Package 'mdgc'

March 12, 2021

Type Package

Title Missing Data Imputation Using Gaussian Copulas

Version 0.1.4

Description Provides functions to impute missing values using Gaussian copulas for mixed data types as described by Christoffersen et al. (2021) <arXiv:2102.02642>. The method is related to Hoff (2007) <doi:10.1214/07-AOAS107> and Zhao and Udell (2019) <arXiv:1910.12845> but differs by making a direct approximation of the log marginal likelihood using an extended version of the Fortran code created by Genz and Bretz (2002) <doi:10.1198/106186002394> in addition to also support multinomial variables.

License GPL-2

BugReports https://github.com/boennecd/mdgc/issues

URL https://github.com/boennecd/mdgc

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Depends R (>= 3.5.0)

LinkingTo Rcpp, RcppArmadillo, testthat, BH, psqn

Imports Rcpp

Suggests testthat, catdata

SystemRequirements C++14

NeedsCompilation yes

Author Benjamin Christoffersen [cre, aut]
(<https://orcid.org/0000-0002-7182-1346>),
Alan Genz [cph],
Frank Bretz [cph],
Torsten Hothorn [cph],
R-core [cph],
Ross Ihaka [cph]

Maintainer Benjamin Christoffersen <boennecd@gmail.com>

Repository CRAN

Date/Publication 2021-03-12 21:50:03 UTC

R topics documented:

`mdgc-package` *mdgc: Missing Data imputation using Gaussian Copulas*

Description

The mdgc package is used to estimate Gaussian Copula models for mixed data types (continuous, binary, ordinal, and multinomial) that can be used for imputations. The main function is the `mdgc` function. The rest of the functions in the package give the user access to lower level functions.

Examples are provided at https://github.com/boennecd/mdgc. The package is still in a development stage and the API may change.

Author(s)

Maintainer: Benjamin Christoffersen <boennecd@gmail.com> (ORCID)

Other contributors:

- Alan Genz [copyright holder]
- Frank Bretz [copyright holder]
- Torsten Hothorn [copyright holder]
- R-core <R-core@R-project.org> [copyright holder]
- Ross Ihaka [copyright holder]

References

Christoffersen, B., Clements, M., Humphreys, K., & Kjellström, H. (2021). *Asymptotically Exact and Fast Gaussian Copula Models for Imputation of Mixed Data Types*. https://arxiv.org/abs/2102.02642.

See Also

Useful links:

- https://github.com/boennecd/mdgc
- Report bugs at https://github.com/boennecd/mdgc/issues

`get_mdgc` *Get mdgc Object*

Description

Creates a mdgc object which is needed for estimation of the covariance matrix and the mean vector and to perform imputation.

Usage

```
get_mdgc(dat)
```

Arguments

`dat`	`data.frame` with continuous, multinomial, ordinal, and binary variables.

Details

It is important to use appropriate classes for the `data.frame` columns:

- Continuous variables: should be `numeric`s.
- Binary variables: should be `logical`s.
- Multinomial variables: should be `factor`s.
- Ordinal variables: should be `ordered`.

Value

An object of class `mdgc`. It has the following elements:

`lower,upper,code,multinomial,idx_non_zero_mean`	arguments to pass to `get_mdgc_log_ml`.
`margs`	functions to get `lower` and `upper` bounds for each column of `dat`.
`reals,bins,ords`	indices of continuous, binary, and ordinal variables, respectively.
`truth`	the numeric version of `dat`.
`means`	starting values for the non-zero mean terms (see e.g. `mdgc_fit`).

See Also

`get_mdgc_log_ml`, `mdgc_start_value`

Examples

```
# there is a bug on CRAN's check on Solaris which I have failed to reproduce.
# See https://github.com/r-hub/solarischeck/issues/8#issuecomment-796735501.
# Thus, this example is not run on Solaris
is_solaris <- tolower(Sys.info()[["sysname"]]) == "sunos"

if(!is_solaris){
  # randomly mask data
  set.seed(11)
  masked_data <- iris
  masked_data[matrix(runif(prod(dim(iris))) < .10, NROW(iris))] <- NA

  # use the functions in the package
  library(mdgc)
  obj <- get_mdgc(masked_data)
  print(class(obj))
}
```

`get_mdgc_log_ml` *Get Pointer to C++ Object to Approximate the Log Marginal Likelihood*

Description

Creates a C++ object which is needed to approximate the log marginal likelihood. The object cannot be saved.

Usage

```
get_mdgc_log_ml(object, ...)

## S3 method for class 'mdgc'
get_mdgc_log_ml(object, ...)

## S3 method for class 'data.frame'
get_mdgc_log_ml(object, ...)

## Default S3 method:
get_mdgc_log_ml(
  object,
  lower,
  upper,
  code,
  multinomial,
  idx_non_zero_mean,
  ...
)
```

Arguments

object	mdgc object from get_mdgc or a data.frame to pass to get_mdgc. Ignored by the default method.
...	used to pass arguments to S3 methods.
lower	[# variables]x[# observations] matrix with lower bounds for each variable on the normal scale.
upper	[# variables]x[# observations] matrix with upper bounds for each variable on the normal scale.
code	[# variables]x[# observations] matrix integer code for the each variable on the normal scale. Zero implies an observed value (the value in upper), one implies a missing value, and two implies an interval.
multinomial	list where each element is 3x[# multinomial variables] matrix with multinomial outcomes. The first index is the outcome as an integer code, the second index is the number of categories, and the third index is the index of each multinomial variable (this is zero-based).
idx_non_zero_mean	indices for non-zero mean variables. Indices should be sorted.

Details

Indices are zero-based except the outcome index for multinomial variables.

idx_non_zero_mean indices with terms with code equal to zero (observed values) are ignored.

Value

A Rcpp::XPtr to pass to e.g. mdgc_log_ml.

See Also

mdgc_fit, mdgc_log_ml

Examples

```
# there is a bug on CRAN's check on Solaris which I have failed to reproduce.
# See https://github.com/r-hub/solarischeck/issues/8#issuecomment-796735501.
# Thus, this example is not run on Solaris
is_solaris <- tolower(Sys.info()[["sysname"]]) == "sunos"

if(!is_solaris){
  # randomly mask data
  set.seed(11)
  masked_data <- iris
  masked_data[matrix(runif(prod(dim(iris))) < .10, NROW(iris))] <- NA

  # use the functions in the package
  library(mdgc)
  obj <- get_mdgc(masked_data)
  ptr <- get_mdgc_log_ml(obj)
```

```
}
```

---

mdgc *Perform Model Estimation and Imputation*

Description

A convenience function to perform model estimation and imputation in one call. The learning rate is likely model specific and should be altered. See `mdgc_fit`.

See the README at https://github.com/boennecd/mdgc for examples.

Usage

```
mdgc(
  dat,
  lr = 0.001,
  maxit = 25L,
  batch_size = NULL,
  rel_eps = 0.001,
  method = c("svrg", "adam", "aug_Lagran"),
  seed = 1L,
  epsilon = 1e-08,
  beta_1 = 0.9,
  beta_2 = 0.999,
  n_threads = 1L,
  do_reorder = TRUE,
  abs_eps = -1,
  maxpts = 10000L,
  minvls = 100L,
  verbose = FALSE,
  irel_eps = rel_eps,
  imaxit = maxpts,
  iabs_eps = abs_eps,
  iminvls = 1000L,
  start_val = NULL,
  decay = 0.98,
  conv_crit = 1e-05,
  use_aprx = FALSE
)
```

Arguments

dat	`data.frame` with continuous, multinomial, ordinal, and binary variables.
lr	learning rate.
maxit	maximum number of iteration.

batch_size	number of observations in each batch.
rel_eps	relative error for each marginal likelihood factor.
method	estimation method to use. Can be "svrg", "adam", or "aug_Lagran".
seed	fixed seed to use. Use NULL if the seed should not be fixed.
epsilon	ADAM parameters.
beta_1	ADAM parameters.
beta_2	ADAM parameters.
n_threads	number of threads to use.
do_reorder	logical for whether to use a heuristic variable reordering. TRUE is likely the best option.
abs_eps	absolute convergence threshold for each marginal likelihood factor.
maxpts	maximum number of samples to draw for each marginal likelihood term.
minvls	minimum number of samples.
verbose	logical for whether to print output during the estimation.
irel_eps	relative error for each term in the imputation.
imaxit	maximum number of samples to draw in the imputation.
iabs_eps	absolute convergence threshold for each term in the imputation.
iminvls	minimum number of samples in the imputation.
start_val	starting value for the covariance matrix. Use NULL if unspecified.
decay	the learning rate used by SVRG is given by lr * decay^iteration_number.
conv_crit	relative convergence threshold.
use_aprx	logical for whether to use an approximation of pnorm and qnorm. This may yield a noticeable reduction in the computation time.

Details

It is important that the input for data has the appropriate types and classes. See get_mdgc.

Value

A list with the following entries:

ximp	data.frame with the observed and imputed values.
imputed	output from mdgc_impute.
vcov	the estimated covariance matrix.
mea	the estimated non-zero mean terms.

Additional elements may be present depending on the chosen method. See mdgc_fit.

References

Kingma, D.P., & Ba, J. (2015). *Adam: A Method for Stochastic Optimization*. abs/1412.6980.

Johnson, R., & Zhang, T. (2013). *Accelerating stochastic gradient descent using predictive variance reduction*. In Advances in neural information processing systems.

See Also

get_mdgc, mdgc_start_value, get_mdgc_log_ml, mdgc_fit, mdgc_impute

Examples

```
# there is a bug on CRAN's check on Solaris which I have failed to reproduce.
# See https://github.com/r-hub/solarischeck/issues/8#issuecomment-796735501.
# Thus, this example is not run on Solaris
is_solaris <- tolower(Sys.info()[["sysname"]]) == "sunos"

if(!is_solaris && require(catdata)){
  data(retinopathy)

  # prepare data and save true data set
  retinopathy$RET <- as.ordered(retinopathy$RET)
  retinopathy$SM <- as.logical(retinopathy$SM)

  # randomly mask data
  set.seed(28325145)
  truth <- retinopathy
  for(i in seq_along(retinopathy))
    retinopathy[[i]][runif(NROW(retinopathy)) < .3] <- NA

  cat("\nMasked data:\n")
  print(head(retinopathy, 10))
  cat("\n")

  # impute data
  impu <- mdgc(retinopathy, lr = 1e-3, maxit = 25L, batch_size = 25L,
               rel_eps = 1e-3, maxpts = 5000L, verbose = TRUE,
               n_threads = 1L, method = "svrg")

  # show correlation matrix
  cat("\nEstimated correlation matrix\n")
  print(impu$vcov)

  # compare imputed and true values
  cat("\nObserved;\n")
  print(head(retinopathy, 10))
  cat("\nImputed values:\n")
  print(head(impu$ximp, 10))
  cat("\nTruth:\n")
  print(head(truth, 10))

  # using augmented Lagrangian method
  cat("\n")
  impu_aug <- mdgc(retinopathy, maxit = 25L, rel_eps = 1e-3,
                   maxpts = 5000L, verbose = TRUE,
                   n_threads = 1L, method = "aug_Lagran")

  # compare the log-likelihood estimate
```

```
  obj <- get_mdgc_log_ml(retinopathy)
  cat(sprintf(
    "Maximum log likelihood with SVRG vs. augmented Lagrangian:\n  %.2f vs. %.2f\n",
    mdgc_log_ml(obj, vcov = impu    $vcov, mea = impu    $mea, rel_eps = 1e-3),
    mdgc_log_ml(obj, vcov = impu_aug$vcov, mea = impu_aug$mea, rel_eps = 1e-3)))

  # show correlation matrix
  cat("\nEstimated correlation matrix (augmented Lagrangian)\n")
  print(impu_aug$vcov)

  cat("\nImputed values (augmented Lagrangian):\n")
  print(head(impu_aug$ximp, 10))
}
```

`mdgc_fit` *Estimate the Model Parameters*

Description

Estimates the covariance matrix and the non-zero mean terms. The `lr` parameter and the `batch_size` parameter are likely data dependent. Convergence should be monitored e.g. by using `verbose = TRUE` with `method = "svrg"`.

See the README at https://github.com/boennecd/mdgc for examples.

Usage

```
mdgc_fit(
  ptr,
  vcov,
  mea,
  lr = 0.001,
  rel_eps = 0.001,
  maxit = 25L,
  batch_size = NULL,
  method = c("svrg", "adam", "aug_Lagran"),
  seed = 1L,
  epsilon = 1e-08,
  beta_1 = 0.9,
  beta_2 = 0.999,
  n_threads = 1L,
  do_reorder = TRUE,
  abs_eps = -1,
  maxpts = 10000L,
  minvls = 100L,
  verbose = FALSE,
  decay = 0.98,
```

```
  conv_crit = 1e-06,
  use_aprx = FALSE,
  mu = 1,
  lambda = NULL
)
```

Arguments

`ptr`	returned object from `get_mdgc_log_ml`.
`vcov, mea`	starting value for the covariance matrix and the non-zero mean entries.
`lr`	learning rate.
`rel_eps`	relative error for each marginal likelihood factor.
`maxit`	maximum number of iteration.
`batch_size`	number of observations in each batch.
`method`	estimation method to use. Can be `"svrg"`, `"adam"`, or `"aug_Lagran"`.
`seed`	fixed seed to use. Use `NULL` if the seed should not be fixed.
`epsilon, beta_1, beta_2`	ADAM parameters.
`n_threads`	number of threads to use.
`do_reorder`	logical for whether to use a heuristic variable reordering. `TRUE` is likely the best option.
`abs_eps`	absolute convergence threshold for each marginal likelihood factor.
`maxpts`	maximum number of samples to draw for each marginal likelihood term.
`minvls`	minimum number of samples.
`verbose`	logical for whether to print output during the estimation.
`decay`	the learning rate used by SVRG is given by `lr * decay^iteration_number`.
`conv_crit`	relative convergence threshold.
`use_aprx`	logical for whether to use an approximation of `pnorm` and `qnorm`. This may yield a noticeable reduction in the computation time.
`mu`	starting value for the penalty in the augmented Lagrangian method.
`lambda`	starting values for the Lagrange multiplier estimates. `NULL` yields a default.

Value

An `list` with the following elements:

`result`	`list` with two elements: `vcov` is the estimated covariance matrix and `mea` is the estimated non-zero mean terms.
`estimates`	If present, the estimated parameters after each iteration.
`fun_vals`	If present, the output of `mdgc_log_ml` after each iteration.
`mu,lambda`	If present, the `mu` and `lambda` values at the end.

The elements that may be present depending on the chosen `method`.

References

Kingma, D.P., & Ba, J. (2015). *Adam: A Method for Stochastic Optimization*. abs/1412.6980.

Johnson, R., & Zhang, T. (2013). *Accelerating stochastic gradient descent using predictive variance reduction*. In Advances in neural information processing systems.

See Also

mdgc_log_ml, mdgc_start_value, mdgc_impute.

Examples

```
# there is a bug on CRAN's check on Solaris which I have failed to reproduce.
# See https://github.com/r-hub/solarischeck/issues/8#issuecomment-796735501.
# Thus, this example is not run on Solaris
is_solaris <- tolower(Sys.info()[["sysname"]]) == "sunos"

if(!is_solaris){
  # randomly mask data
  set.seed(11)
  masked_data <- iris
  masked_data[matrix(runif(prod(dim(iris))) < .10, NROW(iris))] <- NA

  # use the functions in the package
  library(mdgc)
  obj <- get_mdgc(masked_data)
  ptr <- get_mdgc_log_ml(obj)
  start_vals <- mdgc_start_value(obj)

  fit <- mdgc_fit(ptr, start_vals, obj$means, rel_eps = 1e-2, maxpts = 10000L,
                  minvls = 1000L, use_aprx = TRUE, batch_size = 100L, lr = .001,
                  maxit = 100L, n_threads = 2L)
  print(fit$result$vcov)
  print(fit$result$mea)
}
```

---

mdgc_impute *Impute Missing Values*

Description

Imputes missing values given a covariance matrix and mean vector using a similar quasi-random numbers method as mdgc_log_ml.

Usage

```
mdgc_impute(
  object,
  vcov,
  mea,
  rel_eps = 0.001,
  maxit = 10000L,
  abs_eps = -1,
  n_threads = 1L,
  do_reorder = TRUE,
  minvls = 1000L,
  use_aprx = FALSE
)
```

Arguments

object	returned object from get_mdgc.
vcov	covariance matrix to condition on in the imputation.
mea	vector with non-zero mean entries to condition on.
rel_eps	relative convergence threshold for each term in the approximation.
maxit	maximum number of samples
abs_eps	absolute convergence threshold for each term in the approximation.
n_threads	number of threads to use.
do_reorder	logical for whether to use a heuristic variable reordering. TRUE is likely the best option.
minvls	minimum number of samples.
use_aprx	logical for whether to use an approximation of pnorm and qnorm. This may yield a noticeable reduction in the computation time.

Value

A list of lists with imputed values for the continuous variables and a vector with probabilities for each level for the ordinal, binary, and multinomial variables.

Examples

```
# there is a bug on CRAN's check on Solaris which I have failed to reproduce.
# See https://github.com/r-hub/solarischeck/issues/8#issuecomment-796735501.
# Thus, this example is not run on Solaris
is_solaris <- tolower(Sys.info()[["sysname"]]) == "sunos"

if(!is_solaris){
  # randomly mask data
  set.seed(11)
  masked_data <- iris
  masked_data[matrix(runif(prod(dim(iris))) < .10, NROW(iris))] <- NA
```

```
  # use the functions in the package
  library(mdgc)
  obj <- get_mdgc(masked_data)
  ptr <- get_mdgc_log_ml(obj)
  start_vals <- mdgc_start_value(obj)

  fit <- mdgc_fit(ptr, start_vals, obj$means, rel_eps = 1e-2, maxpts = 10000L,
                  minvls = 1000L, use_aprx = TRUE, batch_size = 100L, lr = .001,
                  maxit = 100L, n_threads = 2L)

  # impute using the estimated values
  imputed <- mdgc_impute(obj, fit$result$vcov, fit$result$mea, minvls = 1000L,
                         maxit = 10000L, n_threads = 2L, use_aprx = TRUE)
  print(imputed[1:5]) # first 5 observations
  print(head(masked_data, 5)) # observed
  print(head(iris       , 5)) # truth
}
```

---

mdgc_log_ml *Evaluate the Log Marginal Likelihood and Its Derivatives*

Description

Approximates the log marginal likelihood and the derivatives using randomized quasi-Monte Carlo. The method uses a generalization of the Fortran code by Genz and Bretz (2002).

Mean terms for observed continuous variables are always assumed to be zero.

The returned log marginal likelihood is not a proper log marginal likelihood if the `ptr` object is constructed from a mdgc object from `get_mdgc` as it does not include the log of the determinants of the Jacobians for the transformation of the continuous variables.

Usage

```
mdgc_log_ml(
  ptr,
  vcov,
  mea,
  rel_eps = 0.01,
  n_threads = 1L,
  comp_derivs = FALSE,
  indices = NULL,
  do_reorder = TRUE,
  maxpts = 100000L,
  abs_eps = -1,
  minvls = 100L,
  use_aprx = FALSE
)
```

Arguments

ptr	object returned by get_mdgc_log_ml.
vcov	covariance matrix.
mea	vector with non-zero mean entries.
rel_eps	relative error for each marginal likelihood factor.
n_threads	number of threads to use.
comp_derivs	logical for whether to approximate the gradient.
indices	integer vector with which terms (observations) to include. Must be zero-based. NULL yields all observations.
do_reorder	logical for whether to use a heuristic variable reordering. TRUE is likely the best option.
maxpts	maximum number of samples to draw for each marginal likelihood term.
abs_eps	absolute convergence threshold for each marginal likelihood factor.
minvls	minimum number of samples.
use_aprx	logical for whether to use an approximation of pnorm and qnorm. This may yield a noticeable reduction in the computation time.

Value

A numeric vector with a single element with the log marginal likelihood approximation. Two attributes are added if comp_derivs is TRUE: "grad_vcov" for the derivative approximation with respect to vcov and "grad_mea" for the derivative approximation with respect to mea.

References

Genz, A., & Bretz, F. (2002). *Comparison of Methods for the Computation of Multivariate t Probabilities*. Journal of Computational and Graphical Statistics.

Genz, A., & Bretz, F. (2008). *Computation of Multivariate Normal and t Probabilities*. Springer-Verlag, Heidelberg.

See Also

mdgc_fit

Examples

```
# there is a bug on CRAN's check on Solaris which I have failed to reproduce.
# See https://github.com/r-hub/solarischeck/issues/8#issuecomment-796735501.
# Thus, this example is not run on Solaris
is_solaris <- tolower(Sys.info()[["sysname"]]) == "sunos"

if(!is_solaris){
  # randomly mask data
  set.seed(11)
  masked_data <- iris
  masked_data[matrix(runif(prod(dim(iris))) < .10, NROW(iris))] <- NA
```

```
  # use the functions in the package
  library(mdgc)
  obj <- get_mdgc(masked_data)
  ptr <- get_mdgc_log_ml(obj)
  start_vals <- mdgc_start_value(obj)
  print(mdgc_log_ml(ptr, start_vals, obj$means))
  print(mdgc_log_ml(ptr, start_vals, obj$means, use_aprx = TRUE))
  print(mdgc_log_ml(ptr, start_vals, obj$means, use_aprx = TRUE,
                    comp_derivs = TRUE))
}
```

`mdgc_start_value` *Get Starting Value for the Covariance Matrix Using a Heuristic*

Description

Uses a heuristic to get starting values for the covariance matrix. These can be passed e.g. to `mdgc_fit`.

Usage

```
mdgc_start_value(object, ...)

## S3 method for class 'mdgc'
mdgc_start_value(object, ...)

## Default S3 method:
mdgc_start_value(
  object,
  lower,
  upper,
  code,
  multinomial,
  idx_non_zero_mean,
  mea,
  n_threads = 1L,
  ...
)
```

Arguments

`object`	mdgc object from `get_mdgc`. Ignored by the default method.
`...`	used to pass arguments to S3 methods.
`lower`	[# variables]x[# observations] matrix with lower bounds for each variable on the normal scale.

upper	[# variables]x[# observations] matrix with upper bounds for each variable on the normal scale.
code	[# variables]x[# observations] matrix integer code for the each variable on the normal scale. Zero implies an observed value (the value in upper), one implies a missing value, and two implies an interval.
multinomial	list where each element is 3x[# multinomial variables] matrix with multinomial outcomes. The first index is the outcome as an integer code, the second index is the number of categories, and the third index is the index of each multinomial variable (this is zero-based).
idx_non_zero_mean	indices for non-zero mean variables. Indices should be sorted.
mea	vector with non-zero mean entries.
n_threads	number of threads to use.

Value

The starting value for the covariance matrix.

Examples

```
# there is a bug on CRAN's check on Solaris which I have failed to reproduce.
# See https://github.com/r-hub/solarischeck/issues/8#issuecomment-796735501.
# Thus, this example is not run on Solaris
is_solaris <- tolower(Sys.info()[["sysname"]]) == "sunos"

if(!is_solaris){
  # randomly mask data
  set.seed(11)
  masked_data <- iris
  masked_data[matrix(runif(prod(dim(iris))) < .10, NROW(iris))] <- NA

  # use the functions in the package
  library(mdgc)
  obj <- get_mdgc(masked_data)
  ptr <- get_mdgc_log_ml(obj)
  start_vals <- mdgc_start_value(obj)
  print(start_vals) # starting value for the covariance matrix
}
```

Index