

# Package ‘imputeR’

October 13, 2022

**Title** A General Multivariate Imputation Framework

**Version** 2.2

**Date** 2020-01-20

**Author** Steffen Moritz [aut, cre] (<<https://orcid.org/0000-0002-0085-1804>>),  
Lingbing Feng [aut],  
Gen Nowak [ctb],  
Alan. H. Welsh [ctb],  
Terry. J. O'Neill [ctb]

**Maintainer** Steffen Moritz <[steffen.moritz10@gmail.com](mailto:steffen.moritz10@gmail.com)>

**Description** Multivariate Expectation-Maximization (EM) based imputation framework that offers several different algorithms. These include regularisation methods like Lasso and Ridge regression, tree-based models and dimensionality reduction methods like PCA and PLS.

**LazyData** yes

**Type** Package

**ByteCompile** TRUE

**BugReports** <https://github.com/SteffenMoritz/imputeR/issues>

**URL** <http://github.com/SteffenMoritz/imputeR>

**Repository** CRAN

**Depends** R (>= 3.1.0),

**Imports** stats, utils, graphics, reshape2

**Suggests** testthat, caret, glmnet, pls, Cubist, ridge, gbm, mboost,  
rpart, earth

**License** GPL-3

**RoxygenNote** 6.1.1

**NeedsCompilation** no

**Date/Publication** 2020-01-20 22:00:03 UTC

**R topics documented:**

imputeR-package	2
CubistR	3
Detect	4
gbmC	4
glmboostR	5
guess	6
impute	6
lassoC	8
lassoR	8
major	9
mixError	10
mixGuess	10
mr	11
orderbox	12
parkinson	13
pcrR	13
plotIm	14
plsR	15
ridgeC	16
ridgeR	16
Rmse	17
rpartC	18
SimEval	18
SimIm	20
spect	21
stepBackC	21
stepBackR	22
stepBothC	23
stepBothR	23
stepForC	24
stepForR	25
tic	25
<b>Index</b>	<b>27</b>

---

imputeR-package	<i>imputeR-package description</i>
-----------------	------------------------------------

---

**Description**

The imputeR package offers a General Multivariate Imputation Framework

**Details**

The imputeR package is a Multivariate Expectation-Maximization (EM) based imputation framework that offers several different algorithms. These include regularisation methods like Lasso and Ridge regression, tree-based models and dimensionality reduction methods like PCA and PLS.

**Author(s)**

Steffen Moritz, Lingbing Feng, Gen Nowak, Alan. H. Welsh, Terry. J. O'Neill

---

CubistR

*Cubist method for imputation*

---

**Description**

Quinlan's Cubist model for imputation

**Usage**

```
CubistR(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function and the optimal value for the "neighbors".

**See Also**

[cubist](#)

**Examples**

```
data(parkinson)
missdata <- SimIm(parkinson, 0.1)

impdata <- impute(missdata, lmFun = "CubistR")
```

---

Detect	<i>Detect variable type in a data matrix</i>
--------	--

---

**Description**

This function detects the type of the variables in a data matrix. Types can be continuous only, categorical only or mixed type. The rule for defining a variable as a categorical variable is when: (1) it is a character vector, (2) it contains no more than  $n = 5$  unique values

**Usage**

```
Detect(x, n = 5)
```

**Arguments**

x	is the data matrix that need to be detected.
n	is a number, indicating how many levels, if outnumbered, can be seen as an numeric variable, rather than a categorical variable.

**Value**

the variable type for every column, can either be "numeric" or "character".

**Examples**

```
data(parkinson)
Detect(parkinson)
data(spect)
Detect(spect)
data(tic)
table(Detect(tic))
```

---

gbmC	<i>boosting tree for imputation</i>
------	-------------------------------------

---

**Description**

boosting tree for imputation

**Usage**

```
gbmC(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function and the best.iter for gbm model.

**See Also**

[gbm](#)

**Examples**

```
data(spect)
misssdata <- SimIm(spect, 0.1)

impdata <- impute(spect, cFun = "gbmC")
```

---

glmboostR

*Boosting for regression*

---

**Description**

boosting variable selection for continuous data

**Usage**

```
glmboostR(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**Examples**

```
data(parkinson)
misssdata <- SimIm(parkinson, 0.1)

impdata <- impute(misssdata, lmFun = "glmboostR")
```

---

guess	<i>Impute by (educated) guessing</i>
-------	--------------------------------------

---

### Description

This function use some primitive methods, including mean imputation, median imputation, random guess, or majority imputation (only for categorical variables), to impute a missing data matrix.

### Usage

```
guess(x, type = "mean")
```

### Arguments

x	a matrix or data frame
type	is the guessing type, including "mean" for mean imputation, "median" for median imputation, "random" for random guess, and "majority" for majority imputation for categorical variables.

### Examples

```
data(parkinson)
# introduce some random missing values
misdata <- SimIm(parkinson, 0.1)
# impute by mean imputation
impdata <- guess(misdata)
# caculate the NRMSE
Rmse(impdata, misdata, parkinson, norm = TRUE)
# by random guessing, the NRMSE should be much bigger
impdata2 <- guess(misdata, "random")
Rmse(impdata2, misdata, parkinson, norm = TRUE)
```

---

impute	<i>General Imputation Framework in R</i>
--------	--

---

### Description

Impute missing values under the general framework in R

### Usage

```
impute(misdata, lmFun = NULL, cFun = NULL, ini = NULL,
       maxiter = 100, verbose = TRUE, conv = TRUE)
```

**Arguments**

misssdata	data matrix with missing values encoded as NA.
lmFun	the variable selection method for continuous data.
cFun	the variable selection method for categorical data.
ini	the method for initialisation. It is a length one character if misssdata contains only one type of variables only. For continuous only data, ini can be "mean" (mean imputation), "median" (median imputation) or "random" (random guess), the default is "mean". For categorical data, it can be either "majority" or "random", the default is "majority". If misssdata is mixed of continuous and categorical data, then ini has to be a vector of two characters, with the first element indicating the method for continuous variables and the other element for categorical variables, and the default is c("mean", "majority".)
maxiter	is the maximum number of iterations
verbose	is logical, if TRUE then detailed information will be printed in the console while running.
conv	logical, if TRUE, the convergence details will be returned

**Details**

This function can impute several kinds of data, including continuous-only data, categorical-only data and mixed-type data. Many methods can be used, including regularisation method like LASSO and ridge regression, tree-based model and dimensionality reduction method like PCA and PLS.

**Value**

if conv = FALSE, it returns a completed data matrix with no missing values; if TRUE, it returns a list of components including:

imp	the imputed data matrix with no missing values
conv	the convergence status during the imputation

**See Also**

[SimIm](#) for missing value simulation.

**Examples**

```
data(parkinson)
# introduce 10% random missing values into the parkinson data
misssdata <- SimIm(parkinson, 0.1)
# impute the missing values by LASSO

impdata <- impute(misssdata, lmFun = "lassoR")
# calculate the normalised RMSE for the imputation
Rmse(impdata$imp, misssdata, parkinson, norm = TRUE)
```

lassoC

*logistic regression with lasso for imputation*

---

**Description**

logistic regression with lasso for imputation

**Usage**

```
lassoC(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**See Also**

[cv.glmnet](#) and [glmnet](#)

**Examples**

```
data(spect)
misssdata <- SimIm(spect, 0.1)

impdata <- impute(spect, cFun = "lassoC")
```

---

lassoR*LASSO for regression*

---

**Description**

LASSO variable selection for continuous data

**Usage**

```
lassoR(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the `impute` function

**Examples**

```
data(parkinson)
misssdata <- SimIm(parkinson, 0.1)

impdata <- impute(misssdata, lmFun = "lassoR")
```

---

major

*Majority imputation for a vector*

---

**Description**

This function is internally used by `guess`, it may be useless in reality.

**Usage**

```
major(x)
```

**Arguments**

x a character (or numeric categorical) vector with missing values

**Value**

the same length of vector with missing values being imputed by the majority class in this vector.

**Examples**

```
a <- c(rep(0, 10), rep(1, 15), rep(2, 5))
a[sample(seq_along(a), 5)] <- NA
a
b <- major(a)
b
```

---

mixError	<i>Calculate mixed error when the imputed matrix is mixed type</i>
----------	--

---

**Description**

Calculate mixed error when the imputed matrix is mixed type

**Usage**

```
mixError(imp, mis, true, norm = TRUE)
```

**Arguments**

imp	the imputed matrix
mis	the original matrix with missing values
true	the true matrix
norm	logical, if TRUE, the non-normalised RMSE will return for continuous variables

**Value**

a vector of two values indicating the mixed error the the imputation, the first one if either RMSE or NRMSE, the second one is MCE.

**Examples**

```
data(tic)
Detect(tic)
missdata <- SimIm(tic, 0.3)

library(earth)
impdata <- impute(tic, lmFun = "earth", cFun = "rpartC")
mixError(impdata$imp, missdata, tic)
```

---

mixGuess	<i>Naive imputation for mixed type data</i>
----------	---

---

**Description**

Naive imputation for mixed type data

**Usage**

```
mixGuess(missdata, method = c("mean", "majority"))
```

**Arguments**

`misssdata` a data matrix with missing values

`method` a character vector of length 2 indicating which two methods to use respectively for continuous variables and categorical variables. There are three options for continuous variables: "mean", "median" and "random", and two options for categorical variables: "majority" and "random". The default method is "mean" for the continuous part and "majority" for the categorical part.

**Value**

the same size data matrix with no missing value.

**Examples**

```
data(tic)
misssdata <- SimIm(tic, 0.1)
sum(is.na(misssdata))
impdata <- mixGuess(misssdata)
sum(is.na(impdata))
```

---

`mr` *calculate miss-classification error*

---

**Description**

This function calculates the misclassification error given the imputed data, the missing data and the true data.

**Usage**

```
mr(imp, mis, true)
```

**Arguments**

`imp` the imputed data matrix

`mis` the missing data matrix

`true` the true data matrix

**Value**

The misclassification error

**Examples**

```

data(spect)
Detect(spect)
missdata <- SimIm(spect, 0.1)

sum(is.na(missdata))
# impute using rpart
impdata <- impute(missdata, cFun = "rpartC")
# calculate the misclassification error
mr(impdata$imp, missdata, spect)

```

---

orderbox

*Ordered boxplot for a data matrix*


---

**Description**

Ordered boxplot for a data matrix

**Usage**

```

orderbox(x, names = c("method", "MCE"), order.by = mean,
         decreasing = TRUE, notch = TRUE, col = "bisque", mar = c(7, 4.1,
         4.1, 2), ...)

```

**Arguments**

x	a matrix
names	a length two character vector, default is c("method", "MCE")
order.by	which statistics to order by, default is mean
decreasing	default is TRUE, the boxplot will be arranged in a decreasing order
notch	logical, default is TRUE
col	color for the boxplots, default is "bisque".
mar	the margin for the plot, adjust it to your need.
...	some other arguments that can be passed to the boxplot function

**Value**

a boxplot

**Examples**

```

data(parkinson)

orderbox(parkinson)

```

---

parkinson

*Parkinsons Data Set*

---

### Description

This dataset contains a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease. Each row corresponds to one of 195 individuals and each column a measurement variable. This data was originally obtained from the UCI Machine Learning Repository. For detailed information about the columns, see the reference and the source below. In the study of simulation, this dataset can be treated as continuous-only data

### Format

A data frame with 195 rows and 22 variables

### Details

- MDVP:Fo(Hz). Average vocal fundamental frequency
- MDVP:Fhi(Hz). Maximum vocal fundamental frequency
- MDVP:Flo(Hz). Minimum vocal fundamental frequency
- ...

### Source

<http://archive.ics.uci.edu/ml/datasets/Parkinsons>

### References

Little MA, McSharry PE, Roberts SJ, Costello DAE, Moroz IM, 2007 Exploiting Nonlinear Recurrence and Fractal Scaling Properties for Voice Disorder Detection, *BioMedical Engineering OnLine*

---

pcrR

*Principle component regression for imputation*

---

### Description

Principle component regression method for imputation

### Usage

```
pcrR(x, y)
```

### Arguments

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the `impute` function

**See Also**

`pcr`

**Examples**

```
data(parkinson)
misdata <- SimIm(parkinson, 0.1)

impdata <- impute(misdata, lmFun = "pcrR")
```

---

`plotIm`*Plot function for imputation*

---

**Description**

this is a plot function for assessing imputation performance given the imputed data and the original true data

**Usage**

```
plotIm(imp, mis, true, ...)
```

**Arguments**

<code>imp</code>	the imputed data matrix
<code>mis</code>	the missing data matrix
<code>true,</code>	the true data matrix
<code>...</code>	other arguments that can be passed to plot

**Value**

a plot object that show the imputation performance

**Examples**

```
data(parkinson)
# introduce 10% random missing values into the parkinson data
misdata <- SimIm(parkinson, 0.1)

# impute the missing values by LASSO
impdata <- impute(misdata, lmFun = "lassoR")

# calculate the normalised RMSE for the imputation
```

```
Rmse(impdata$imp, missdata, parkinson, norm = T)

# Plot imputation performance
plotIm(impdata$imp, missdata, parkinson)
```

---

plsR

*Partial Least Square regression for imputation*

---

## Description

Principle component regression method for imputation

## Usage

```
plsR(x, y)
```

## Arguments

x	predictor matrix
y	response vector

## Value

a model object that can be used by the [impute](#) function

## See Also

[plsR](#)

## Examples

```
data(parkinson)
missdata <- SimIm(parkinson, 0.1)

impdata <- impute(missdata, lmFun = "plsR")
```

---

ridgeC	<i>Ridge regression with lasso for imputation</i>
--------	---

---

**Description**

Ridge regression with lasso for imputation

**Usage**

```
ridgeC(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**See Also**

[logisticRidge](#)

**Examples**

```
data(spect)
misssdata <- SimIm(spect, 0.1)

impdata <- impute(spect, cFun = "ridgeC")
```

---

ridgeR	<i>Ridge shrinkage for regression</i>
--------	---------------------------------------

---

**Description**

Ridge shrinkage variable selection for continuous data

**Usage**

```
ridgeR(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**Examples**

```
data(parkinson)
misssdata <- SimIm(parkinson, 0.1)

impdata <- impute(misssdata, lmFun = "ridgeR")
```

---

Rmse

*calculate the RMSE or NRMSE*

---

**Description**

This function calculate imputation error given the imputed data, the missing data and the true data

**Usage**

```
Rmse(imp, mis, true, norm = FALSE)
```

**Arguments**

imp	the imputaed data matrix
mis	the missing data matrix
true	the true data matrix
norm	logical, if TRUE then the normalized RMSE will be returned

**Value**

the RMSE or NRMSE

**See Also**

[impute](#) for the main imputation function, [mr](#) for the misclassification error metric.

**Examples**

```
data(parkinson)
# introduce 10% random missing values into the parkinson data
misssdata <- SimIm(parkinson, 0.1)

# impute the missing values by LASSO
impdata <- impute(misssdata, lmFun = "lassoR")

# calculate the normalised RMSE for the imputation
Rmse(impdata$imp, misssdata, parkinson, norm = TRUE)
```

---

rpartC	<i>classification tree for imputation</i>
--------	---

---

**Description**

classification tree for imputation

**Usage**

```
rpartC(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**See Also**

[rpart](#)

**Examples**

```
data(spect)
missdata <- SimIm(spect, 0.1)

impdata <- impute(spect, cFun = "rpartC")
```

---

SimEval	<i>Evaluate imputation performance by simulation</i>
---------	--

---

**Description**

Evaluate imputation performance by simulation

**Usage**

```
SimEval(data, task = NULL, p = 0.1, n.sim = 100, ini = "mean",
  method = NULL, guess = FALSE, guess.method = NULL, other = NULL,
  verbose = TRUE, seed = 1234)
```

**Arguments**

data	is the complete data matrix that will be used for simulation
task	task type, either be 1 for regression, 2 for classification or 3 for mixed type
p	is the percentage of missing values that will be introduction into data, it has to be a value between 0 and 1
n.sim	the number of simulations, default is 100 times
ini	is the initialization setting for some relevant imputation methods , the default setting is "mean", while "median" and "random" can also be used. See also <a href="#">guess</a>
method	the imputaion method based on variable selection for simulation some other imputation method can be passed to the 'other' argument
guess	logical value, if is TRUE, then <a href="#">guess</a> will be used as the imputation method for simulation
guess.method,	guess type for the guess function. It cannot be NULL if guess is TRUE
other	some other imputation method that is based on variable selection can be used. The requirement for this 'other' method is strict: it receives a data matrix including missing values and returns a complete data matrix.
verbose	logical, if TRUE, additional output information will be provided during iterations, i.e., the method that is using, the iteration number, the convegence difference as compared to the precious iteration. The progression bar will show up irrespective of this option and it can not be got rid of.
seed	set the seed for simulation so simulations using different imputation methods are comparable. The default value is set to 1234, which is not supposed to mean anything. But if 1234 is used, then the seed for simulating the first missing data matrix is 1234, then it sums by one for every subsequent simulationg data matrix.

**Value**

	a list of componentes including
call	the method used for imputation
task	the name of the task
time	computational time
error	the imputation error
conv	the number of iterations to converge

**Examples**

```
data(parkinson)
# WARNING: simulation may take considerable time.

SimEval(parkinson, method = "lassoR")
```

---

`SimIm`*Introduce some missing values into a data matrix*

---

**Description**

This function randomly introduce some amount of missing values into a matrix.

**Usage**

```
SimIm(data, p = 0.1)
```

**Arguments**

<code>data</code>	a data matrix to simulate
<code>p</code>	the percentage of missing values introduced into the data matrix it should be a value between 0 and 1.

**Value**

the same size matrix with simulated missing values.

**Examples**

```
# Create data without missing values as example
simdata <- matrix(rnorm(100), 10, 10)

# Now let's introduce some missing values into the dataset
missingdata <- SimIm(simdata, p = 0.15)

# count the number of missing values afterwards
sum(is.na(missingdata))

#-----

# There is no missing values in the original parkinson data
data(parkinson)

# Let's introduce some missing values into the dataset
misssdata <- SimIm(parkinson, 0.1)

# count the number of missing values afterwards
sum(is.na(misssdata))
```

---

spect

*SPECT Heart Data Set*

---

### Description

The dataset describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. Each of the patients is classified into two categories: normal and abnormal. The database of 267 SPECT image sets (patients) was processed to extract features that summarize the original SPECT images. As a result, 44 continuous feature patterns were created for each patient. The pattern was further processed to obtain 22 binary feature patterns. The CLIP3 algorithm was used to generate classification rules from these patterns. The CLIP3 algorithm generated rules that were 84.0%. SPECT is a good data set for testing ML algorithms; it has 267 instances that are described by 23 binary attributes. In the imputation study, it can be treated as a categorical-only data. For detailed information, please refer to the Source and the Reference.

### Format

A data frame with 266 rows and 23 variables

### Details

- X1. OVERALL\_DIAGNOSIS: 0,1 (class attribute, binary)
- X0. F1: 0,1 (the partial diagnosis 1, binary)
- ...

### Source

<http://archive.ics.uci.edu/ml/datasets/SPECT+Heart>

### References

Kurgan, L.A., Cios, K.J., Tadeusiewicz, R., Ogiela, M. & Goodenday, L.S. 2001 Knowledge Discovery Approach to Automated Cardiac SPECT Diagnosis *Artificial Intelligence in Medicine*, vol. 23:2, pp 149-169

---

stepBackC

*Best subset for classification (backward)*

---

### Description

Best subset variable selection from both forward and backward direction for categorical data

### Usage

stepBackC(x, y)

**Arguments**

x            predictor matrix  
y            response vector

**Value**

a model object that can be used by the [impute](#) function

**See Also**

[step](#), [stepBackR](#)

**Examples**

```
data(spect)
missdata <- SimIm(spect, 0.1)

impdata <- impute(spect, cFun = "stepBackC")
```

---

stepBackR

*Best subset (backward direction) for regression*

---

**Description**

Best subset variable selection (backward direction) for continuous data

**Usage**

```
stepBackR(x, y)
```

**Arguments**

x            predictor matrix  
y            response vector

**Value**

a model object that can be used by the [impute](#) function

**Examples**

```
data(parkinson)
missdata <- SimIm(parkinson, 0.1)

impdata <- impute(missdata, lmFun = "stepBackR")
```

---

stepBothC	<i>Best subset for classification (both direction)</i>
-----------	--

---

**Description**

Best subset variable selection from both forward and backward direction for categorical data

**Usage**

```
stepBothC(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**See Also**

[step](#), [stepBothR](#)

**Examples**

```
data(spect)
misssdata <- SimIm(spect, 0.1)

impdata <- impute(spect, cFun = "stepBothC")
```

---

stepBothR	<i>Best subset for regression (both direction)</i>
-----------	--

---

**Description**

Best subset variable selection from both forward and backward direction for continuous data

**Usage**

```
stepBothR(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**Examples**

```
data(parkinson)
misssdata <- SimIm(parkinson, 0.1)

impdata <- impute(misssdata, lmFun = "stepBothR")
```

---

stepForC

*Best subset for classification (forward direction)*

---

**Description**

Best subset variable selection from both forward and backward direction for categorical data

**Usage**

```
stepForC(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**See Also**

[step](#), [stepForR](#)

**Examples**

```
data(spect)
misssdata <- SimIm(spect, 0.1)

impdata <- impute(spect, cFun = "stepForC")
```

---

stepForR	<i>Best subset (forward direction) for regression</i>
----------	---

---

**Description**

Best subset variable selection (forward direction) for continuous data

**Usage**

```
stepForR(x, y)
```

**Arguments**

x	predictor matrix
y	response vector

**Value**

a model object that can be used by the [impute](#) function

**Examples**

```
data(parkinson)
missdata <- SimIm(parkinson, 0.1)

impdata <- impute(missdata, lmFun = "stepForR")
```

---

tic	<i>Insurance Company Benchmark (COIL 2000) Data Set</i>
-----	---

---

**Description**

This data set used in the CoIL 2000 Challenge contains information on customers of an insurance company. The data consists of 86 variables and includes product usage data and socio-demographic data. Detailed information, please refer to the Source. For imputation study, this dataset can be treated as a mixed-type data.

**Format**

A data frame with 266 rows and 23 variables

**Details**

- V1. a numeric variable
- V2. a categorical variable
- ...

**Source**

[http://archive.ics.uci.edu/ml/datasets/Insurance+Company+Benchmark+\(COIL+2000\)](http://archive.ics.uci.edu/ml/datasets/Insurance+Company+Benchmark+(COIL+2000))

**References**

P. van der Putten and M. van Someren (eds). CoIL Challenge 2000: The Insurance Company Case. Published by Sentient Machine Research, Amsterdam. Also a Leiden Institute of Advanced Computer Science Technical Report 2000-09. June 22, 2000.

# Index

- \* **datasets**
  - parkinson, [13](#)
  - spect, [21](#)
  - tic, [25](#)
- cubist, [3](#)
- CubistR, [3](#)
- cv.glmnet, [8](#)
- Detect, [4](#)
- gbm, [5](#)
- gbmC, [4](#)
- glmboostR, [5](#)
- glmnet, [8](#)
- guess, [6](#), [9](#), [19](#)
- impute, [3](#), [5](#), [6](#), [8](#), [9](#), [14–18](#), [22–25](#)
- imputeR-package, [2](#)
- lassoC, [8](#)
- lassoR, [8](#)
- logisticRidge, [16](#)
- major, [9](#)
- mixError, [10](#)
- mixGuess, [10](#)
- mr, [11](#), [17](#)
- orderbox, [12](#)
- parkinson, [13](#)
- pcr, [14](#)
- pcrR, [13](#)
- plotIm, [14](#)
- plsR, [15](#)
- plsr, [15](#)
- ridgeC, [16](#)
- ridgeR, [16](#)
- Rmse, [17](#)
- rpart, [18](#)
- rpartC, [18](#)
- SimEval, [18](#)
- SimIm, [7](#), [20](#)
- spect, [21](#)
- step, [22–24](#)
- stepBackC, [21](#)
- stepBackR, [22](#), [22](#)
- stepBothC, [23](#)
- stepBothR, [23](#), [23](#)
- stepForC, [24](#)
- stepForR, [24](#), [25](#)
- tic, [25](#)