlr3pipelines, paradox

Imports mlr3learners, missForest, missMDA, doParallel, testthat, Amelia, VIM, softImpute, missRanger, methods, mice, data.table, foreach, glmnet

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1.9000

Suggests knitr, rmarkdown

VignetteBuilder knitr

R topics documented:

autotune_Amelia 2
autotune_mice 4
autotune_missForest 6
autotune_missRanger 9
autotune_softImpute 10
autotune_VIM_hotdeck 12

autotune_VIM_Irmi 14
autotune_VIM_kNN 15
autotune_VIM_regrImp 17
fetch_data 19
formula_creating 19
mice.reuse 20
mids.append 21
missMDA_FMAD_MCA_PCA 22
missMDA_MFA 24
PipeOpAmelia 25
PipeOpHist_B 27
PipeOpMean_B 29
PipeOpMedian_B 30
PipeOpMice 31
PipeOpMice_A 33
PipeOpmissForest 35
PipeOpmissMDA_MFA 37
PipeOpmissMDA_PCA_MCA_FMAD 38
PipeOpmissRanger 40
PipeOpMode_B 42
PipeOpOOR_B 43
PipeOpSample_B 44
PipeOpSimulateMissings 45
PipeOpSoftImpute 47
PipeOpVIM_HD 48
PipeOpVIM_IRMI 50
PipeOpVIM_kNN 51
PipeOpVIM_regrImp 53
random_param_mice_search 54
replace_overimputes 55
simulate_missings 56

Index **58**

`autotune_Amelia` *Perform imputation using Amelia package and EMB algorithm.*

Description

Function use EMB (Expectation-Maximization with Bootstrapping) to impute missing data. Function performance is highly depend from data structure and chosen parameters.

Usage

```
autotune_Amelia(
  df,
  col_type,
  percent_of_missing,
  col_0_1 = FALSE,
  parallel = TRUE,
  polytime = NULL,
  splinetime = NULL,
  intercs = FALSE,
  empir = NULL,
  verbose = FALSE,
  return_one = TRUE,
  m = 3,
  out_file = NULL
)
```

Arguments

`df`	data.frame. Df to impute with column names and without target column.
`col_type`	character vector. Vector containing column type names.
`percent_of_missing`	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
`col_0_1`	Decaid if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False. (Works only for returning one dataset).
`parallel`	If true parallel calculation is used.
`polytime`	parameter pass to amelia function
`splinetime`	parameter pass to amelia finction
`intercs`	parameter pass to amleia function
`empir`	parameter pass to amelia function as empir in Amelia == empir*nrow(df). If empir dont set empir=nrow(df)*0.015.
`verbose`	If true function will print on console.
`return_one`	Decide if one dataset or amelia object will be returned.
`m`	Number of datasets generated by amelia. If retrun_one=TRUE first dataset will be given.
`out_file`	Output log file location if file already exists log message will be added. If NULL no log will be produced.

Value

Return one data.frame with imputed values or amelia object.

References

James Honaker, Gary King, Matthew Blackwell (2011). Amelia II: A Program for Missing Data. Journal of Statistical Software, 45(7), 1-47. URL http://www.jstatsoft.org/v45/i07/.

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


  imp_data <- autotune_Amelia(raw_data, col_type, percent_of_missing)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

`autotune_mice` *Automatical tuning of parameters and imputation using mice package.*

Description

Function impute missing data using mice functions. First perform random search using linear models (generalized linear models if only categorical values are available). Using glm its problematic. Function allows users to skip optimization in that case but it can lead to errors. Function optimize prediction matrix and method. Other mice parameters like number of sets(m) or max number of iterations(maxit) should be set as hight as possible for best results(higher values are required more time to perform imputation). If u chose to use one inputted dataset m is not important. More information can be found in `random_param_mice_search` and `formula_creating` and `mice`.

Usage

```
autotune_mice(
  df,
  m = 5,
  maxit = 5,
  col_miss,
  col_no_miss,
  col_type,
  set_cor = 0.5,
  set_method = "pmm",
  percent_of_missing,
  low_corr = 0,
  up_corr = 1,
  methods_random = c("pmm"),
  iter,
  random.seed = 123,
  optimize = TRUE,
  correlation = TRUE,
  return_one = TRUE,
  col_0_1 = FALSE,
  verbose = FALSE,
  out_file = NULL
)
```

Arguments

`df`	data frame for imputation.
`m`	number of sets produced by mice.
`maxit`	maximum number of iteration for mice.
`col_miss`	name of columns with missing values.
`col_no_miss`	character vector. Names of columns without NA.
`col_type`	character vector. Vector containing column type names.
`set_cor`	Correlation or fraction of featurs using if optimize= False
`set_method`	Method used if optimize=False. If NULL default method is used (more in methods_random section).
`percent_of_missing`	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
`low_corr`	double betwen 0,1 default 0 lower boundry of correlation set.
`up_corr`	double between 0,1 default 1 upper boundary of correlation set. Both of these parameters work the same for a fraction of features.
`methods_random`	set of methods to chose. Default 'pmm'. If seted on NULL this methods are used predictive mean matching (numeric data) logreg, logistic regression imputation (binary data, factor with 2 levels) polyreg, polytomous regression imputation for unordered categorical data (factor ¿ 2 levels) polr, proportional odds model for (ordered, ¿ 2 levels).

`iter`	number of iteration for randomSearch.
`random.seed`	random seed.
`optimize`	if user wont to optimize.
`correlation`	If True correlation is using if Fales fraction of features. Default True.
`return_one`	One or many imputed sets will be returned. Default True.
`col_0_1`	Decaid if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False. (Works only for returning one dataset).
`verbose`	If FALSE function didn't print on console.
`out_file`	Output log file location if file already exists log message will be added. If NULL no log will be produced.

Value

Return imputed datasets or mids object containing multi imputation datasets.

Examples

```
{
  raw_data <- mice::nhanes2

  col_type <- 1:ncol(raw_data)
  for (i in col_type) {
    col_type[i] <- class(raw_data[, i])
  }

  percent_of_missing <- 1:ncol(raw_data)
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }
  col_no_miss <- colnames(raw_data)[percent_of_missing == 0]
  col_miss <- colnames(raw_data)[percent_of_missing > 0]
  imp_data <- autotune_mice(raw_data, optimize = FALSE, iter = 2,
   col_type = col_type, percent_of_missing = percent_of_missing,
   col_no_miss = col_no_miss, col_miss = col_miss)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

`autotune_missForest` *Perform imputation using missForest form missForest package.*

Description

Function use missForest package for data imputation. OBBerror (more in `autotune_mice`) is used to perform grid search.

Usage

```
autotune_missForest(
  df,
  col_type,
  percent_of_missing,
  cores = NULL,
  ntree_set = c(100, 200, 500, 1000),
  mtry_set = NULL,
  parallel = FALSE,
  col_0_1 = FALSE,
  optimize = TRUE,
  ntree = 100,
  mtry = NULL,
  verbose = FALSE,
  maxiter = 20,
  maxnodes = NULL,
  out_file = NULL
)
```

Arguments

df	data.frame. Df to impute with column names.
col_type	character vector. Vector containing column type names.
percent_of_missing	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
cores	integer. Number of threads used by parallel calculations. By default approximately half of available CPU cores.
ntree_set	integer vector. Vector contains numbers of tree for grid search.
mtry_set	integer vector. Vector contains numbers of variables randomly sampled at each split.
parallel	logical. If TRUE parallel calculation is using.
col_0_1	decide if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False.
optimize	optimize inside function
ntree	ntree from missForest function
mtry	mtry form missforest function
verbose	If FALSE funtion didn't print on console.
maxiter	maxiter form missForest function.
maxnodes	maxnodes from missForest function.
out_file	Output log file location if file already exists log message will be added. If NULL no log will be produced.

Details

Function try to use parallel backend if it's possible. Half of the available cores are used or number pass as cores param. (Number of used cores can't be higher then number of variables in df. If it happened a number of cores will be set at ncol(df)-2 unless this number is ¡= 0 then cores =1). To perform parallel calculation function use registerDoParallel to create parallel backend. Creating backend can have significant time cost so for very small df cores=1 can speed up calculation. After calculation function turns off parallel backend.

Gride search is used to chose a sample for each tree and the number of trees can be turn off. Params in grid search have significant influence on imputation quality but function should work on any reasonable values of this parameter.

Value

Return data.frame with imputed values.

References

Daniel J. Stekhoven (2013). missForest: Nonparametric Missing Value Imputation using Random Forest. R package version 1.4. Stekhoven D. J., & Buehlmann, P. (2012). MissForest - non-parametric missing value imputation for mixed-type data. Bioinformatics, 28(1), 112-118.

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


 imp_data <- autotune_missForest(raw_data, col_type, percent_of_missing, optimize = FALSE)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

`autotune_missRanger` *Perform imputation using missRenger form missRegnger package.*

Description

Function use missRenger package for data imputation. Function use OBBerror (more in missForest documentation) to perform random search.

Usage

```
autotune_missRanger(
  df,
  percent_of_missing,
  maxiter = 10,
  random.seed = 123,
  mtry = NULL,
  num.trees = 500,
  verbose = F,
  col_0_1 = F,
  out_file = NULL,
  pmm.k = 5,
  optimize = T,
  iter = 10
)
```

Arguments

`df`	data.frame. Df to impute with column names and without target column.
`percent_of_missing`	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
`maxiter`	maximum number of iteration for missRanger algorithm
`random.seed`	random seed use in imputation
`mtry`	sample fraction use by missRanger. This param isn't optimized automatically. If NULL default value from ranger package will be used.
`num.trees`	number of trees. If optimize == TRUE. Param set seq(10,num.trees,iter) will be used.
`verbose`	If FALSE function doesn't print on console.
`col_0_1`	decide if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False.
`out_file`	Output log file location if file already exists log message will be added. If NULL no log will be produced.

pmm.k	Number of candidate non-missing values to sample from in the predictive meanmatching step. 0 to avoid this step. If optimize == TRUE param set sample(1:pmm.k,iter) will be used. If pmm.k==0 missRanger == missForest.
optimize	If TRUE inside optimization will be performed.
iter	Number of iteration for a random search.

Value

Return data.frame with imputed values.

References

Michael Mayer (2019). missRanger: Fast Imputation of Missing Values. R package version 2.1.0. https://CRAN.R-project.org/package=missRanger

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


  imp_data <- autotune_missRanger(raw_data, percent_of_missing, optimize = FALSE)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

---

autotune_softImpute *Perform imputation using softImpute package*

Description

Function use softImpute to impute missing data it works only with numeric data. Columns with categorical values are imputed by a selected function.

Usage

```
autotune_softImpute(
  df,
  percent_of_missing,
  col_type,
  col_0_1 = F,
  cat_Fun = VIM::maxCat,
  lambda = 0,
  rank.max = 2,
  type = "als",
  thresh = 1e-05,
  maxit = 100,
  out_file = NULL
)
```

Arguments

`df`	data.frame. Df to impute with column names and without target column.
`percent_of_missing`	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
`col_type`	Character vector with types of columns.
`col_0_1`	Decaid if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False. (Works only for returning one dataset).
`cat_Fun`	Function to impute categorical features. Default maxCat (mode). Can be every function with input one character vector and return atomic object.
`lambda`	nuclear-norm regularization parameter. If lambda=0, the algorithm reverts to "hardImpute", for which convergence is typically slower. If null lambda is set automatically at the highest possible values.
`rank.max`	This restricts the rank of the solution. Defoult 2 if set as NULL rank.max=min(dim(X))-1.
`type`	Chose of algoritm 'als' or 'svd . Defoult 'als'.
`thresh`	Threshold for convergence.
`maxit`	Maximum number of iterations.
`out_file`	Output log file location if file already exists log message will be added. If NULL no log will be produced.

Details

Function use algorithm base on matrix whats meaning if only one numeric column exists in dataset imputation algorithm don't work. In that case, this column will be imputed using a function for categorical columns. Because of this algorithm is working properly only with at least two numeric features in the dataset. To specify column type argument col_type is used so it's possible to forcefully use for example numeric factors in imputation. Action like this can led to errors and its not.

Value

Return one data.frame with imputed values.

References

Trevor Hastie and Rahul Mazumder (2015). softImpute: Matrix Completion via Iterative Soft-Thresholded SVD. R package version 1.4. https://CRAN.R-project.org/package=softImpute

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


  imp_data <- autotune_softImpute(raw_data, percent_of_missing, col_type)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

---

autotune_VIM_hotdeck *Hot-Deck imputation using VIM package.*

Description

Function perform hotdeck function from VIM package. Any tunable parameters aren't available in this algorithm.

Usage

```
autotune_VIM_hotdeck(df, percent_of_missing, col_0_1 = FALSE, out_file = NULL)
```

Arguments

`df` data.frame. Df to impute with column names and without target column.

`percent_of_missing` numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)

`col_0_1` decide if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False.

`out_file` Output log file location if file already exists log message will be added. If NULL no log will be produced.

Value

Return data.frame with imputed values.

References

Alexander Kowarik, Matthias Templ (2016). Imputation with the R Package VIM. Journal of Statistical Software, 74(7), 1-16. doi:10.18637/jss.v074.i07

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


  imp_data <- autotune_VIM_hotdeck(raw_data, percent_of_missing)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

autotune_VIM_Irmi — *Perform imputation using VIM package and irmi function*

Description

Function use IRMI (Iterative robust model-based imputation) to impute missing data.

Usage

```
autotune_VIM_Irmi(
  df,
  col_type,
  percent_of_missing,
  eps = 5,
  maxit = 100,
  step = FALSE,
  robust = FALSE,
  init.method = "kNN",
  force = FALSE,
  col_0_1 = FALSE,
  out_file = NULL
)
```

Arguments

`df`	data.frame. Df to impute with column names and without target column.
`col_type`	character vector. Vector containing column type names.
`percent_of_missing`	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
`eps`	threshold for convergency
`maxit`	maximum number of iterations
`step`	stepwise model selection is applied when the parameter is set to TRUE
`robust`	if TRUE, robust regression methods will be applied (it's impossible to set step=TRUE and robust=TRUE at the same time)
`init.method`	Method for initialization of missing values (kNN or median)
`force`	if TRUE, the algorithm tries to find a solution in any case, possible by using different robust methods automatically. (should be set FALSE for simulation)
`col_0_1`	Decaid if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False. (Works only for returning one dataset).
`out_file`	Output log file location if file already exists log message will be added. If NULL no log will be produced.

Details

Function can work with various different times depending on data size and structure. In some cases when selected param wouldn't work function try to run on default. Most important param for both quality and reliability its eps.

Value

Return one data.frame with imputed values.

References

Alexander Kowarik, Matthias Templ (2016). Imputation with the R Package VIM. Journal of Statistical Software, 74(7), 1-16. doi:10.18637/jss.v074.i07

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


  imp_data <- autotune_VIM_Irmi(raw_data, col_type, percent_of_missing)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

---

autotune_VIM_kNN *K nearest neighbor imputation using VIM package.*

Description

Function perform kNN function from VIM packge.

@details Function don't perform any inside param tuning. Users can change important param for kNN like number or nearest or aggregation functions.

Usage

```
autotune_VIM_kNN(
  df,
  percent_of_missing,
  k = 5,
  numFun = stats::median,
  catFun = VIM::maxCat,
  col_0_1 = FALSE,
  out_file = NULL
)
```

Arguments

df	data.frame. Df to impute with column names and without target column.
percent_of_missing	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
k	Value of k use if optimize=FALSE
numFun	function for aggregating the k Nearest Neighbours in the case of a numerical variable. Default median.
catFun	function for aggregating the k Nearest Neighbours in the case of a categorical variable. Default mode.
col_0_1	decide if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False.
out_file	Output log file location if file already exists log message will be added. If NULL no log will be produced.

References

Alexander Kowarik, Matthias Templ (2016). Imputation with the R Package VIM. Journal of Statistical Software, 74(7), 1-16. doi:10.18637/jss.v074.i07

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
```

```
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


  imp_data <- autotune_VIM_kNN(raw_data, percent_of_missing)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

autotune_VIM_regrImp — *Perform imputation using VIM package and regressionImp function.*

Description

Function use Regression models to impute missing data.

Usage

```
autotune_VIM_regrImp(
  df,
  col_type,
  percent_of_missing,
  col_0_1 = F,
  robust = F,
  mod_cat = F,
  use_imputed = F,
  out_file = NULL
)
```

Arguments

- `df` — data.frame. Df to impute with column names and without target column.
- `col_type` — Character vector with types of columns.
- `percent_of_missing` — numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
- `col_0_1` — Decaid if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False. (Works only for returning one dataset).
- `robust` — TRUE/FALSE if robust regression should be used.
- `mod_cat` — TRUE/FALSE if TRUE for categorical variables the level with the highest prediction probability is selected, otherwise it is sampled according to the probabilities.

use_imputed TRUE/FALSe if TURE already imputed columns will be used to impute another.

out_file Output log file location if file already exists log message will be added. If NULL no log will be produced.

Details

Function impute one column per iteration to allow more control of imputation. All columns with missing values can be imputed with different formulas. For every new column to imputation one of four formula is used
1. col to impute ~ all columns without missing
2. col to impute ~ all numeric columns without missing
3. col to impute ~ first of columns without missing
4. col to impute ~ first of numeric columns without missing
For example, if formula 1 and 2 can't be used algorithm will try with formula 3. If all formula can't be used function will be stoped and error form tries with formula 4 or 3 presented. In some case, setting use_imputed on TRUE can solve this problem but in general its lower quality of imputation.

Value

Return one data.frame with imputed values.

References

Alexander Kowarik, Matthias Templ (2016). Imputation with the R Package VIM. Journal of Statistical Software, 74(7), 1-16. doi:10.18637/jss.v074.i07

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


  imp_data <- autotune_VIM_regrImp(raw_data, col_type, percent_of_missing)

  # Check if all missing value was imputed
```

```
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

`fetch_data` *Fetch data. Used in mice.reuse.*

Description

Retrieve the main imputation object when within the 'mice:::sampler' post-imputation calling environment and return the data object (including missingness) stored within.

Usage

```
fetch_data()
```

Value

data.frame the original, non-imputed dataset of the mids object

---

`formula_creating` *Creating a formula for use in mice imputation evaluation.*

Description

Function is used in `autotune_mice` but can be use sepraetly.

Usage

```
formula_creating(df, col_miss, col_no_miss, col_type, percent_of_missing)
```

Arguments

`df`	data.frame. Data frame to impute missing values with column names.
`col_miss`	character vector. Names of columns with NA.
`col_no_miss`	character vector. Names of columns without NA.
`col_type`	character vector. A vector containing column type names.
`percent_of_missing`	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)

Details

Function create a formula as follows. It creates one of the formulas its next possible formula impossible possible formula is created:
1. Numeric no missing ~ 3 numeric with most missing
2. Numeric no missing ~ all available numeric with missing
3. Numeric with less missing ~ 3 numeric with most missing
4. Numeric with less missing ~ all available numeric with missing
5. No numeric no missing ~ 3 most missing no numeric
6. No numeric no missing ~ all available no numeric with missing
7. No numeric with less missing ~ 3 no numeric with most missing
8. No numeric with less missing ~ all available no numeric with missing.
For example, if its impossible to create formula 1 and 2 formula 3 will be created but if it's possible to create formula 1 and 5 formula 1 will be created.

Value

List with formula object[1] and information if its no numeric value in dataset[2].

References

Stef van Buuren, Karin Groothuis-Oudshoorn (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1-67. URL https://www.jstatsoft.org/v45/i03/.

`mice.reuse` *Reuseble mice function*

Description

Reuse a previously fit multivariate imputation by chained equations to impute values for previously unseen data without changing the imputation fit (i.e. solely use the original training data to guide the imputation models).

Note: see https://github.com/stefvanbuuren/mice/issues/32 for discussion

Usage

```
mice.reuse(mids, newdata, maxit = 5, printFlag = TRUE, seed = NA)
```

Arguments

`mids`	: mids object An object of class mids, typically produces by a previous call to mice() or mice.mids()
`newdata`	: data.frame Previously unseen data of the same structur as used to generate 'mids'
`maxit`	: integer scalar The number of additional Gibbs sampling iterations to refine the new imputations

`printFlag`	: logical scalar A Boolean flag. If TRUE, diagnostic information during the Gibbs sampling iterations will be written to the command window. The default is TRUE.
`seed`	: integer scalar An integer that is used as argument by the set.seed() for offsetting the random number generator. Default is to use the last seed value stored in ‘mids‘

Value

data : list of data.frames the imputations of newdata

lastSeedValue : integer vector the random seed at the end of the procedure

Author(s)

Patrick Rockenschaub git https://github.com/prockenschaub

`mids.append` *Joining mice objects. Used in mice.reuse.*

Description

Append one mids object to another. Both objects are expected to have the same variables.

Usage

```
mids.append(x, y)
```

Arguments

`x`	mids object provides both data and specification of imputation procedure
`y`	mids object only data information will be retained in the combined object

Details

Only the data specific aspects are copied (i.e. $data, $imp, $where, $nmis), all other information in ‘y‘ is discarded. Therefore, only the imputation model of ‘x‘ is kept and ‘y‘ must not contain missing data in variables that did not have missing data in ‘x‘ (but the reverse is allowed).

Value

mids object a new mids object that contains all of ‘x‘ and the additional data in ‘y‘

missMDA_FMAD_MCA_PCA *Perform imputation using MCA, PCA, or FMAD algorithm.*

Description

Function use missMDA package to perform data imputation. Function can found the best number of dimensions for this imputation. User can choose whether to return one imputed dataset or list or imputed datasets form Multiple Imputation.

Usage

```
missMDA_FMAD_MCA_PCA(
  df,
  col_type,
  percent_of_missing,
  optimize_ncp = TRUE,
  set_ncp = 2,
  col_0_1 = FALSE,
  ncp.max = 5,
  return_one = TRUE,
  random.seed = 123,
  maxiter = 998,
  coeff.ridge = 1,
  threshold = 1e-06,
  method = "Regularized",
  out_file = NULL
)
```

Arguments

`df`	data.frame. Df to impute with column names and without target column.
`col_type`	character vector. Vector containing column type names.
`percent_of_missing`	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
`optimize_ncp`	logical. If true number of dimensions used to predict the missing entries will be optimized. If False by default ncp = 2 it's used.
`set_ncp`	intiger ¿0. Number of dimensions used by algortims. Used only if optimize_ncp = Flase.
`col_0_1`	Decaid if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False. (Works only for returning one dataset).
`ncp.max`	integer corresponding to the maximum number of components to test. Default 5.
`return_one`	One or many imputed sets will be returned. Default True.

`random.seed` random seed.

`maxiter` maximal number of iteration in algortihm.

`coeff.ridge` Value use in Regularized method.

`threshold` threshold for convergence.

`method` method used in imputation algoritm.

`out_file` Output log file location if file already exists log message will be added. If NULL no log will be produced.

Details

Function use different algorithm to adjust for variable types in df. For only numeric data PCA will be used. MCA for only categorical and FMAD for mixed. If optimize==TRUE function will try to find optimal ncp if its not possible default ncp=2 will be used. In some cases ncp=1 will be used if ncp=2 don't work. For multiple imputations, if set ncp don't work error will be return.

Value

Retrun one imputed data.frame if retrun_one=True or list of imputed data.frames if retrun_one=False.

References

Julie Josse, Francois Husson (2016). missMDA: A Package for Handling Missing Values in Multivariate Data Analysis. Journal of Statistical Software, 70(1), 1-31. doi:10.18637/jss.v070.i01

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


 imp_data <- missMDA_FMAD_MCA_PCA(raw_data, col_type, percent_of_missing, optimize_ncp = FALSE)
  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

missMDA_MFA *Perform imputation using MFA algorithm.*

Description

Function use MFA (Multiple Factor Analysis) to impute missing data.

Usage

```
missMDA_MFA(
  df,
  col_type,
  percent_of_missing,
  random.seed = 123,
  ncp = 2,
  col_0_1 = F,
  maxiter = 1000,
  coeff.ridge = 1,
  threshold = 1e-06,
  method = "Regularized",
  out_file = NULL
)
```

Arguments

`df`	data.frame. Df to impute with column names and without target column.
`col_type`	character vector. Vector containing column type names.
`percent_of_missing`	numeric vector. Vector contatining percent of missing data in columns for example c(0,1,0,0,11.3,..)
`random.seed`	random seed.
`ncp`	Number of dimensions used by algorithm. Default 2.
`col_0_1`	Decaid if add bonus column informing where imputation been done. 0 - value was in dataset, 1 - value was imputed. Default False. (Works only for returning one dataset).
`maxiter`	maximal number of iteration in algorithm.
`coeff.ridge`	Value use in Regularized method.
`threshold`	for convergence.
`method`	used in imputation algorithm.
`out_file`	Output log file location if file already exists log message will be added. If NULL no log will be produced.

Details

Groups are created using the original column order and taking as much variable to one group as possible. MFA requires selecting group type but numeric types can only be set as 'c' - centered and 's' - scale to unit variance. It's impossible to provide these conditions so numeric type is always set as 's'. Because of that imputation can depend from column order. In this function, no param is set automatically but if selected ncp don't work function will try use ncp=1.

Value

Return one data.frame with imputed values.

References

Julie Josse, Francois Husson (2016). missMDA: A Package for Handling Missing Values in Multivariate Data Analysis. Journal of Statistical Software, 70(1), 1-31. doi:10.18637/jss.v070.i01

Examples

```
{
  raw_data <- data.frame(
    a = as.factor(sample(c("red", "yellow", "blue", NA), 1000, replace = TRUE)),
    b = as.integer(1:1000),
    c = as.factor(sample(c("YES", "NO", NA), 1000, replace = TRUE)),
    d = runif(1000, 1, 10),
    e = as.factor(sample(c("YES", "NO"), 1000, replace = TRUE)),
   f = as.factor(sample(c("male", "female", "trans", "other", NA), 1000, replace = TRUE)))

  # Prepering col_type
  col_type <- c("factor", "integer", "factor", "numeric", "factor", "factor")

  percent_of_missing <- 1:6
  for (i in percent_of_missing) {
    percent_of_missing[i] <- 100 * (sum(is.na(raw_data[, i])) / nrow(raw_data))
  }


  imp_data <- missMDA_MFA(raw_data, col_type, percent_of_missing)

  # Check if all missing value was imputed
  sum(is.na(imp_data)) == 0
  # TRUE
}
```

`PipeOpAmelia` *PipeOpAmelia*

Description

Implements EMB methods as mlr3 pipeline more about Amelia autotune_Amelia or https://cran.r-project.org/package=Amelia

Input and Output Channels

Input and output channels are inherited from PipeOpImpute.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_Amelia"`.
- `m` :: `integer(1)`
  Number of datasets generated by Amelia, default `3`.
- `polytime` :: `integer(1)`
  Integer between 0 and 3 indicating what power of polynomial should be included in the imputation model to account for the effects of time. A setting of 0 would indicate constant levels, 1 would indicate linear time effects, 2 would indicate squared effects, and 3 would indicate cubic time effects, default `NULL`.
- `splinetime` :: `integer(1)`
  Integer value of 0 or greater to control cubic smoothing splines of time. Values between 0 and 3 create a simple polynomial of time (identical to the polytime argument). Values k greater than 3 create a spline with an additional k-3 knotpoints, default `NULL`.
- `intercs` :: `logical(1)`
  Variable indicating if the time effects of polytime should vary across the cross-section, default `FALSE`.
- `empir` :: `double(1)`
  Number indicating level of the empirical (or ridge) prior. This prior shrinks the covariances of the data, but keeps the means and variances the same for problems of high missingness, small N's or large correlations among the variables. Should be kept small, perhaps 0.5 to 1 percent of the rows of the data; a reasonable upper bound is around 10 percent of the rows of the data. If empir is not set, empir=nrow(df)*0.015, default `NULL`.
- `parallel` :: `double(1)`
  If true parallel calculation is used, default `TRUE`.
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `Amelia_imputation`

Methods

Public methods:

- PipeOpAmelia$new()
- PipeOpAmelia$clone()

Method new():

Usage:

```
PipeOpAmelia$new(
  id = "impute_Amelia_B",
  polytime = NULL,
  splinetime = NULL,
  intercs = FALSE,
  empir = NULL,
  m = 3,
  parallel = TRUE,
  out_file = NULL
)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpAmelia$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
{

# Using debug learner for example purpose

  graph <- PipeOpAmelia$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpHist_B *PipeOpHist_B*

Description

Impute numerical features by histogram in approach B (independently during the training and prediction phase).

Input and Output Channels

Input and output channels are inherited from PipeOpImpute.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default ‘"impute_hist_B"‘.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `Hist_B_imputation`

Methods

Public methods:

- `PipeOpHist_B$new()`
- `PipeOpHist_B$clone()`

Method `new()`:

Usage:

```
PipeOpHist_B$new(id = "impute_hist_B", param_vals = list())
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpHist_B$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{

# Using debug learner for example purpose

  graph <- PipeOpHist_B$new() %>>%  LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpMean_B *PipeOpMean_B*

Description

Impute numerical features by their mean in approach B (independently during the training and prediction phase).

Input and Output Channels

Input and output channels are inherited from PipeOpImpute.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default "`imput_mean_B`".

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `Mean_B_imputation`

Methods

Public methods:

- `PipeOpMean_B$new()`
- `PipeOpMean_B$clone()`

Method `new()`:

Usage:

```
PipeOpMean_B$new(id = "impute_mean_B", param_vals = list())
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpMean_B$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{

 # Using debug learner for example purpose

  graph <- PipeOpMean_B$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

`PipeOpMedian_B` *PipeOpMedian_B*

Description

Impute features by OOR imputation in approach B (independently during the training and prediction phase).

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default ‘"impute_median_B"‘.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `Median_B_imputation`

Methods

Public methods:

- `PipeOpMedian_B$new()`
- `PipeOpMedian_B$clone()`

Method `new()`:

Usage:

`PipeOpMedian_B$new(id = "impute_median_B", param_vals = list())`

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`PipeOpMedian_B$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{

# Using debug learner for example purpose

  graph <- PipeOpMedian_B$new() %>>%  LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpMice *PipeOpMice*

Description

Implements mice methods as mlr3 pipeline more about mice `autotune_mice`

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_mice"`.
- `m` :: `integer(1)`
  Number of datasets produced by mice, default `5`.
- `maxit` :: `integer(1)`
  Maximum number of iterations for mice, default `5`.
- `set_corr` :: `double(1)`
  Correlation or fraction of features used when optimize=FALSE. When correlation=FALSE, it represents a fraction of case to use in imputation for each variable, default `0.5`.
- `set_method` :: `character(1)`
  Method used if optimize=FALSE. If NULL default method is used (more in methods_random section), default `'pmm'`.
- `low_corr` :: `double(1)`
  Double between 0-1. Lower boundary of correlation used in inner optimization (used only when optimize=TRUE), default `0`.

- `up_corr` :: `double(1)`
  Double between 0-1. Upper boundary of correlation used in inner optimization (used only when optimize=TRUE). Both of these parameters work the same for a fraction of case if correlation=FALSE,default `1`.
- `methods_random` :: `character(1)`
  set of methods to chose. Avalible methods "pmm", "midastouch", "sample", "cart", "rf" Default 'pmm'. If seted on NULL this methods are used predictive mean matching (numeric data) logreg, logistic regression imputation (binary data, factor with 2 levels) polyreg, polytomous regression imputation for unordered categorical data (factor ¿ 2 levels) polr, proportional odds model for (ordered, ¿ 2 levels).
- `iter` :: `integer(1)`
  Number of iteration for random search, default `5`.
- `random.seed` :: `integer(1)`
  Random seed, default `123`.
- `optimize` :: `logical(1)`
  If set TRUE, function will optimize parameters of imputation automatically. If parameters will be tuned by other method, should be set to FALSE, default `FALSE`.
- `correlation` :: `logical(1)`
  If set TRUE correlation is used, if set FALSE then fraction of case, default `TRUE`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `mice_imputation`

Methods

Public methods:

- `PipeOpMice$new()`
- `PipeOpMice$clone()`

Method `new()`:

Usage:

```
PipeOpMice$new(
  id = "impute_mice_B",
  m = 5,
  maxit = 5,
  set_cor = 0.5,
  set_method = "pmm",
  low_corr = 0,
  up_corr = 1,
  methods_random = c("pmm"),
  iter = 5,
  random.seed = 123,
  optimize = F,
  correlation = F,
  out_file = NULL
)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`PipeOpMice$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{

# Using debug learner for example purpose

  graph <- PipeOpMice$new() %>>%  LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

`PipeOpMice_A` *PipeOpMice_A*

Description

Implements mice methods as mlr3 in A approach (training imputation model on training data and used a trained model on test data).

Details

Code of used function was writen by https://github.com/prockenschaub more information aboute this aproche can be found here https://github.com/amices/mice/issues/32

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_mice_A"`.
- `m` :: `integer(1)`
  Number of datasets produced by mice, default `5`.

- maxit :: integer(1)
  Maximum number of iterations for mice, default 5.
- set_corr :: double(1)
  Correlation or fraction of features used when optimize=FALSE. When correlation=FALSE, it represents a fraction of case to use in imputation for each variable, default 0.5.
- random.seed :: integer(1)
  Random seed, default 123.
- correlation :: logical(1)
  If set TRUE correlation is used, if set FALSE then fraction of case, default TRUE.

Super classes

mlr3pipelines::PipeOp -¿ mlr3pipelines::PipeOpImpute -¿ mice_A_imputation

Methods

Public methods:

- PipeOpMice_A$new()
- PipeOpMice_A$clone()

Method new():

Usage:

```
PipeOpMice_A$new(
  id = "impute_mice_A",
  set_cor = 0.5,
  m = 5,
  maxit = 5,
  random.seed = 123,
  correlation = F,
  methods = NULL
)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpMice_A$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:

 # Using debug learner for example purpose

  graph <- PipeOpMice_A$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)
```

```
  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))

## End(Not run)
```

`PipeOpmissForest` *PipeOpmissForest*

Description

Implements missForest methods as mlr3 pipeline more about missForest `autotune_missForest`

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_missForest"`.
- `cores` :: `integer(1)`
  Number of threads used by parallel calculations. If NULL approximately half of available CPU cores will be used, default `NULL`.
- `ntree_set` :: `integer(1)`
  Vector with *number of trees* values for grid search, used only when optimize=TRUE, default `c(100,200,500,1000)`.
- `mtry_set` :: `integer(1)`
  Vector with *number of variables* values randomly sampled at each split, used only when optimize=TRUE, default `NULL`.
- `parallel` :: `logical(1)`
  If TRUE parallel calculations are used, default `FALSE`.
- `ntree` :: `integer(1)`
  ntree from missForest function, default `100`.
- `optimize` :: `logical(1)`
  If set TRUE, function will optimize parameters of imputation automatically. If parameters will be tuned by other method, should be set to FALSE, default `FALSE`.
- `mtry` :: `integer(1)`
  mtry from missForest function, default `NULL`.
- `maxiter` :: `integer(1)`
  maxiter from missForest function, default `20`.

- `maxnodes` :: `character(1)`
  maxnodes from missForest function, default `NULL`
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `missForest_imputation`

Methods

Public methods:

- `PipeOpmissForest$new()`
- `PipeOpmissForest$clone()`

Method `new()`:

Usage:

```
PipeOpmissForest$new(
  id = "impute_missForest_B",
  cores = NULL,
  ntree_set = c(100, 200, 500, 1000),
  mtry_set = NULL,
  parallel = F,
  mtry = NULL,
  ntree = 100,
  optimize = FALSE,
  maxiter = 20,
  maxnodes = NULL,
  out_file = NULL
)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpmissForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:

  # Using debug learner for example purpose

  graph <- PipeOpmissForest$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA
```

```
  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))

## End(Not run)
```

PipeOpmissMDA_MFA *PipeOpmissMDA_MFA*

Description

Implements MFA methods as mlr3 pipeline, more about MFA `missMDA_MFA`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_missMDA_MFA"`.
- `ncp` :: `integer(1)`
  Number of dimensions used by algorithm, default `2`.
- `random.seed` :: `integer(1)`
  Random seed, default `123`.
- `maxiter` :: `integer(1)`
  Maximal number of iteration in algorithm, default `998`.
- `coeff.ridge` :: `integer(1)`
  Value used in *Regularized* method, default `1`.
- `threshold` :: `double(1)`
  Threshold for convergence, default `1e-06`.
- `method` :: `character(1)`
  Method used in imputation algorithm, default `'Regularized'`.
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `missMDA_MFAimputation`

Methods

Public methods:

- PipeOpMissMDA_MFA$new()
- PipeOpMissMDA_MFA$clone()

Method new():

Usage:

```
PipeOpMissMDA_MFA$new(
  id = "impute_missMDA_MFA_B",
  ncp = 2,
  random.seed = 123,
  maxiter = 998,
  coeff.ridge = 1,
  threshold = 1e-06,
  method = "Regularized",
  out_file = NULL
)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpMissMDA_MFA$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
{

 # Using debug learner for example purpose

  graph <- PipeOpMissMDA_MFA$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpmissMDA_PCA_MCA_FMAD

PipeOpmissMDA_PCA_MCA_FMAD

Description

Implements PCA, MCA, FMAD methods as mlr3 pipeline, more about methods missMDA_FMAD_MCA_PCA.

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_missMDA_MCA_PCA_FMAD"`.
- `optimize_ncp` :: `logical(1)`
  If TRUE, parameter *number of dimensions*, used to predict the missing values, will be optimized. If FALSE, by default ncp=2 is used, default `TRUE`.
- `set_ncp` :: `integer(1)`
  integer ¿0. Number of dimensions used by algortims. Used only if optimize_ncp = Flase, default `2`.
- `ncp.max` :: `integer(1)`
  Number corresponding to the maximum number of components to test when optimize_ncp=TRUE, default `5`.
- `random.seed` :: `integer(1)`
  Random seed, default `123`.
- `maxiter` :: `integer(1)`
  Maximal number of iteration in algortihm, default `998`.
- `coeff.ridge` :: `double(1)`
  Value used in *Regularized* method, default `1`.
- `threshold` :: `double(1)`
  Threshold for convergence, default `1e-6`.
- `method` :: `character(1)`
  Method used in imputation algorithm, default `'Regularized'`.
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `missMDA_MCA_PCA_FMAD_imputation`

Methods

Public methods:

- `PipeOpMissMDA_PCA_MCA_FMAD$new()`
- `PipeOpMissMDA_PCA_MCA_FMAD$clone()`

Method `new()`:

Usage:

```
PipeOpMissMDA_PCA_MCA_FMAD$new(
  id = "impute_missMDA_MCA_PCA_FMAD_B",
  optimize_ncp = T,
  set_ncp = 2,
  ncp.max = 5,
  random.seed = 123,
  maxiter = 998,
  coeff.ridge = 1,
  threshold = 1e-06,
  method = "Regularized",
  out_file = NULL
)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpMissMDA_PCA_MCA_FMAD$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
{

 # Using debug learner for example purpose


  graph <- PipeOpMissMDA_PCA_MCA_FMAD$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA
  set.seed(1)
  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpmissRanger *PipeOpmissRanger*

Description

Implements missRanger methods as mlr3 pipeline, more about missRanger autotune_missRanger.

Input and Output Channels

Input and output channels are inherited from PipeOpImpute.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_missRanger"`.
- `mtry` :: `integer(1)`
  Sample fraction used by missRanger. This param isn’t optimized automatically. If NULL default value from ranger package will be used, `NULL`.
- `num.trees` :: `integer(1)`
  Number of trees. If optimize == TRUE. Param set seq(10,num.trees,iter) will be used, default `500`
- `pmm.k` :: `integer(1)`
  Number of candidate non-missing values to sample from in the predictive mean matching step. 0 to avoid this step. If optimize=TRUE params set: sample(1:pmm.k, iter) will be used. If pmm.k=0, missRanger is the same as missForest, default `5`.
- `random.seed` :: `integer(1)`
  Random seed, default `123`.
- `iter` :: `integer(1)`
  Number of iterations for a random search, default `10`.
- `optimize` :: `logical(1)`
  If set TRUE, function will optimize parameters of imputation automatically. If parameters will be tuned by other method, should be set to FALSE, default `FALSE`.
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `missRanger_imputation`

Methods

Public methods:

- `PipeOpmissRanger$new()`
- `PipeOpmissRanger$clone()`

Method `new()`:

Usage:

```
PipeOpmissRanger$new(
  id = "impute_missRanger_B",
  maxiter = 10,
  random.seed = 123,
  mtry = NULL,
  num.trees = 500,
  pmm.k = 5,
```

```
    optimize = F,
    iter = 10,
    out_file = NULL
  )
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`PipeOpmissRanger$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:

 # Using debug learner for example purpose

  graph <- PipeOpmissRanger$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))

## End(Not run)
```

PipeOpMode_B *PipeOpMode_B*

Description

Impute features by their mode in approach B (independently during the training and prediction phase).

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default ‘"impute_mode_B"‘.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `Mode_B_imputation`

Methods

Public methods:

- PipeOpMode_B$new()
- PipeOpMode_B$clone()

Method new():

Usage:

```
PipeOpMode_B$new(id = "impute_mode_B", param_vals = list())
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpMode_B$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
{
 # Using debug learner for example purpose

  graph <- PipeOpMode_B$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpOOR_B *PipeOpOOR_B*

Description

Impute features by OOR imputation in approach B (independently during the training and prediction phase).

Input and Output Channels

Input and output channels are inherited from PipeOpImpute.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- id :: character(1)
  Identifier of resulting object, default ‘"impute_OOR_B"‘.

Super classes

mlr3pipelines::PipeOp -¿ mlr3pipelines::PipeOpImpute -¿ OOR_B_imputation

Methods

Public methods:

- PipeOpOOR_B$new()
- PipeOpOOR_B$clone()

Method `new()`:

Usage:

```
PipeOpOOR_B$new(id = "impute_oor_B", param_vals = list())
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpOOR_B$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{

 # Using debug learner for example purpose

  graph <- PipeOpOOR_B$new() %>>% LearnerClassifDebug$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpSample_B *PipeOpSample_B*

Description

Impute features by sampling from non-missing data in approach B (independently during the training and prediction phase).

Input and Output Channels

Input and output channels are inherited from PipeOpImpute.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default ‘”impute_sample_B”‘.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `Sample_B_imputation`

Methods

Public methods:

- `PipeOpSample_B$new()`
- `PipeOpSample_B$clone()`

Method `new()`:

Usage:

```
PipeOpSample_B$new(id = "impute_sample_B", param_vals = list())
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpSample_B$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{
  graph <- PipeOpSample_B$new() %>>% mlr3learners::LearnerClassifGlmnet$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

`PipeOpSimulateMissings` *PipeOpSimulateMissings*

Description

Generates MCAR missing values in mlr3 pipeline according to set parameters. Missings are inserted to task data once during first training.

Input and Output Channels

Input and output channels are inherited from PipeOpTaskPreproc.

Parameters

- `per_missings` :: `double(1)`
  Overall percentage of missing values generated in dataset [0, 100]. Must be set every time, default 50
- `per_instances_missings` :: `double(1)`
  Percentage of instances which will have missing values [0, 100].
- `per_variables_missings` :: `double(1)`
  Percentage of variables which will have missing values [0, 100].
- `variables_missings` :: `integer`
  Only when 'per_variables_missings' is 'NULL'. Vector of indexes of columns in which missings will be generated.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpTaskPreproc` -¿ `PipeOpSimulateMissings`

Methods

Public methods:

- `PipeOpSimulateMissings$new()`
- `PipeOpSimulateMissings$clone()`

Method `new()`:

Usage:

```
PipeOpSimulateMissings$new(
  id = "simulate_missings",
  param_vals = list(per_missings = 50)
)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpSimulateMissings$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{
  task_NA <- PipeOpSimulateMissings$new()$train(list(tsk("iris")))[[1]]

  # check
  sum(task_NA$missings()) > 0
}
```

`PipeOpSoftImpute` *PipeOpSoftImpute*

Description

Implements SoftImpute methods as mlr3 pipeline, more about SoftImpute `autotune_softImpute`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_softImpute"`.
- `lambda` :: `integer(1)`
  Nuclear-norm regularization parameter. If lambda=0, the algorithm reverts to "hardImpute", for which convergence is typically slower. If NULL lambda is set automatically at the highest possible value, default `0`.
- `rank.max` :: `integer(1)`
  This param restricts the rank of the solution. If set as NULL: rank.max=min(dim(X))-1, default `2`.
- `type` :: `character(1)`
  Two algorithms are implemented: type="svd" or the default type="als". The "svd" algorithm repeatedly computes the svd of the completed matrix, and soft thresholds its singular values. Each new soft-thresholded svd is used to re-impute the missing entries. For large matrices of class "Incomplete", the svd is achieved by an efficient form of alternating orthogonal ridge regression. The "als" algorithm uses the same alternating ridge regression, but updates the imputation at each step, leading to quite substantial speedups in some cases. The "als" approach does not currently have the same theoretical convergence guarantees as the "svd" approach, default `'als'`.
- `thresh` :: `double(1)`
  Threshold for convergence, default `1e-5`
- `maxit` :: `integer(1)`
  Maximum number of iterations, default `100`.
- `cat_Fun` :: `function(){}`
  Function for aggregating the k Nearest Neighbors in case of categorical variables. It can be any function with input=not_numeric_vector and output=atomic_object, default `VIM::maxCat`.
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

mlr3pipelines::PipeOp -¿ mlr3pipelines::PipeOpImpute -¿ softImpute_imputation

Methods

Public methods:

- PipeOpSoftImpute$new()
- PipeOpSoftImpute$clone()

Method new():

Usage:

```
PipeOpSoftImpute$new(
  id = "impute_softImpute_B",
  cat_Fun = VIM::maxCat,
  lambda = 0,
  rank.max = 2,
  type = "als",
  thresh = 1e-05,
  maxit = 100,
  out_file = NULL
)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpSoftImpute$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
{
  graph <- PipeOpAmelia$new() %>>% mlr3learners::LearnerClassifGlmnet$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpVIM_HD *PipeOpVIM_HD*

Description

Implements Hot Deck methods as mlr3 pipeline more about VIM_HD autotune_VIM_hotdeck

Input and Output Channels

Input and output channels are inherited from PipeOpImpute.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default "`imput_VIM_HD`".
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `VIM_HD_imputation`

Methods

Public methods:

- `PipeOpVIM_HD$new()`
- `PipeOpVIM_HD$clone()`

Method `new()`:

Usage:

```
PipeOpVIM_HD$new(id = "impute_VIM_HD_B", out_file = NULL)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpVIM_HD$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{
  graph <- PipeOpVIM_HD$new() %>>% mlr3learners::LearnerClassifGlmnet$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpVIM_IRMI *PipeOpVIM_IRMI*

Description

Implements IRMI methods as mlr3 pipeline, more about VIM_IRMI `autotune_VIM_Irmi`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_VIM_IRMI"`.
- `eps` :: `double(1)`
  Threshold for convergence, default `5`.
- `maxit` :: `integer(1)`
  Maximum number of iterations, default `100`
- `step` :: `logical(1)`
  Stepwise model selection is applied when the parameter is set to TRUE, default `FALSE`.
- `robust` :: `logical(1)`
  If TRUE, robust regression methods will be applied (it's impossible to set step=TRUE and robust=TRUE at the same time), default `FALSE`.
- `init.method` :: `character(1)`
  Method for initialization of missing values (kNN or median), default `'kNN'`.
- `force` :: `logical(1)`
  If TRUE, the algorithm tries to find a solution in any case by using different robust methods automatically (should be set FALSE for simulation), default `FALSE`.
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `VIM_IRMI_imputation`

Methods

Public methods:

- `PipeOpVIM_IRMI$new()`
- `PipeOpVIM_IRMI$clone()`

Method `new()`:

Usage:

```
PipeOpVIM_IRMI$new(
  id = "impute_VIM_IRMI_B",
  eps = 5,
  maxit = 100,
  step = FALSE,
  robust = FALSE,
  init.method = "kNN",
  force = FALSE,
  out_file = NULL
)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpVIM_IRMI$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{
  graph <- PipeOpVIM_IRMI$new() %>>% mlr3learners::LearnerClassifGlmnet$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

`PipeOpVIM_kNN` *PipeOpVIM_kNN*

Description

Implements KNN methods as mlr3 pipeline, more about VIM_KNN `autotune_VIM_kNN`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_VIM_kNN"`.

- `k` :: `intiger(1)`
  Threshold for convergence, default `5`.
- `numFUN` :: `function()`{}
  Function for aggregating the k Nearest Neighbours in the case of a numerical variable. Can be ever function with input=numeric_vector and output=atomic_object, default `median`.
- `catFUN` :: `function()`{}
  Function for aggregating the k Nearest Neighbours in case of categorical variables. It can be any function with input=not_numeric_vector and output=atomic_object, default `VIM::maxCat`
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `VIM_kNN_imputation`

Methods

Public methods:

- `PipeOpVIM_kNN$new()`
- `PipeOpVIM_kNN$clone()`

Method `new()`:

Usage:

```
PipeOpVIM_kNN$new(
  id = "impute_VIM_kNN_B",
  k = 5,
  numFun = median,
  catFun = VIM::maxCat,
  out_file = NULL
)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpVIM_kNN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{
  graph <- PipeOpVIM_kNN$new() %>>% mlr3learners::LearnerClassifGlmnet$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA
```

```
  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

PipeOpVIM_regrImp *PipeOpVIM_regrImp*

Description

Implements Regression Imputation methods as mlr3 pipeline, more about RI `autotune_VIM_regrImp`.

Input and Output Channels

Input and output channels are inherited from `PipeOpImpute`.

Parameters

The parameters include inherited from [‘PipeOpImpute‘], as well as:

- `id` :: `character(1)`
  Identifier of resulting object, default `"imput_VIM_regrImp"`.
- `robust` :: `logical(1)`
  TRUE/FALSE: whether to use robust regression, default `FALSE`.
- `mod_cat` :: `logical(1)`
  TTRUE/FALSE if TRUE for categorical variables the level with the highest prediction probability is selected, otherwise it is sampled according to the probabilities, default `FALSE`.
- `use_imputed` :: `logical(1)`
  TRUE/FALSe: if TURE, already imputed columns will be used to impute others, default `FALSE`.
- `out_fill` :: `character(1)`
  Output log file location. If file already exists log message will be added. If NULL no log will be produced, default `NULL`.

Super classes

`mlr3pipelines::PipeOp` -¿ `mlr3pipelines::PipeOpImpute` -¿ `VIM_regrImp_imputation`

Methods

Public methods:

- `PipeOpVIM_regrImp$new()`
- `PipeOpVIM_regrImp$clone()`

Method `new()`:

Usage:

```
PipeOpVIM_regrImp$new(
  id = "impute_VIM_regrImp_B",
  robust = FALSE,
  mod_cat = FALSE,
  use_imputed = FALSE,
  out_file = NULL
)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpVIM_regrImp$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
{
  graph <- PipeOpVIM_regrImp$new() %>>% mlr3learners::LearnerClassifGlmnet$new()
  graph_learner <- GraphLearner$new(graph)

  # Task with NA

  resample(tsk("pima"), graph_learner, rsmp("cv", folds = 3))
}
```

---

random_param_mice_search

Performing randomSearch for selecting the best method and correlation or fraction of features used to create a prediction matrix.

Description

This function perform random search and return values corresponding to best mean MIF (missing information fraction). Function is mainly used in `autotune_mice` but can be use separately.

Usage

```
random_param_mice_search(
  low_corr = 0,
  up_corr = 1,
  methods_random = c("pmm"),
  df,
  formula,
  no_numeric,
  iter,
  random.seed = 123,
  correlation = T
)
```

Arguments

- `low_corr` double between 0,1 default 0 lower boundry of correlation set.
- `up_corr` double between 0,1 default 1 upper boundary of correlation set. Both of these parameters work the same for a fraction of features.
- `methods_random` set of methods to chose. Default 'pmm'.
- `df` data frame to input.
- `formula` first product of formula_creating() funtion. For example formula_creating(...)[1]
- `no_numeric` second product of formula_creating() function.
- `iter` number of iteration for randomSearch.
- `random.seed` radnom seed.
- `correlation` If True correlation is using if Fales fraction of features. Default True.

Details

Function use Random Search Technik to found the best param for mice imputation. To evaluate the next iteration logistic regression or linear regression (depending on available features) are used. Model is build using a formula from `formula_creating` function. As metric MIF (missing information fraction) is used. Params combination with lowest (best) MIF is chosen. Even if a correlation is set at False correlation it's still used to select the best features. That main problem with calculating correlation between categorical columns is still important.

Value

List with best correlation (or fraction) at first place, best method at second, and results of every iteration at 3.

`replace_overimputes` *Replace overimputes. Used in mice.reuse.*

Description

Replace all overimputed data points in the mice imputation of one variable. Overimputed data points are those data that were not missing in the original but were marked for imputation manually and imputed by the imputation procedure.

Usage

```
replace_overimputes(data, imp, j, i)
```

Arguments

- `data` — data.frame the original, non-imputed dataset (mids$data)
- `imp` — list of data.frames all imputations stored in the mids object
- `j` — character scalar the name of the variable whose imputations should be replaced
- `i` — character or integer scalar the number of the current imputation (can be 1:m)

`simulate_missings` *Generate MCAR missings in dataset.*

Description

Function generates random missing values in given dataset according to set parameters.

Usage

```
simulate_missings(
  df,
  per_missings,
  per_instances_missings = NULL,
  per_variables_missings = NULL,
  variables_with_missings = NULL
)
```

Arguments

- `df` — Data.frame or data.table where missing values will be generated
- `per_missings` — Overall percentage of missing values generated in dataset. Must be set every time.
- `per_instances_missings` — Percentage of instances which will have missing values.
- `per_variables_missings` — Percentage of variables which will have missing values.
- `variables_with_missings` — Only when ‘per_variables_missings‘ is ‘NULL‘. Vector of column indexes where missings will be generated.

Value

Dataset with generated missings.

Examples

```
{
  data_NA <- simulate_missings(iris, 20)

  # check
  sum(is.na(data_NA)) > 0
}
```

Index

autotune_Amelia, 2, *26*
autotune_mice, 4, *6*, *19*, *31*, *54*
autotune_missForest, 6, *35*
autotune_missRanger, 9, *40*
autotune_softImpute, 10, *47*
autotune_VIM_hotdeck, 12, *48*
autotune_VIM_Irmi, 14, *50*
autotune_VIM_kNN, 15, *51*
autotune_VIM_regrImp, 17, *53*

fetch_data, 19
formula_creating, *4*, 19, *55*

mice, *4*
mice.reuse, 20
mids.append, 21
missMDA_FMAD_MCA_PCA, 22, *38*
missMDA_MFA, 24, *37*
mlr3pipelines::PipeOp, *26*, *28–30*, *32*, *34*, *36*, *37*, *39*, *41*, *42*, *44–46*, *48–50*, *52*, *53*
mlr3pipelines::PipeOpImpute, *26*, *28–30*, *32*, *34*, *36*, *37*, *39*, *41*, *42*, *44*, *45*, *48–50*, *52*, *53*
mlr3pipelines::PipeOpTaskPreproc, *46*

PipeOpAmelia, 25
PipeOpHist_B, 27
PipeOpImpute, *26*, *27*, *29–31*, *33*, *35*, *37*, *39*, *40*, *42–44*, *47*, *49–51*, *53*
PipeOpMean_B, 29
PipeOpMedian_B, 30
PipeOpMice, 31
PipeOpMice_A, 33
PipeOpmissForest, 35
PipeOpMissMDA_MFA *(*PipeOpmissMDA_MFA*)*, 37
PipeOpmissMDA_MFA, 37
PipeOpMissMDA_PCA_MCA_FMAD *(*PipeOpmissMDA_PCA_MCA_FMAD*)*, 38
PipeOpmissMDA_PCA_MCA_FMAD, 38
PipeOpmissRanger, 40
PipeOpMode_B, 42
PipeOpOOR_B, 43
PipeOpSample_B, 44
PipeOpSimulateMissings, 45
PipeOpSoftImpute, 47
PipeOpTaskPreproc, *46*
PipeOpVIM_HD, 48
PipeOpVIM_IRMI, 50
PipeOpVIM_kNN, 51
PipeOpVIM_regrImp, 53

random_param_mice_search, *4*, 54
registerDoParallel, *8*
replace_overimputes, 55

simulate_missings, 56