Package ‘missForestPredict’

July 22, 2025

Type Package

Title Missing Value Imputation using Random Forest for Prediction Settings

Version 1.0.1

Date 2025-05-23

Depends R (>= 4.0)

Imports ranger, methods, stats

Suggests knitr, rmarkdown, ggplot2, dplyr, tidyr

Description Missing data imputation based on the 'missForest' algorithm (Stekhoven, Daniel J (2012) <doi:10.1093/bioinformatics/btr597>) with adaptations for prediction settings. The function missForest() is used to impute a (training) dataset with missing values and to learn imputation models that can be later used for imputing new observations. The function missForestPredict() is used to impute one or multiple new observations (test set) using the models learned on the training data. For more details see Albu, E., Gao, S., Wynants, L., & Van Calster, B. (2024). missForestPredict--Missing data imputation for prediction settings <doi:10.48550/arXiv.2407.03379>.

Encoding UTF-8

License GPL (>= 2)

URL `https://github.com/sibipx/missForestPredict`

BugReports `https://github.com/sibipx/missForestPredict/issues`

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Elena Albu [aut, cre] (ORCID: <`https://orcid.org/0000-0003-2602-0918`>, funding: KU Leuven)

Maintainer Elena Albu <`elenaa.albu@gmail.com`>

Repository CRAN

Date/Publication 2025-05-24 15:40:05 UTC

Contents

`calculate_convergence` *Calculates convergence based on NMSE*

Description

Calculates convergence based on NMSE. Details on the convergence criterion calculation are provided in the package vignettes.

Usage

```
calculate_convergence(err, weights)
```

Arguments

`err`	dataframe containing OOB or apparent errors for each iteration.
`weights`	vector of weights in the same format as for the `missForest` function.

Value

A list with elements

`converged`	boolean indicating if the algorithm has converged (TRUE) or not (FALSE)
`measure_old`	the total error of the previous iteration
`measure_new`	the total error of the last iteration

`check_predictor_matrix` *Performs checks on a custom predictor matrix*

Description

Performs a number of checks on a custom predictor matrix and returns error if the matrix is not in a valid format.

Usage

```
check_predictor_matrix(predictor_matrix, data, verbose = TRUE)
```

Arguments

`predictor_matrix`	custom predictor matrix
`data`	dataframe to be imputed
`verbose`	if TRUE, returns human readable message for the checks performed.

Value

No return value, only verifies the validity the provided predictor matrix

---

`create_predictor_matrix` *Creates the default predictor matrix with 0 diagonal and 1 elements in the rest.*

Description

Creates a square predictor matrix with number of rows and columns equal to the number of columns of the data. Each row name represents the name of the column to be imputed. Each column name represents a predictor for the imputation model. The values in a row will contain 1 for the variables that should be included as predictors in the imputation model and 0 if the variable should not be included as predictor. The diagonal of the predictor matrix is 0, indicating that variable X will not be a predictor for the imputation model of variable X. All other values are 1, meaning that all other variables will be included in the imputation model. This is the default predictor matrix used.

Usage

```
create_predictor_matrix(data)
```

Arguments

`data`	dataframe to be imputed

Value

predictor matrix that can be used as a start for setting a custom predictor matrix

---

`evaluate_imputation_error`

Evaluate the imputation error when true values are known.

Description

Evaluate the imputation error when true values are known. Useful when missing values are simulated and true values are known (the errors are calculated as distances from the true values). For continuous variables, MSE (mean square error) and NMSE (normalized mean square error) are returned. For categorical variables, MER (misclassification error rate) is returned.

Usage

```
evaluate_imputation_error(ximp, xmis, xtrue, all = FALSE)
```

Arguments

`ximp`	imputed dataframe.
`xmis`	original dataframe with missing values.
`xtrue`	true dataframe with no missing values.
`all`	calculate error on all observations (TRUE) or only on missing observations (FALSE). Default is FALSE.

Details

For details check the advanced vignette on convergence criteria and error monitoring.

Value

Dataframe with variables in rows and performance measures in columns.

missForest — *Imputes a dataframe and returns imputation models to be used on new observations*

Description

Imputes a dataframe and (if save_models = TRUE) returns imputation models to be used on new observations.

Usage

```
missForest(
  xmis,
  maxiter = 10,
  fixed_maxiter = FALSE,
  var_weights = NULL,
  decreasing = FALSE,
  initialization = "mean/mode",
  x_init = NULL,
  class.weights = NULL,
  return_integer_as_integer = FALSE,
  save_models = TRUE,
  predictor_matrix = NULL,
  proportion_usable_cases = c(1, 0),
  verbose = TRUE,
  convergence_error = "OOB",
  ...
)
```

Arguments

`xmis`	dataframe containing missing values of class dataframe ("tibble" class tbl_df is also supported). Matrix format is not supported. See details for column format.
`maxiter`	maximum number of iterations. By default the algorithm will stop when converge is reached or after running for maxiter, whichever occurs first.
`fixed_maxiter`	if set to TRUE, the algorithm will run for the exact number of iterations specified in maxiter, regardless of the convergence criteria. Default is FALSE.
`var_weights`	named vector of weights for each variable in the convergence criteria. The names should correspond to variable names. By default the weights are set to the proportion of missing values on each variable.
`decreasing`	(boolean) if TRUE the order in which the variables are imputed is by decreasing amount of missing values. (the variable with highest amount of missing values will be imputed first). If FALSE the variable with lowest amount of missing values will be imputed first.
`initialization`	initialization method before running RF models; supported: mean/mode, median/mode and custom. Default is mean/mode.

`x_init`	if `initialization` = `custom`; a complete dataframe to be used as initialization (see vignette for example).
`class.weights`	a named list containing `class.weights` parameter to be passed to ranger for categorical variables. The names of the list needs to respect the names of the categorical variables in the dataframe. (See `ranger` function documentation in `ranger` package for details).
`return_integer_as_integer`	Internally, integer columns are treated as double (double precision floating point numbers). If TRUE, the imputations will be rounded to closest integer and returned as integer (This might be desirable for count variables). If FALSE, integer columns will be returned as double (This might be desirable, for example, for patient age imputation). Default is FALSE. The same behaviour will be applied to new observations when using missForestPredict.
`save_models`	if TRUE, imputation models are saved and a new observation (or a test set) can be imputed using the models learned; saving models on a dataset with a high number of variables will occupy RAM memory on the machine. Default is TRUE.
`predictor_matrix`	predictor matrix indicating which variables to use in the imputation of each variable. See documentation for function `create_predictor_matrix` for details on the matrix format.
`proportion_usable_cases`	a vector with two components: the first one is a minimum threshold for `p_obs` and the second one is a maximum threshold for `p_miss`. Variables for which `p_obs` is greater than or equal to 1 (by default) will be filtered from the predictor matrix. Variables for which `p_miss` is lower than or equal to 0 (by default) will be filtered from the predictor matrix. For more details on `p_obs` and `p_miss` see the documentation for the `prop_usable_cases` function. If parameter `predictor_matrix` is specified, the `proportion_usable_cases` will be applied to this provided matrix.
`verbose`	(boolean) if TRUE then missForest returns OOB error estimates (MSE and NMSE) and runtime.
`convergence_error`	Which error should be used for the convergence criterion. Supported values: OOB and apparent. If a different value is provided, it defaults to OOB. See vignette for full details on convergence.
`...`	other arguments passed to ranger function (some arguments that are specific to each variable type are not supported). See vignette for `num.trees` example.

Details

An adaptation of the original missForest algorithm (Stekhoven et al. 2012) is used. Variables are initialized with a mean/mode, median/mode or custom imputation. Then, they are imputed iteratively "on the fly" for a maximum number of iterations or until the convergence criteria are met. The imputation sequence is either increasing or decreasing. At each iteration, a random forest model is build for each variable using as outcome on the observed (non-missing) values of the variable and as predictors the values of the other variables from previous iteration for the first variable in the sequence or current iteration for next variables in the sequence (on-the-fly). The ranger package (Wright et al. 2017) is used for building the random forest models.

The convergence criterion is based on the out-of-boostrap (OOB) error or the apparent error and uses NMSE (normalized mean squared error) for both continuous and categorical variables.

Imputation models for all variables and all iterations are saved (if `save_models` is TRUE) and can be later applied to new observations.

Both dataframe and tibble (tbl_df class) are supported as input. The imputed dataframe will be retured with the same class. Numeric and integer columns are supported and treated internally as continuous variables. Factor and character columns are supported and treated internally as categorical variables. Other types (like boolean or dates) are not supported. NA values are considered missing values.

Value

Object of class `missForest` with elements

`ximp`	dataframe with imputed values
`init`	x_init if custom initalization is used; otherwise list of mean/mode or median/mode for each variable
`initialization`	value of initialization parameter
`impute_sequence`	vector variable names in the order in which imputation has been run
`maxiter`	maxiter parameter as passed to the function
`models`	list of random forest models for each iteration
`return_integer_as_integer`	Parameter return_integer_as_integer as passed to the function
`integer_columns`	list of columns of integer type in the data
`OOB_err`	dataframe with out-of-bag errors for each iteration and each variable

References

- Stekhoven, D. J., & Bühlmann, P. (2012). MissForest-non-parametric missing value imputation for mixed-type data. Bioinformatics, 28(1), 112-118. doi:10.1093/bioinformatics/btr597
- Wright, M. N. & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. J Stat Softw 77:1-17. doi:10.18637/jss.v077.i01.

Examples

```
data(iris)
iris_mis <- produce_NA(iris, proportion = 0.1)
imputation_object <- missForest(iris_mis, num.threads = 2)
iris_imp <- imputation_object$ximp
```

missForestPredict *Imputes a new dataframe based on the missForest models*

Description

Imputes a new dataframe based on the missForest models. The same number of iterations as in missForest are used.

Usage

```
missForestPredict(missForestObj, newdata, x_init = NULL)
```

Arguments

missForestObj	missForest object as returned by the missForest function.
newdata	new data to impute. The column names should be the same as in the imputation model.
x_init	initialization dataframe in case custom initialization mode has been used. It needs to be complete dataframe (with no missing values). See vignette for a full example.

Details

A new observation is initialized in the same manner as passed through the `initialization` parameter to the `missForest` function. Then, variables are imputed in the same sequence and for the same number of iterations using the random models saved for each iteration. This ensures that a new observation is imputed in the same manner as the training set (imputed by the function `missForest`). Re-imputing the training set with the `missForestPredict` will yield the same result as the original imputation returned by the `missForest` function.

Value

an imputed dataframe

Examples

```
data(iris)
# split train / test and create missing values
id_test <- sample(1:nrow(iris), floor(nrow(iris)/3))

iris_train <- iris[-id_test,]
iris_test <- iris[id_test,]

iris_train_miss <- produce_NA(iris_train, proportion = 0.1)
iris_test_miss <- produce_NA(iris_test, proportion = 0.1)

# impute train and learn imputation models
iris_train_imp_obj <- missForest(iris_train_miss, save_models = TRUE, num.threads = 2)
```

```
# impute test
iris_test_imp_new <- missForestPredict(iris_train_imp_obj, newdata = iris_test_miss)
head(iris_test_imp_new)
```

---

produce_NA *Produces a dataframe with missing values*

Description

Produces a dataframe with missing values. For each variable, a random sample of observations of size `proportion * nrow(x)` will be set to NA.

Usage

```
produce_NA(x, proportion = 0.1)
```

Arguments

`x`	a dataframe
`proportion`	proportion of missing values to be produced; a vector of size `ncol(x)` or a single value to be applied to all variables in the dataframe.

Value

dataframe with missing values.

---

prop_usable_cases *Calculates variable-wise proportion of usable cases (missing and observed)*

Description

Calculates variable-wise proportion of usable cases (missing and observed) as in Molenberghs et al. (2014).

Usage

```
prop_usable_cases(data)
```

Arguments

`data`	dataframe to be imputed

Details

missForest builds models for each variable using the observed values of that variable as outcome of a random forest model. It then imputes the missing part of the variable using the learned models.

If all values of a predictor are missing among the observed value of the outcome, the value of `p_obs` will be 1 and the model built will rely heavily on the initialized values. If all values of a predictor are observed among the observed values of the outcome, `p_obs` will be 0 and the model will rely on observed values. Low values of `p_obs` are preferred.

Similarly, if all values of a predictor are missing among the missing values of the outcome, `p_miss` will have a value of 0 and the imputations (predictions) will heavily rely on the initialized values. If all values of a predictor are observed among the missing value of the outcome, `p_miss` will have a value of 1 and the imputations (predictions) will rely on real values. High values of `p_miss` are preferred.

Each row represents a variable to be imputed and each column a predictor.

Value

a list with two elements: `p_obs` and `p_miss`

`p_obs`	the proportion of missing Y_k among observed Y_j; j in rows, k in columns
`p_miss`	the proportion of observed Y_k among missing Y_j; j in rows, k in columns

References

- Molenberghs, G., Fitzmaurice, G., Kenward, M. G., Tsiatis, A., & Verbeke, G. (Eds.). (2014). Handbook of missing data methodology. CRC Press. Chapter "Multiple Imputation"

Index