Package: imputeGeneric (via r-universe)

September 10, 2024

Title Ease the Implementation of Imputation Methods

Version 0.1.0.9000

Description The general workflow of most imputation methods is quite similar. The aim of this package is to provide parts of this general workflow to make the implementation of imputation methods easier. The heart of an imputation method is normally the used model. These models can be defined using the 'parsnip' package or customized specifications. The rest of an imputation method are more technical specification e.g. which columns and rows should be used for imputation and in which order. These technical specifications can be set inside the imputation functions.

License GPL (>= 3)

URL https://github.com/torockel/imputeGeneric

BugReports https://github.com/torockel/imputeGeneric/issues

Imports gower, parsnip, stats

Suggests missMethods, rpart, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Repository https://torockel.r-universe.dev

RemoteUrl https://github.com/torockel/imputegeneric

RemoteRef HEAD

RemoteSha 5a2920f426cc850b718e91d55165c9cf80e1a020

Contents

`impute_iterative` *Iterative imputation*

Description

Iterative imputation of a data set

Usage

```
impute_iterative(
  ds,
  model_spec_parsnip = linear_reg(),
  model_fun_unsupervised = NULL,
  predict_fun_unsupervised = NULL,
  max_iter = 10,
  stop_fun = NULL,
  initial_imputation_fun = NULL,
  cols_used_for_imputation = "only_complete",
  cols_order = seq_len(ncol(ds)),
  rows_used_for_imputation = "only_complete",
  rows_order = seq_len(nrow(ds)),
  update_model = "every_iteration",
  update_ds_model = "every_iteration",
  stop_fun_args = NULL,
  M = is.na(ds),
  model_arg = NULL,
  warn_incomplete_imputation = TRUE,
  ...
)
```

Arguments

`ds`	The data set to be imputed. Must be a data frame with column names.
`model_spec_parsnip`	The model type used for supervised imputation (see (`impute_supervised()` for details).
`model_fun_unsupervised`	An unsupervised model function (see `impute_unsupervised()` for details).

predict_fun_unsupervised
: A predict function for unsupervised imputation (see impute_unsupervised() for details).

max_iter
: Maximum number of iterations

stop_fun
: A stopping function (see details below) or NULL. If NULL, iterations are only stopped after max_iter is reached.

initial_imputation_fun
: This function will do the initial imputation of the missing values. If NULL, no initial imputation is done. Some common choices like mean imputation are implemented in the package missMethods.

cols_used_for_imputation
: Which columns should be used to impute other columns? Possible choices: "only_complete", "already_imputed", "all"

cols_order
: Ordering of the columns for imputation. This can be a vector with indices or an order_option from order_cols().

rows_used_for_imputation
: Which rows should be used to impute other rows? Possible choices: "only_complete", "partly_complete", "complete_in_k", "already_imputed", "all_except_i", "all"

rows_order
: Ordering of the rows for imputation. This can be a vector with indices or an order_option from order_rows().

update_model
: How often should the model for imputation be updated?

update_ds_model
: How often should the data set for the inner model be updated?

stop_fun_args
: Further arguments passed on to stop_fun.

M
: Missing data indicator matrix

model_arg
: Further arguments for model_fun_unsupervised (see impute_unsupervised() for details).

warn_incomplete_imputation
: Should a warning be given, if the returned data set still contains NA?

...
: Further arguments passed on to stats::predict() or predict_fun_unsupervised.

Details

This function impute a data set in an iterative way. Internally, either impute_supervised() or impute_unsupervised() is used, depending on the values of model_spec_parsnip, model_fun_unsupervised and predict_fun_unsupervised. If you want to use a supervised inner method, model_spec_parsnip must be specified and model_fun_unsupervised and predict_fun_unsupervised must both be NULL. For an unsupervised inner method, model_fun_unsupervised and predict_fun_unsupervised must be specified and model_spec_parsnip must be NULL. Some arguments of this function are only meaningful for impute_supervised() or impute_unsupervised().

Value

an imputed data set (or a return value of stop_fun)

stop_fun

The stop_fun should take the arguments

- ds (the data set imputed in the current iteration)
- ds_old (the data set imputed in the last iteration)
- a list (with named elements M, nr_iterations, max_iter)
- stop_fun_args
- res_stop_fun (the return value of stop_fun from the last iteration. Initial value for the first iteration: list(stop_iter = FALSE)) in this order.

To allow for a next iteration, the stop_fun must return a list which contains the named element stop_iter = FALSE. The simple return list(stop_iter = FALSE) will allow the iteration to continue. However, the list can include more information which are handed over to stop_fun in the next iteration. For example, the return value list(stop_iter = FALSE, last_eps = 0.3) would also lead to another iteration. If stop_fun does not return a list or the list does not contain stop_iter = FALSE the iteration is stopped and the return value of stop_fun is returned as result of impute_iterative(). Therefore, this return value should normally include the imputed data set ds or ds_old.

An example for a stop_fun is stop_ds_difference().

See Also

- impute_supervised() and impute_unsupervised() as the workhorses for the imputation.
- stop_ds_difference() as an example of a stop function.

Examples

```
set.seed(123)
# simple example
ds_mis <- missMethods::delete_MCAR(
  data.frame(X = rnorm(20), Y = rnorm(20)), 0.2, 1
)
impute_iterative(ds_mis, max_iter = 2)
# using pre-imputation
ds_mis <- missMethods::delete_MCAR(
  data.frame(X = rnorm(20), Y = rnorm(20)), 0.2
)
impute_iterative(
  ds_mis,
  max_iter = 2, initial_imputation_fun = missMethods::impute_mean
)
# example using stop_ds_difference() as stop_fun
ds_mis <- missMethods::delete_MCAR(
  data.frame(X = rnorm(20), Y = rnorm(20)), 0.2
)
ds_imp <- impute_iterative(
  ds_mis,
  initial_imputation_fun = missMethods::impute_mean,
  stop_fun = stop_ds_difference, stop_fun_args = list(eps = 0.5)
```

```
)
attr(ds_imp, "nr_iterations")
```

`impute_supervised` *Supervised imputation*

Description

Impute a data set with a supervised inner method. This function is one main function which can be used inside of `impute_iterative()`. If you need pre-imputation or iterations, directly use `impute_iterative()`.

Usage

```
impute_supervised(
  ds,
  model_spec_parsnip = linear_reg(),
  cols_used_for_imputation = "only_complete",
  cols_order = seq_len(ncol(ds)),
  rows_used_for_imputation = "only_complete",
  rows_order = seq_len(nrow(ds)),
  update_model = "each_column",
  update_ds_model = "each_column",
  M = is.na(ds),
  warn_incomplete_imputation = TRUE,
  ...
)
```

Arguments

`ds`	The data set to be imputed. Must be a data frame with column names.
`model_spec_parsnip`	The model type used for imputation. It is defined via the `parsnip` package.
`cols_used_for_imputation`	Which columns should be used to impute other columns? Possible choices: "only_complete", "already_imputed", "all"
`cols_order`	Ordering of the columns for imputation. This can be a vector with indices or an `order_option` from `order_cols()`.
`rows_used_for_imputation`	Which rows should be used to impute other rows? Possible choices: "only_complete", "partly_complete", "complete_in_k", "already_imputed", "all_except_i", "all"
`rows_order`	Ordering of the rows for imputation. This can be a vector with indices or an `order_option` from `order_rows()`.
`update_model`	How often should the model for imputation be updated? Possible choices are: "everytime" (after every imputed value), "each_column" (only one update per column) and "every_iteration" (an alias for "each_column").

update_ds_model
: How often should the data set for the inner model be updated? Possible choices are: "everytime" (after every imputed value), "each_column" (only one update per column) and "every_iteration".

M
: Missing data indicator matrix

warn_incomplete_imputation
: Should a warning be given, if the returned data set still contains NA?

...
: Arguments passed on to stats::predict().

Details

This function imputes the columns of the data set ds column by column. The imputation order of the columns can be specified by cols_order. Furthermore, cols_used_for_imputation controls which columns are used for the imputation. The same options are available for the rows of ds via rows_order and rows_used_for_imputation. If ds is pre-imputed, the missing data indicator matrix can be supplied via M.

The inner method can be specified via model_spec_parsnip which should be a parsnip model type like parsnip::linear_reg(), parsnip::rand_forest() (for a complete list see https://www.tidymodels.org/find/parsnip, you can also build a new parsnip model and use it inside of impute_supervised(), see https://www.tidymodels.org/learn/develop/models for more information on building a parsnip model).

The options "all" for cols_used_for_imputation and "all_except_i", "all" for rows_used_for_imputation should only be used, if ds is complete or the model (model_spec_parsnip) can handle missing data.

The choice update_model = "each_column" can be much faster than update_model = "everytime", especially, if the data set has many missing values in some columns.

Value

The imputed data set.

Examples

```
ds_mis <- missMethods::delete_MCAR(
  data.frame(X = rnorm(20), Y = rnorm(20)), 0.2, 1
)
impute_supervised(ds_mis)
```

---

impute_unsupervised *Unsupervised imputation*

Description

Impute a data set with an unsupervised inner method. This function is one main function which can be used inside of impute_iterative(). If you need pre-imputation or iterations, directly use impute_iterative().

Usage

```
impute_unsupervised(
  ds,
  model_fun,
  predict_fun,
  rows_used_for_imputation = "only_complete",
  rows_order = seq_len(nrow(ds)),
  update_model = "every_iteration",
  update_ds_model = "every_iteration",
  model_arg = NULL,
  M = is.na(ds),
  ...
)
```

Arguments

ds	The data set to be imputed. Must be a data frame with column names.
model_fun	An unsupervised model function which take as arguments ds_used (the data set used to build the model, specified via rows_used_for_imputation), M and i (the index of the row currently under imputation).
predict_fun	A predict function which uses the via model_fun generated model (model_imp) to predict the missing values of a row. It should take the arguments model_imp, ds_used, M and i.
rows_used_for_imputation	Which rows should be used to impute other rows? Possible choices: "only_complete", "already_imputed", "all_except_i", "all"
rows_order	Ordering of the rows for imputation. This can be a vector with indices or an order_option from order_rows().
update_model	How often should the model for imputation be updated? Possible choices are: "everytime" (after every imputed value) and "every_iteration" (only one model is created and used for all missing values).
update_ds_model	How often should the data set for the inner model be updated? Possible choices are: "everytime" (after every imputed value), and "every_iteration".
model_arg	Further arguments for model_fun. This can be a list, if it is more than one argument.
M	Missing data indicator matrix
...	Further arguments given to predict_fun.

Details

This function imputes the rows of the data set ds row by row. The imputation order of the rows can be specified by rows_order. Furthermore, rows_used_for_imputation controls which rows are used for the imputation. If ds is pre-imputed, the missing data indicator matrix can be supplied via M.

The inner method used to impute the data set can be defined with `model_fun`. This `model_fun` must take a data set, the missing data indicator matrix `M`, the index `i` of the row which should be imputed right now (which is `NULL`, if the model is updated only once per iteration or only uses complete rows) and `model_arg` in this order. It must return a model `model_imp` which is given to `predict_fun` to generate imputation values for the missing values in a row `i`. The `model_fun` and `predict_fun` can be self-written or a predefined one (see below) can be used.

If `update_model = "every_iteration"` only one model is fitted and the argument `update_ds_model` is ignored. This option can be considerably faster than `update_model = "everytime"`, especially, for data sets with many rows with missing values. However, some methods (like nearest neighbors) need `update_model = "everytime"`.

Value

The imputed data set.

See Also

`model_donor()` and `predict_donor()` for a pair of predefined functions for `model_fun` and `predict_fun`.

Examples

```
ds_mis <- missMethods::delete_MCAR(
  data.frame(X = rnorm(20), Y = rnorm(20)), 0.2, 1
)
impute_unsupervised(ds_mis, model_donor, predict_donor)
# knn imputation with k = 2
impute_unsupervised(ds_mis, model_donor, predict_donor,
  update_model = "everytime", model_arg = list(k = 2)
)
```

---

model_donor *Model for donor-based imputation*

Description

This function is intended to be used inside of `impute_unsupervised()` as `model_fun`.

Usage

```
model_donor(ds, M = is.na(ds), i = NULL, model_arg = NULL)
```

Arguments

`ds`	The data set to be imputed. Must be a data frame with column names.
`M`	Missing data indicator matrix
`i`	Index for row of `ds` or `NULL`
`model_arg`	A list with two named elements (missing elements will be replaced by default values):

- `selection` How to select the donors? Possible choices are: `complete_rows` (default), `partly_complete_rows`, `knn_complete_rows`, `knn_partly_complete_rows`
- `k` number of selected closest donor (default: 10), only used for knn `selections`

Value

A "model" for `predict_donor()` which is merely a data frame.

See Also

`predict_donor()`

Examples

```
set.seed(123)
ds_mis <- data.frame(X = rnorm(10), Y = rnorm(10))
ds_mis[2:4, 1] <- NA
ds_mis[4:6, 2] <- NA
# default returns only complete rows
model_donor(ds_mis)
# with partly_complete and knn returned objects depends on i
model_donor(ds_mis,
  i = 2,
  model_arg = list(selection = "partly_complete_rows")
)
model_donor(ds_mis,
  i = 4,
  model_arg = list(selection = "partly_complete_rows")
)
model_donor(ds_mis,
  i = 5,
  model_arg = list(selection = "partly_complete_rows")
)
model_donor(ds_mis,
  i = 5,
  model_arg = list(selection = "knn_partly_complete_rows", k = 2)
)
```

---

order_cols *Order column indices*

Description

Order the indices of the columns of `ds` for imputation.

Usage

```
order_cols(ds, order_option, M = is.na(ds))
```

Arguments

ds	A data frame
order_option	This option defines the ordering of the indices. Possible choices are "lowest_md_first", "highest_md_first", "increasing_index", "decreasing_index".
M	Missing data indicator matrix

Value

The ordered column indices of ds as a vector.

Examples

```
ds <- data.frame(X = c(NA, NA, NA, 4), Y = rep(2, 4), Z = c(1, NA, NA, 4))
order_cols(ds, "highest_md_first")
```

---

order_rows *Order row indices*

Description

Order the indices of the rows of ds for imputation.

Usage

```
order_rows(ds, order_option, M = is.na(ds))
```

Arguments

ds	A data frame
order_option	This option defines the ordering of the indices. Possible choices are "lowest_md_first", "highest_md_first", "increasing_index", "decreasing_index".
M	Missing data indicator matrix

Value

The ordered row indices of ds as a vector.

Examples

```
ds <- data.frame(X = c(NA, NA, 3, 4), Y = c(1, NA, NA, 4))
order_rows(ds, "lowest_md_first")
```

predict_donor *Prediction for donor-based imputation*

Description

This function is intended to be used inside of `impute_unsupervised()` as `predict_fun`.

Usage

```
predict_donor(
  ds_donors,
  ds,
  M = is.na(ds),
  i,
  donor_aggregation = "choose_random"
)
```

Arguments

`ds_donors`	Data set with donors, normally generated by `model_donor()`
`ds`	The data set to be imputed. Must be a data frame with column names.
`M`	Missing data indicator matrix
`i`	Index of row of `ds` which should be imputed
`donor_aggregation`	Type of donor aggregation. Can be one of 'choose_random' and 'average'.

Value

The imputation values for row `i`.

See Also

`model_donor()`

Examples

```
set.seed(123)
ds_mis <- data.frame(X = rnorm(10), Y = rnorm(10))
ds_mis[2:4, 1] <- NA
ds_mis[4:6, 2] <- NA
# default for ds_donors and predict_donors
ds_donors <- model_donor(ds_mis)
predict_donor(ds_donors, ds_mis, i = 2)
predict_donor(ds_donors, ds_mis, i = 4)
# with partly_complete, knn and average of neighbors
ds_donors <- model_donor(
  ds_mis,
```

```
  i = 5, model_arg = list(selection = "knn_partly_complete_rows", k = 2)
)
ds_donors
predict_donor(ds_donors, ds_mis, i = 5, donor_aggregation = "average")
```

`stop_ds_difference` *Compare differences between two data sets*

Description

This function is intended to be used as `stop_fun` inside of `impute_iterative()`. It compares the difference of two (numeric) data sets and return `ds`, if difference is small enough (less than `stop_args$eps`).

Usage

```
stop_ds_difference(
  ds,
  ds_old,
  info_list,
  stop_args = list(eps = 1e-06, p = 1, sum_diffs = TRUE, na_rm = TRUE),
  res_stop_fun = NULL
)
```

Arguments

`ds`	A numeric data set
`ds_old`	A numeric data set
`info_list`	`info_list` used inside of `impute_iterative()`. Only the list element `nr_iterations` is used/needed.
`stop_args`	A list with following named components (missing elements will be replaced by default ones): • `eps` Threshold value for the difference (default = 1e-6). • `p` Exponent used for the calculation of differences similar to Minkowski distance. For `p = 1` (default) the absolute differences are used. For `p = 2` The quadratic differences are summed and the square root of this sum is compared with `stop_eps`. • `sum_diffs` Should differences be summed (default) or averaged (`sum_diffs = FALSE`)? • `na_rm` Should `NA`-values be removed (default) when calculating the sum/average? If `na_rm = FALSE` and there are `NA`s, the function returns `FALSE`.
`res_stop_fun`	Only needed to be a valid stop function. Internally, this argument is ignored at the moment.

Value

`list(stop_iter = FALSE)`, if the difference is too big. Otherwise `ds` with number of iterations (`nr_iterations`) as attribute.

Examples

```
set.seed(123)
ds1 <- data.frame(X = rnorm(10), Y = rnorm(10))
ds2 <- data.frame(X = rnorm(10), Y = rnorm(10))
all.equal(
  stop_ds_difference(ds1, ds1, list(nr_iterations = 3)),
  structure(ds1, nr_iterations = 3)
)
stop_ds_difference(ds1, ds2, list(nr_iterations = 42))
```

Index