Package ‘clusterMI’

March 12, 2024

Description Allows clustering of incomplete observations by addressing missing values using multiple imputation. For achieving this goal, the methodology consists in three steps.
I) Missing data imputation using dedicated models. Four multiple imputation methods are proposed, two are based on joint modelling and two are fully sequential methods.
II) cluster analysis of imputed data sets. Six clustering methods are available (distances-based or model-based), but custom methods can also be easily used.
III) Partition pooling, The set of partitions is aggregated using Non-negative Matrix Factorization based method. An associated instability measure is computed by bootstrap. Among applications, this instability measure can be used to choose a number of clusters with missing values.
The package also proposes several diagnostic tools to tune the number of imputed data sets, to tune the number of iterations in fully sequential imputation, to check the fit of imputation models, etc.

License GPL-2 | GPL-3

Depends R (>= 3.5.0)

Imports stats, graphics, parallel, mice, micemd, mclust, mix, fpc, usedist, knockoff, withr, glmnet, cluster, ClusterR, FactoMineR, diceR, NPBayesImputeCat, e1071, Rfast, cat, utils, lattice, reshape2, methods, Rcpp

RoxygenNote 7.3.1

Suggests knitr, rmarkdown, stargazer, VIM, missMDA, clustrd, clusterCrit, ggplot2, bookdown

VignetteBuilder knitr

LazyData true

LinkingTo Rcpp, RcppArmadillo

RcppModules IO_module

NeedsCompilation yes
Description

`clusterMI` is a R package to perform clustering with missing values. For achieving this goal, multiple imputation is used. The package offers various multiple imputation methods dedicated to clustered individuals, as discussed in Audigier et al. (2021) [arXiv:2106.04424]. In addition, it allows pooling results both in terms of partition and instability, as proposed in Audigier and Niang (2022) [doi:10.1007/s11634-022-00519-1]. Among applications, this instability measure can be used to choose a number of clusters with missing values.

References

**Examples**

```r
# Load data
require(parallel)
set.seed(123456)
ref <- wine$cult
nb.clust <- 3
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

# Imputation
m <- 5 # Number of imputed data sets. Should be larger in practice
res.imp <- imputedata(data.na = wine.na, nb.clust = nb.clust, m = m)

# Cluster analysis by kmeans and pooling
nnodes <- 2 # Number of CPU cores for parallel computing
res.pool <- clusterMI(res.imp, nnodes = nnodes)

res.pool$instability
table(ref, res.pool$part)

# Choice of nb.clust
res.nbclust <- choosenbclust(res.pool)
res.nbclust$nb.clust
```

---

**chooseB**

*Diagnostic plot for the number of iterations used in the varselbest function*

---

**Description**

chooseB plots the proportion of times an explanatory variable is selected according to the number of iterations (B).

**Usage**

```r
chooseB(
  res.varselbest,
  gridB = NULL,
  xlim = NULL,
  plotvars = NULL,
  cex = 0.2,
  type = "b",
```

---

`chooseB` requires the `imputedata` and `clusterMI` functions from the `varselMI` package.
```r
xlab = "B",
ylab = "proportion",
graph = TRUE,
pch = 16
```

### Arguments

- `res.varselbest`: an output from the varselbest function
- `gridB`: a grid for the number of iterations. By default, the grid is tuned to `1:B` where `B` is the argument used in varselbest.
- `xlim`: the x limits (x1, x2) of the plot
- `plotvars`: index of variables for which a curve is plotted
- `cex`: a numerical vector giving the amount by which plotting characters and symbols should be scaled relative to the default
- `type`: what type of plot should be drawn
- `xlab`: a title for the x axis
- `ylab`: a title for the y axis
- `graph`: a boolean. If FALSE, no graphics are plotted. Default value is TRUE
- `pch`: an integer

### Details

varselbest performs variable selection on random subsets of variables and, then, combines them to recover which explanatory variables are related to the response, following Bar-Hen and Audigier (2022) [<doi:10.1080/00949655.2022.2070621>]. More precisely, the outline of the algorithm are as follows: let consider a random subset of `sizeblock` among `p` variables. Then, any selection variables scheme can be applied. By resampling `B` times, a sample of size `sizeblock` among the `p` variables, we may count how many times a variable is considered as significantly related to the response and how many times it is not. The number of iterations `B` should be large so that the proportion of times a variable is selected becomes stable. chooseB plots the values of proportion according to the number of iterations.

### Value

A list of matrices where each row corresponds to the vector of proportions (for all explanatory variables) obtained for a given value of `B`

### References


### See Also

- `varselbest`
Examples

```r
data(wine)

require(parallel)
ref <- wine$cult
nb.clust <- 3
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

nnodes <- 2 # Number of CPU cores for parallel computing
B <- 80 # Number of iterations for variable selection

# variable selection
res.varsel <- varselbest(data.na = wine.na,
                         listvar = "alco",
                         B = B,
                         nnodes = nnodes,
                         nb.clust = nb.clust,
                         graph = FALSE)

# convergence
res.chooseB <- chooseB(res.varsel)
```

**Description**

For an object generated by the function `clusterMI`, the `choosem` function browses the sequence of the contributory partitions and computes the consensus partition at each step. Then, the rand index between successive consensus partitions is plotted.

**Usage**

`choosem(output, graph = TRUE)`

**Arguments**

- `output` an output from the `clusterMI` function
- `graph` a boolean indicating if a graphic is plotted
Details

The number of imputed datasets ($m$) should be sufficiently large to improve the partition accuracy. The `choosem` function can be used to check if this number is suitable. This function computes the consensus partition by considering only the first imputed datasets. By this way, a sequence of $m$ consensus partitions is obtained. Then, the rand index between successive partitions is computed and reported in a graph. The rand index measures the proximity between two partitions. If the rand index between the last consensus partitions of the sequence reaches its maximum values (1), then it means last imputed dataset does not modify the consensus partition. Consequently, the number of imputed datasets can be considered as sufficiently large.

Value

A list of two objects

- `part` m-columns matrix that contains in column p the consensus partition using only the p first imputed datasets
- `rand` a m-1 vector given the rand index between the m successive consensus partitions

References


See Also

`clusterMI`, `imputedata`

Examples

data(wine)

set.seed(123456)
ref <- wine$cult
nb.clust <- 3
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

#imputation
m <- 5 # number of imputed data sets. Should be larger in practice
res.imp <- imputedata(data.na = wine.na, nb.clust = nb.clust, m = m)

#pooling
res.pool <- clusterMI(res.imp, instability = FALSE)
res.choosem <- choosem(res.pool)
choosemaxit

Diagnostic plot for the number of iterations used in sequential imputation methods

Description

The choosemaxit function plots the within and between variance for each variable (specified in plotvars) against the iteration number for each of the replications (specified in plotm).

Usage

```r
choosemaxit(
  output,
  plotvars = NULL,
  plotm = 1:5,
  cex = 0.3,
  pch = 16,
  type = "b",
  xlab = "iterations",
  ylab = "var",
  layout = NULL
)
```

Arguments

- **output**: an output from the imputedata function
- **plotvars**: index of variables for which a curve is plotted
- **plotm**: a vector indicating which imputed datasets must be plotted
- **cex**: a numerical vector giving the amount by which plotting characters and symbols should be scaled relative to the default
- **pch**: a vector of plotting characters or symbols
- **type**: what type of plot should be drawn
- **xlab**: a title for the x axis
- **ylab**: a title for the y axis
- **layout**: a vector of the form c(nrow, ncol)

Value

No return value
choosenbclust

Tune the number of clusters according to the partition instability

Description

choosenbclust reports the cluster instability according to the number of clusters chosen.

Usage

choosenbclust(output, grid = 2:5, graph = TRUE, verbose = TRUE, nnodes = NULL)

Arguments

- **output**: an output from the clusterMI function
- **grid**: a vector indicating the grid of values tested for nb.clust. By default 2:5
- **graph**: a boolean indicating if a graphic is plotted
- **verbose**: if TRUE, choosenbclust will print messages on console
- **nnodes**: number of CPU cores for parallel computing. By default, the value used in the call to the clusterMI function

Details

The choosenbclust function browses a grid of values for the number of clusters and for each one imputes the data and computes the instability.

Value

- a list of two objects
  - **nb.clust**: the number of clusters in grid minimizing the instability
  - **crit**: a vector indicating the instability for each value in the grid
 chooser

References

Audigier, V. and Niang, N., Clustering with missing data: which equivalent for Rubin’s rules?

See Also

imputedata

Examples

data(wine)

require(parallel)
set.seed(123456)
ref <- wine$cult
nb.clust <- 3
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

# imputation
res.imp <- imputedata(data.na=wine.na, nb.clust = nb.clust, m = 5)

# pooling
nnodes <- 2 # number of CPU cores for parallel computing
res.pool <- clusterMI(res.imp, nnodes = nnodes, instability = FALSE)

# choice of nb.clust
choosenbclust(res.pool)

doctor

choose

K-fold cross-validation for specifying threshold $r$

Description

chooser returns a list specifying the optimal threshold $r$ for each outcome as well as the associated set of explanatory variables selected, and the cross-validation error for each value of the grid

Usage

chooser(
  res.varsel,
  K = 10,
  seed = 12345,
  listvar = NULL,
)
grid.r = seq(0, 1, 1/1000),
graph = TRUE,
printflag = TRUE,
nb.clust = NULL,
nnodes = NULL,
sizeblock = NULL,
method.select = NULL,
B = NULL,
modelNames = NULL,
nbvarused = NULL,
path.outfile = NULL
)

Arguments

res.varsel an output from the varselbest function
K an integer given the number of folds
seed a integer
listvar a vector of characters specifying variables (outcomes) for which cross-validation should be done. By default, all variables that have been considered for varselbest are used.
grid.r a grid for the tuning parameter r
graph a boolean. If TRUE, cross-validation results are printed
printflag a boolean. If TRUE, messages are printed
nb.clust number of clusters. By default, the same as the one used in varselbest
nnodes an integer specifying the number of nodes for parallel computing. By default, the same as the one used in varselbest
sizeblock number of sampled variables at each iteration. By default, the same as the one used in varselbest
method.select variable selection method used. By default, the same as the one used in varselbest
B number of iterations. By default, the same as the one used in varselbest
modelNames mixture model specification for imputation of subsets. By default, the same as the one used in varselbest
nbvarused a maximal number of selected variables (can be required for a dataset with a large number of variables)
path.outfile a path for message redirection

Details

varselbest performs variable selection on random subsets of variables and, then, combines them to recover which explanatory variables are related to the response. More precisely, the outline of the algorithm are as follows: let consider a random subset of sizeblock among p variables. By choosing sizeblock small, this subset is low dimensional, allowing treatment of missing values by standard imputation method for clustered individuals. Then, any selection variable scheme can be applied (lasso, stepwise and knockoff are proposed by tuning the method.select argument). By
resampling \( B \) times, a sample of size \( \text{sizeblock} \) among the \( p \) variables, we may count how many times, a variable is considered as significantly related to the response and how many times it is not. We need to define a threshold \( (r) \) to conclude if a given variable is significantly related to the response. \texttt{chooser} aims at finding the optimal value for the threshold \( r \) using Kfold cross-validation.

**Value**

A list where each object refers to an outcome variable called in the \texttt{listvar} argument. Each element is composed of three objects

\[
\begin{align*}
\text{r} & \quad \text{the optimal value for the threshold} \\
\text{error} & \quad \text{the cross-validation error for each value in } \text{grid.r} \\
\text{selection} & \quad \text{the subset of selected variables for the optimal threshold}
\end{align*}
\]

**References**


**Examples**

data(wine)

require(parallel)
set.seed(123456)
ref <- wine$cult
nb.clust <- 3
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

nnodes <- 2 # parallel::detectCores()
\( B \) <- 100 # Number of iterations
\( m \) <- 5 # Number of imputed data sets

# variables selection for incomplete variable "alco"
listvar <- "alco"
res.varsel <- varselbest(data.na = wine.na,
                         nb.clust = nb.clust,
                         listvar = listvar,
                         \( B \) = \( B \),
                         nnodes = nnodes)

#predictormatrix with the default threshold value
predictmat <- res.varsel$predictormatrix

# r optimal and associated predictor matrix
clusterMI <- chooser(res.varsel = res.varsel)
thresh <- res.choose$[listvar]$r
predictmat[protein, res.varsel$proportion$[listvar,<thresh]<-0

# imputation
res.imp.select <- imputedata(data.na = wine.na, method = "FCS-homo",
nb.clust = nb.clust, predictmat = predictmat, m = m)

clusterMI
Cluster analysis and pooling after multiple imputation

Description

From a list of imputed datasets clusterMI performs cluster analysis on each imputed data set, estimates the instability of each partition using bootstrap (following Fang, Y. and Wang, J., 2012 <doi:10.1016/j.csda.2011.09.003>) and pools results as proposed in Audigier and Niang (2022) <doi:10.1007/s11634-022-00519-1>.

Usage

clusterMI(
  output,
  method.clustering = "kmeans",
  method.consensus = "NMF",
  scaling = TRUE,
  nb.clust = NULL,
  Cboot = 50,
  method.agnes = "average",
  modelNames = NULL,
  modelName.hc = "VVV",
  nstart.kmeans = 100,
  iter.max.kmeans = 10,
  m.cmeans = 2,
  samples.clara = 500,
  nnodes = 1,
  instability = TRUE,
  verbose = TRUE,
  nmf.threshold = 10^(-5),
  nmf.nstart = 100,
  nmf.early_stop_iter = 10,
  nmf.initializer = "random",
  nmf.batch_size = NULL,
  nmf.iter.max = 50
)
Arguments

output: an output from the imputedata function

method.clustering: a single string specifying the clustering algorithm used ("kmeans", "pam", "clara", "agnes" or "mixture","cmeans")

method.consensus: a single string specifying the consensus method used to pool the contributory partitions ("NMF" or "CSPA")

scaling: boolean. If TRUE, variables are scaled. Default value is TRUE

nb.clust: an integer specifying the number of clusters

Cboot: an integer specifying the number of bootstrap replications. Default value is 50

method.agnes: character string defining the clustering method for hierarchical clustering (required only if method.clustering = "agnes")

modelNames: character string indicating the models to be fitted in the EM phase of clustering (required only if method.clustering = "mixture"). By default modelNames = NULL.

modelName.hc: A character string indicating the model to be used in model-based agglomerative hierarchical clustering.(required only if method.clustering = "mixture"). By default modelName.hc = "VVV".

nstart.kmeans: how many random sets should be chosen for kmeans initialization. Default value is 100 (required only if method.clustering = "kmeans")

iter.max.kmeans: how many iterations should be chosen for kmeans. Default value is 10 (required only if method.clustering = "kmeans")

m.cmeans: degree of fuzzification in cmeans clustering. By default m.cmeans = 2

samples.clara: number of samples to be drawn from the dataset when performing clustering using clara algorithm. Default value is 500.

nnodes: number of CPU cores for parallel computing. By default, nnodes = 1

instability: a boolean indicating if cluster instability must be computed. Default value is TRUE

verbose: a boolean. If TRUE, a message is printed at each step. Default value is TRUE

nmf.threshold: Default value is 10^(-5),
nmf.nstart: Default value is 100,
nmf.early_stop_iter: Default value is 10,
nmf.initializer: Default value is ‘random’,
nmf.batch_size: Default value is 20,
nmf.iter.max: Default value is 50
Details

clusterMI performs cluster analysis (according to the method.clustering argument) and pooling after multiple imputation. For achieving this goal, the clusterMI function uses as an input an output from the imputedata function and then

1. applies the cluster analysis method on each imputed data set
2. pools contributory partitions using non-negative matrix factorization
3. computes the instability of each partition by bootstrap
4. computes the total instability

Step 1 can be tuned by specifying the cluster analysis method used (method.clustering argument). If method.clustering = "kmeans" or "pam", then the number of clusters can be specified by tuning the nb.clust argument. By default, the same number as the one used for imputation is used. The number of random initializations can also be tuned through the nstart.kmeans argument. If method.clustering = "agnes" (hierarchical clustering), the method used can be specified (see agnes). By default "average" is used. Furthermore, the number of clusters can be specified, but it can also be automatically chosen if nb.clust < 0. If method.clustering = "mixture" (model-based clustering using gaussian mixture models), the model to be fitted can be tuned by modifying the modelNames argument (see Mclust). If method.clustering = "cmeans" (clustering using the fuzzy c-means algorithm), then the fuzziness parameter can be modified by tuning the m.cmeans argument. By default, m.cmeans = 2.

Step 2 performs consensus clustering by Non-Negative Matrix Factorization, following Li and Ding (2007) <doi:10.1109/ICDM.2007.98>.

Step 3 applies the nselectboot function on each imputed data set and returns the instability of each cluster obtained at step 1. The method is based on bootstrap sampling, followong Fang, Y. and Wang, J. (2012) <doi:10.1016/j.csda.2011.09.003>. The number of iterations can be tuned using the Cboot argument.

Step 4 averages the previous instability measures given a within instability (\(\bar{U}\)), computes a between instability (\(B\)) and a total instability (\(T = B + \bar{U}\)). See Audigier and Niang (2022) <doi:10.1007/s11634-022-00519-1> for details.

All steps can be performed in parallel by specifying the number of CPU cores (nnodes argument). Steps 3 and 4 are more time consuming. To compute only steps 1 and 2 use instability = FALSE.

Value

A list with three objects

part the consensus partition

instability a list of four objects: \(U\) the within instability measure for each imputed data set, \(\bar{U}\) the associated average, \(B\) the between instability measure, \(T\) the total instability measure

call the matching call
fastnmf

References


See Also

agnes, nselectboot, Mclust, imputedata, cmeans

Examples

data(wine)

require(parallel)
set.seed(123456)
ref <- wine$cult
nb.clust <- 3
m <- 5 # number of imputed data sets. Should be larger in practice
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

#imputation
res.imp <- imputedata(data.na = wine.na, nb.clust = nb.clust, m = m)

#analysis by kmeans and pooling
nnodes <- 2 # parallel::detectCores()
res.pool <- clusterMI(res.imp, nnodes = nnodes)
res.pool$instability
table(ref, res.pool$part)

fastnmf
Consensus clustering using non-negative matrix factorization

Description

From a list of partitions fastnmf pools partition as proposed in Li and Ding (2007) <doi:10.1109/ICDM.2007.98>.
Usage

fastnmf(
    listpart,
    nb.clust,
    threshold = 10^(-5),
    printflag = TRUE,
    nstart = 100,
    early_stop_iter = 10,
    initializer = "random",
    batch_size = NULL,
    iter.max = 50
)

Arguments

listpart       a list of partitions  
nb.clust       an integer specifying the number of clusters  
threshold      a real specifying when the NMF algorithm is stopped. Default value is 10^(-5)  
printflag      a boolean. If TRUE, nmf will print messages on console. Default value is TRUE  
nstart         how many random sets should be chosen for kmeans initialization. Default value is 100  
early_stop_iter continue that many iterations after calculation of the best within-cluster-sum-of-squared-error. Default value is 10. See MiniBatchKmeans help page.  
initializer    the method of initialization. One of, optimal_init, quantile_init, kmeans++ and random. See MiniBatchKmeans help page.  
batch_size     the size of the mini batches for kmeans clustering. Default value is NULL.  
iter.max       the maximum number of iterations allowed for kmeans. Default value is 50  

Details

fastnmf performs consensus clustering using non-negative matrix factorization following Li and Ding (2007) <doi:10.1109/ICDM.2007.98>. The set of partitions that are aggregated needs to be given as a list where each element is a vector of numeric values. Note that the number of classes for each partition can vary. The number of classes for the consensus partition should be given using the nb.clust argument. The NMF algorithm is iterative and required an initial partition. This latter is based on kmeans clustering on the average of connectivity matrices. If batchsize is NULL, then kmeans clustering is performed using nstart initial values and iter.max iterations. Otherwise, Mini Batch Kmeans is used. This algorithm could be faster than kmeans if the number of individuals is large.

Value

a list of 5 objects  
Htilde               A fuzzy disjunctive table  
S                   A positive matrix
**fastnmf**

- **Mtilde**: The average of connectivity matrices
- **crit**: A vector with the optimized criterion at each iteration
- **cluster**: The consensus partition in nb.clust classes

**References**


**See Also**

- `kmeans`
- `MiniBatchKmeans`

**Examples**

```r
data(wine)
require(clustrd)
set.seed(123456)
ref <- wine$cult
nb.clust <- 3
m <- 3 # number of imputed data sets. Should be larger in practice
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

#imputation
res.imp <- imputedata(data.na = wine.na, nb.clust = nb.clust, m = m)

#analysis using reduced kmeans
## apply the cluspca function on each imputed data set
res.ana.rkm <- lapply(res.imp$res.imp, FUN = cluspca, nclus = nb.clust, ndim = 2, method = "RKM")

## extract the set of partitions (under "list" format)
res.ana.rkm <- lapply(res.ana.rkm, FUN = function(x) x$cluster)

# pooling by NMF
res.pool.rkm <- fastnmf(res.ana.rkm, nb.clust = nb.clust)$clust
```
imputedata returns a list of imputed datasets by using imputation methods dedicated to individuals clustered in (unknown) groups.

Usage:
```
imputedata(
data.na,  
method = "JM-GL",  
nb.clust = NULL,  
m = 20,  
maxit = 50,  
Lstart = 100,  
L = 20,  
method.mice = NULL,  
predictmat = NULL,  
verbose = TRUE,  
seed = 1234,  
bootstrap = FALSE  
)
```

Arguments:
- **data.na**: an incomplete dataframe
- **method**: a single string specifying the imputation method used among "FCS-homo", "FCS-hetero", "JM-DP", "JM-GL". By default method = "JM-GL". See the details section.
- **nb.clust**: number of clusters
- **m**: number of imputed datasets. By default, m = 20.
- **maxit**: number of iterations for FCS methods (only used for method = FCS-homo or method = FCS-hetero)
- **Lstart**: number of iterations for the burn-in period (only used if method = "JM-DP" or "JM-GL")
- **L**: number of skipped iterations to keep one imputed data set after the burn-in period (only used if method = "JM-DP" or "JM-GL")
- **method.mice**: a vector of strings (or a single string) giving the imputation method for each variable (only used for method = FCS-homo or method = FCS-hetero). Default value is 'norm' for FCS-homo and 'mice.impute.2l.jomo' for FCS-hetero
- **predictmat**: predictor matrix used for FCS imputation (only used for method = FCS-homo or method = FCS-hetero)
verbose a boolean. If TRUE, a message is printed at each iteration. Use \texttt{verbose = FALSE} for silent imputation

\texttt{seed} a positive integer initializing the random generator

\texttt{bootstrap} a boolean. Use \texttt{bootstrap = TRUE} for proper imputation with FCS methods (Mclust sometimes fails with multiple points)

\section*{Details}
The \texttt{imputedata} offers various multiple imputation methods dedicated to clustered individuals. In particular, two fully conditional imputation methods are proposed (\texttt{FCS-homo} and \texttt{FCS-hetero}) which essentially differ by the assumption about the covariance in each cluster (constant or not respectively). The imputation requires a pre-specified number of clusters (\texttt{nb.clust}). See \texttt{choosenbclust} if this number is unknown. The \texttt{imputedata} function alternates clustering and imputation given the partition of individuals. When the clustering is performed, the function calls the \texttt{mice} function from the \texttt{mice} R package to perform imputation. The \texttt{mice} package proposes various methods for imputation which can be specified by tuning the \texttt{method.mice} argument. Note that two other joint modelling methods are also available: \texttt{JM-GL} from the R package \texttt{mix} and \texttt{JM-DP} from the R package \texttt{DPImputeCont https://github.com/hang-j-kim/DPImputeCont}

\section*{Value}
a list of 3 objects

\begin{itemize}
  \item \texttt{res.imp} a list with the several imputed datasets
  \item \texttt{res.conv} for FCS methods, an array given the within inertia of each imputed variable at each iteration and for each imputed dataset
  \item \texttt{call} the matching call
\end{itemize}

\section*{References}


\section*{See Also}
\texttt{mice} \texttt{choosenbclust} \texttt{choosemaxit} \texttt{varselbest} \texttt{imp.mix}

\section*{Examples}
\begin{verbatim}
data(wine) set.seed(123456) wine.na <- wine wine.na$cult <- NULL wine.na <- as.matrix(wine.na) wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA
\end{verbatim}
nb.clust <- 3 # number of clusters
m <- 3 # number of imputed data sets
res.imp <- imputedata(data.na = wine.na, nb.clust = nb.clust, m = m)
lapply(res.imp$res.imp, summary)

---

overimpute

Overimputation diagnostic plot

Description

overimpute assesses the fit of the predictive distribution after performing multiple imputation with the imputedata function

Usage

overimpute(
  res.imputedata,
  plotvars = NULL,
  plotinds = NULL,
  nnodes = 2,
  path.outfile = NULL,
  alpha = 0.1,
  mfrow = NULL,
  mar = c(5, 4, 4, 2) - 1.9
)

Arguments

res.imputedata an output from the imputedata function
plotvars column index of the variables overimputed
plotinds row index of the individuals overimputed
nnodes an integer indicating the number of nodes for parallel calculation. Default value is 5
path.outfile a vector of strings indicating the path for redirection of print messages. Default value is NULL, meaning that silent imputation is performed. Otherwise, print messages are saved in the files path.outfile/output.txt. One file per node is generated.
alpha alpha level for prediction intervals
mfrow a vector of the form c(nr, nc)
mar a numerical vector of the form c(bottom, left, top, right)
Details

This function imputes each observed value from each conditional imputation model obtained from the imputedata function. The comparison between the “overimputed” values and the observed values is made by building a confidence interval for each observed value using the quantiles of the overimputed values (see Blackwell et al. (2015) <doi:10.1177/0049124115585360>). Note that confidence intervals built with quantiles require a large number of imputations. If the model fits well the data, then the 90% confidence interval should contain the observed value in 90% of the cases.

The function overimpute takes as an input an output of the imputedata function (res.imputedata argument), the indices of the incomplete continuous variables that are plotted (plotvars), the indices of individuals (can be useful for time consuming imputation methods), the number of CPU cores for parallel computation, and the path for exporting print message generated during the parallel process (path.outfile).

Value

A list of two matrices

res.plot 7-columns matrix that contains (1) the variable which is overimputed, (2) the observed value of the observation, (3) the mean of the overimputations, (4) the lower bound of the confidence interval of the overimputations, (5) the upper bound of the confidence interval of the overimputations, (6) the proportion of the other variables that were missing for that observation in the original data, and (7) the color for graphical representation

res.values a matrix with overimputed values for each cell. The number of columns corresponds to the number of values generated (i.e. the number of imputed datasets)

References


Examples

data(wine)
require(parallel)
set.seed(123456)
ref <- wine$cult
nb.clust <- 3
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

nnodes <- 2 # Number of CPU cores used for parallel computation

# Multiple imputation (with m = 100, but use m>200 in practice)
res.imp.over <- imputedata(data.na = wine.na,
                          nb.clust = nb.clust,
# Overimputation

```r
res.over <- overimpute(res.imp.over,
    nnodes = nnodes,
    plotvars = 1,
    plotinds = sample(seq(nrow(wine.na)), size = 30))
```

### Description

`varselbest` performs variable selection from an incomplete dataset (see Bar-Hen and Audigier (2022) <doi:10.1080/00949655.2022.2070621>) in order to specify the imputation models to use for FCS imputation methods.

### Usage

```r
varselbest(
    data.na = NULL,
    res.imputedata = NULL,
    listvar = NULL,
    nb.clust = NULL,
    nnodes = 1,
    sizeblock = 5,
    method.select = "knockoff",
    B = 200,
    r = 0.3,
    graph = TRUE,
    printflag = TRUE,
    path.outfile = NULL,
    mar = c(2, 4, 2, 0.5) + 0.1,
    cex.names = 0.7,
    modelNames = NULL
)
```

### Arguments

- `data.na`: a dataframe with only numeric variables
- `res.imputedata`: an output from `imputedata`
- `listvar`: a character vector indicating for which subset of incomplete variables variable selection must be performed. By default all column names.
- `nb.clust`: the number of clusters used for imputation
- `nnodes`: number of CPU cores for parallel computing. By default, `nnodes = 1`
varselbest

- **sizeblock**: an integer indicating the number of variables sampled at each iteration
- **method.select**: a single string indicating the variable selection method applied on each subset of variables
- **B**: number of iterations, by default B = 200
- **r**: a numerical vector (or a single real number) indicating the threshold used for each variable in listvar. Each value of r should be between 0 and 1. See details.
- **graph**: a boolean. If TRUE two graphics are plotted per variable in listvar: a graphic reporting the variable importance measure of each explanatory variable and a graphic reporting the influence of the number iterations (B) on the importance measures
- **printflag**: a boolean. If TRUE, a message is printed at each iteration. Use printflag = FALSE for silent selection.
- **path.outfile**: a vector of strings indicating the path for redirection of print messages. Default value is NULL, meaning that silent imputation is performed. Otherwise, print messages are saved in the files path.outfile/output.txt. One file per node is generated.
- **mar**: a numerical vector of the form c(bottom, left, top, right). Only used if graph = TRUE
- **cex.names**: expansion factor for axis names (bar labels) (only used if graph = TRUE)
- **modelNames**: a vector of character strings indicating the models to be fitted in the EM phase of clustering

**Details**

varselbest performs variable selection on random subsets of variables and, then, combines them to recover which explanatory variables are related to the response. More precisely, the outline of the algorithm are as follows: let consider a random subset of sizeblock among p variables. By choosing sizeblock small, this subset is low dimensional, allowing treatment of missing values by standard imputation method for clustered individuals. Then, any selection variable scheme can be applied (lasso, stepwise and knockoff are proposed by tuning the method.select argument). By resampling B times, a sample of size sizeblock among the p variables, we may count how many times, a variable is considered as significantly related to the response and how many times it is not. We need to define a threshold (r) to conclude if a given variable is significantly related to the response.

**Value**

- **predictormatrix**: a numeric matrix containing 0 and 1 specifying on each line the set of predictors to be used for each target column of the incomplete dataset.
- **res.varsel**: a list given details on the variable selection procedure (only required for checking convergence by the chooseB function)
- **proportion**: a numeric matrix of proportion indicating on each line the variable importance of each predictor
- **call**: the matching call
References

See Also
mice, clusterMI, imputedata, knockoff, glmnet, imp.mix

Examples

data(wine)

require(parallel)
set.seed(123456)
ref <- wine$cult
nb.clust <- 3
wine.na <- wine
wine.na$cult <- NULL
wine.na <- as.matrix(wine.na)
wine.na[sample(seq(length(wine.na)), size = ceiling(length(wine.na)/3))] <- NA

nnodes <- 2 # parallel::detectCores()
B <- 150 # Number of iterations
m <- 5 # Number of imputed data sets

# variable selection
res.varsel <- varselbest(data.na = wine.na,
                        nb.clust = nb.clust,
                        listvar = c("alco","malic"),
                        B = B,
                        nnodes = nnodes)
predictmat <- res.varsel$predictormatrix

# imputation
res.imp.select <- imputedata(data.na = wine.na, method = "FCS-homo",
                           nb.clust = nb.clust, predictmat = predictmat, m = m)

wine
Chemical analysis of wines from three different cultivars

Description
Data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.
wine

Usage
data(wine)

Format
A data frame with 178 rows and 14 variables:

cult  Cultivar
alco  Alcohol
malic  Malic acid
ash  Ash
alca  Alcalinity of ash
mg  Magnesium
phe  Total phenols
fla  Flavanoids
nfla  Nonflavanoid phenols
pro  Proanthocyanins
col  Color intensity
hue  Hue
ratio  OD280/OD315 of diluted wines
prol  Proline

Source
https://archive.ics.uci.edu/ml/datasets/wine

References
Index

* datasets
  wine, 24

agnes, 14, 15

chooseB, 3
chooseM, 5
chooseMaxit, 7, 19
chooseNbcust, 8, 19
chooser, 9
clusterMI, 6, 12, 24
clusterMI-package, 2
cmeans, 15

cellnmf, 15

glmnet, 24

imp.mix, 19, 24
imputedata, 6, 9, 15, 18, 20–22, 24

kmeans, 17
knockoff, 24

Mclust, 14, 15
mice, 19, 24
MiniBatchKmeans, 17

nselectboot, 14, 15

overimpute, 20

varselbest, 4, 19, 22

wine, 24