Package ‘doMIsaul’

October 18, 2021

Title  Do Multiple Imputation-Based Semi-Supervised and Unsupervised Learning

Version 1.0.1

Description  Algorithms for (i) unsupervised learning for dataset with missing data and/or left-censored data, using multiple imputation and consensus clustering; (ii) semi-supervised learning with a survival endpoint (right-censored) for complete or incomplete datasets, using multiple imputation and consensus clustering in the latter case. The methods are described in Faucheux et al. (2021) <doi:10.1002/bimj.201900366> and Faucheux et al. (2021) <doi:10.1002/bimj.202000365>, respectively.

License  GPL (>= 3)

URL  https://github.com/LilithF/doMIsaul

BugReports  https://github.com/LilithF/doMIsaul/issues

Imports  aricode, arules, clusterCrit, dplyr, ggplot2, Gmedian, graphics, MASS, methods, mice, NbClust, ncvreg, plyr, scales, stats, survival, utils, withr

Suggests  censReg, cluster, clustMixType, CPE, dbscan, e1071, ggpubr, Hmisc, igraph, mclust, parallel, RColorBrewer, reshape2, testthat (>= 3.0.0), timeROC, truncnorm

Config/testthat/edition  3

Encoding  UTF-8

Language  en-US

RoxygenNote  7.1.1

NeedsCompilation  no

Author  Lilith Faucheux [aut, cre], Sylvie Chevret [ths], Matthieu Resche-Rigon [ctb], Marie Perrot-Dockès [ctb], Eric Han [ctb]

Maintainer  Lilith Faucheux <lilith.faucheux@inserm.fr>
**Description**

Remove small clusters (i.e. unclassified observations for which no consensus was obtained)

**Usage**

```r
cleanUp_partition(
  partition,
  min.cluster.size = 10,
  level.order = NULL,
  Unclassified = c(NA, "Unclassified")
)
```

**Arguments**

- `partition` the partition to clean (vector).
- `min.cluster.size` Minimum cluster size (i.e., smaller clusters will be discarded)
level.order optional. If you supply a variable the cluster levels will be ordinated according to the mean values for the variable
Unclassified string for the label of the unclassified observations. defaults value is NA.

Value

The cleaned up partition (factor).

Examples

```r
part <- factor(kmeans(iris[, 1:4], 8)$cluster)
summary(part)
part.clean <- cleanUp_partition(part, Unclassified = "Unclassified")
```

---

CVE_LP  
*Cross-validation for cox regression using the linear predictor estimator with wrapper for warnings handling*

Description

Cross-validation for cox regression using the linear predictor estimator with wrapper for warnings handling

Usage

```r
CVE_LP(x)
```

Arguments

- `x` list of 3 named elements: `data` (data containing columns time and status), `partition` (dataframe with 1 column), `nfolds` (number of fold for cross-validation).

Value

numeric, cross-validation error

Examples

```r
data(cancer, package = "survival")
cancer$status <- cancer$status - 1
part <- data.frame(Cl = factor(cancer[, "sex"], stringsAsFactors = TRUE))
CVE_LP(list(data = cancer, partition = part, nfolds = 10))
```
evaluate_partition_semisup

Evaluation of a semisupervised obtained partition in comparison to reference partitions

Description

Evaluate number of clusters, ARI, AUC difference, c-index and CPE, with Supervised, unsupervised and Semisupervised reference partitions

Usage

evaluate_partition_semisup(
    partition,
    ref.unsup,
    ref.sup,
    ref.semisup,
    data.surv,
    TMIN = 2,
    TMAX = 5
)

Arguments

partition Vector containing cluster ids of the partition to evaluate.
ref.unsup Vector: Unsupervised reference partition (i.e. data structure).
ref.sup Vector: Supervised reference partition (i.e. using survival parameters).
ref.semisup Vector: Semisupervised reference partition (i.e. combining both).
data.surv dataframe with variables time and status.
TMIN time point to start analyzing AUC.
TMAX time point to analyze AUC.

Value

a list of named performances values

Examples

library(survival) # survival should be loaded in the environment
data(cancer, package = "survival")
cancer$status <- cancer$status - 1
if (requireNamespace("CPE", quietly = TRUE)) {
  res <- evaluate_partition_semisup (
    partition = factor(rep(c(1,2,3), each = 50)),
    ref.unsup = factor(rep(c(1,2,3), times = c(100, 25, 25))),
    ref.sup = factor(rep(c(1,2), times = c(50, 100))),
    ref.semisup = factor(rep(c(1,2), times = c(50, 100))),
    data.surv = cancer, time = cancer$time, status = cancer$status
  )
}
evaluate_partition_unsup

Comparison of an unsupervised obtained partition to a reference partition.

Description

Compares partitions on number of cluster, ARI and percentage of unclassified observations.

Usage

evaluate_partition_unsup(
  partition,
  partition.ref,
  is.missing = NULL,
  is.cens = NULL
)

Arguments

partition vector (factor): the partition to evaluate.
partition.ref reference partition 1 (ex partition on complete data or true partition if known).
is.missing boolean vector identifying observations with missing data (coded TRUE), from those without (coded FALSE).
is.cens the incomplete dataframe with NA for missing and left-censored data (or the complete datasets if all data were observed).

Value

A list containing the following elements: Nbclust: number of clusters of the partition; ARI: ARI value on cases classified by both partitions; ARI.cc: ARI value on cases complete AND classified by both partitions; ARI.nona: ARI on cases with no missing data AND classified by both partitions; ARI.nocens: ARI on cases with no censored data AND classified by both partitions; Per.Unclass: Percentage of observations unclassified in the partition; Per.Unclass.cc: Among complete cases, percentage of observations unclassified in the partition; Per.Unclass.na: Among cases with missing data, percentage of observations unclassified in the partition; Per.Unclass.cens: Among cases with censored data, percentage of observations unclassified in the partition; Per.Unclass.ic: Among incomplete cases, percentage of observations unclassified in the partition.
Examples

```r
res <- evaluate_partition_unsup(
  partition = factor(rep(c(1,2,3), each = 50)),
  partition.ref = factor(rep(c(1,2,3), times = c(100, 25, 25))))

## With missing data
res2 <- evaluate_partition_unsup(
  partition = factor(rep(c(1,2,3), each = 50)),
  partition.ref = factor(rep(c(1,2,3), times = c(100, 25, 25))),
  is.missing = sample(c(TRUE, FALSE), 150, replace = TRUE, prob = c(.2,.8)))

## With missing and censored data
missing.indicator <- sample(c(TRUE, FALSE), 150, replace = TRUE, prob = c(.2,.8))
Censor.indicator <- data.frame(
  X1 = runif(150, 1, 5),
  X2 = runif(150, 6, 8),
  X3 = runif(150, 3, 9))
Censor.indicator$X1[missing.indicator] <- NA
Censor.indicator$X1[sample(c(TRUE, FALSE), 150,replace = TRUE, prob = c(.1,.9))] <- NA
Censor.indicator$X2[sample(c(TRUE, FALSE), 150,replace = TRUE, prob = c(.3,.7))] <- NA
Censor.indicator$X3[sample(c(TRUE, FALSE), 150,replace = TRUE, prob = c(.05,.95))] <- NA
res3 <- evaluate_partition_unsup(
  partition = factor(rep(c(1,2,3), each = 50)),
  partition.ref = factor(rep(c(1,2,3), times = c(100, 25, 25))),
  is.missing = missing.indicator,
  is.cens = Censor.indicator)

## With missing and censored data and unclassified observations
res4 <- evaluate_partition_unsup(
  partition = factor(rep(c(1,2, NA,3), times = c(50, 40, 20, 40))),
  partition.ref = factor(rep(c(1,2,3), times = c(100, 25, 25))),
  is.missing = missing.indicator,
  is.cens = Censor.indicator)
```

### initiate_centers

**Initiate centers for clustering algorithm**

**Description**

Initiate centers for clustering algorithm

**Usage**

```r
initiate_centers(data, N = 1000, t = 1, k, algorithms = NULL, seeds.N = NULL)
```
**mice.impute.cens**

**Arguments**

- `data` Dataset that clustering will be applied on
- `N` Integer. Number clustering initialization (set of centers) to generate
- `t` Numeric between 0 and 1. weight coefficient between only random centers (t=1) and only centers from clustering (t=0).
- `k` Vector of size N containing the number of centers for each initialization.
- `algorithms` list of algorithm(s) (size N * (1-t)) to generate centers if t!=1, given as characters. Possible values are "km" for 'K-means', "kmed" for 'K-medians', "hclust.mean", "hclust.med" for hierarchical clustering with mean or median position of the center.
- `seeds.N` (optional) vector of size N containing seeds for each initialization.

**Value**

list of size N containing coordinates of centers for clustering initialization.

**Examples**

```r
Cent.init <- initiate_centers(data = iris[, 1:4], N = 10,
    k = sample(c(2:7), 10, replace = TRUE))
```

**Description**

Function from Lapidus et al. for imputing left-censored data with mice

**Usage**

```r
mice.impute.cens(
    y,
    ry,
    x,
    lod.j,
    lod.name = "lod",
    REDRAW = FALSE,
    wy = NULL,
    ...
)
```
Arguments

- **y**: Vector to be imputed
- **ry**: Logical vector of length `length(y)` indicating the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (TRUE) and missing values (FALSE) in `y`.
- **x**: Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
- **lod.j**: Censoring value.
- **lod.name**: Suffix name used for the censored variable.
- **REDRAW**: Boolean indicating whether values should be redrawn if some are over the censoring limit.
- **wy**: Logical vector of length `length(y)`. A TRUE value indicates locations in `y` for which imputations are created.
- ... Other named arguments.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`.

MImpute  
*Wrapper functions for multivariate imputation with survival data or left-censored data*

Description

Performs imputation of the missing data using MICE and returns a list in the correct format for the `unsupMI()` and `seMIsupcox()` functions. `MImpute()` performs imputation for datasets with missing data only. `MImpute_surv()` performs imputation for a dataset with survival data. The Nelson Aalen estimator is calculated and used as predictor in the imputation. Time is not used as predictor. `MImpute_lcens()` performs imputation for a dataset with left-censored data. Note that with `MImpute_lcens()` pmm imputation is performed for variables not affected by left-censoring.

Usage

```r
MImpute(
  data,
  mi.m,
  method = NULL,
  predMat = NULL,
  maxit = 10,
  return.midsObject = FALSE
)
```

```r
MImpute_surv(
  data = NULL,
  method = NULL,
  predMat = NULL,
  maxit = 10,
  return.midsObject = FALSE
)```
MImpute

data,
mi.m,
time.status.names = c("time", "status"),
return.midsObject = FALSE
)

MImpute_lcens(
data,
data.lod,
standards,
mi.m,
mice.log = 10,
maxit = 10,
return.midsObject = FALSE
)

MImpute_lcenssurv(
data,
mi.m,
time.status.names = c("time", "status"),
data.lod,
standards,
mice.log = 10,
var.log = NULL,
maxit = 10,
return.midsObject = FALSE
)

Arguments

data Dataframe with incomplete data. (for MImpute_lcens(), with NA for both missing and left-censored data).

mi.m Number of imputations to perform.

method Optional. single string, or a vector of strings specifying the imputation method to be used for each column in data (passed to mice()). If NULL default mice setting are used.

predMat Optional. supply a predictorMatrix (passed to mice()). If NULL default mice setting are used.

maxit passed to mice().

return.midsObject Boolean

time.status.names Names of the variables for time and status (in that order).

data.lod Dataframe containing indicators of which observation are left-censored (censoring value for such observations and any other values for not censored observations). The colnames should correspond to variables in data. The variables that are left-censored are thus given in data (with left-censored data as NA) and in
data.lod with random values for observed data and the LOD for left-censored data. Note that if the data are to be logged (is.numeric(mice.log)), only the argument data will be logged, therefore, the LOD values given here should be given as log(LOD) with the correct number of decimals: round(log(LOD),mice.log).

standards Dataframe of 1 row containing the LOD values (not logged, whatever the value for mice.log).

mice.log set to FALSE if the imputation should be performed on unlogged data. Otherwise, number of decimal to save after taking the log of data (should be 10 unless for specific reasons): in that case the data will be unlogged after imputation.

var.log names of variables to log if mice.log is numeric. If NULL, all variables but those in time.status.names will be logged.

Value

If return.midsObject == FALSE a list of size mi.m, containing the imputed datasets. If return.midsObject == TRUE a list of 2, the first element (imputed.data) being the list of size mi.m as described in the previous sentence, the 2nd element (mids.obj) containing the mids object as returned by mice().

Examples

data(cancer, package = "survival")
cancer.imp <- MImpute(cancer[, -c(1:3)], 2)

## MImpute_surv
data(cancer, package = "survival")
cancer$status <- cancer$status - 1
cancer.imp <- MImpute_surv(cancer, 1)

## MImpute_lcens
toy <- iris[, 1:4]
# censor on variables 3 and 4, with LOD at quantile .1 and .2.
LODs <- toy[1, ]
LODs[1, ] <- c(NA, NA, quantile(toy[,3], .2), quantile(toy[,4], .1))
# Censor indicator
Censored <- data.frame(Petal.Length = runif(150, 50,60),
                        Petal.Width = runif(150, 50,60))
Censored[toy[,3] < LODs[,3], 1] <- LODs[,3]
Censored[toy[,4] < LODs[,4], 2] <- LODs[,4]
# NA for censored data
toy[toy[,3] < LODs[,3], 3] <- NA
toy[toy[,4] < LODs[,4], 4] <- NA
# Additional missing data
toy[sample(1:nrow(toy), 30), 1] <- NA
toy[sample(1:nrow(toy), 30), 3] <- NA
toy[sample(1:nrow(toy), 30), 4] <- NA

toy.imp <- MImpute_lcens(data = toy, data.lod = Censored, standards = LODs,
                          mi.m = 1, mice.log = FALSE)

## MImpute_lcenssurv
data(cancer, package = "survival")
cancer$status <- cancer$status - 1
toy2 <- cancer[, -1]
# censor on variables age and meal.cal, with LOD at quantile .1 and .2.
LODs <- toy2[1,]
LODs[1,] <- c(NA, NA, quantile(toy2[, "age"], .2, na.rm = TRUE), NA, NA, NA, NA, quantile(toy2[, "meal.cal"], .1, na.rm = TRUE), NA)
# Censor indicator
Censored <- data.frame(age = runif(nrow(toy2), 300, 400),
                        meal.cal = runif(nrow(toy2), 50, 60))
Censored[toy2[, "age"] < LODs[1, "age"], "age"] <- LODs[1, "age"]
Censored[!is.na(toy2[, "meal.cal"]) &
          toy2[, "meal.cal"] < LODs[1, "meal.cal"], "meal.cal"] <-
LODs[1, "meal.cal"]
# NA for censored data
toy2[toy2[, "age"] < LODs[1, "age"], "age"] <- NA
toy2[!is.na(toy2[, "meal.cal"])] &
     toy2[, "meal.cal"] < LODs[1, "meal.cal"], "meal.cal"] <- NA
# Additional missing data
toy2[sample(1:nrow(toy2), 30), 6] <- NA
toy2[sample(1:nrow(toy2), 30), 3] <- NA
toy2[sample(1:nrow(toy2), 30), 4] <- NA
toy2$sex <- factor(toy2$sex)
toy2$ph.ecog <- factor(toy2$ph.ecog)

toy2.imp <- MImpute_lcenssurv(
    data = toy2, mi.m = 1, data.lod = Censored, standards = LODs,
    mice.log = FALSE)

**MultiCons Consensus Clustering Algorithm**

**Description**

Performs MultiCons clustering, from Al-Najdi et Al. For some reason, if you want to use mclust() clustering, the package needs to be loaded manually.

**Usage**

```R
MultiCons(
  DB,
  Clust_entry = FALSE,
  Clustering_selection = c("kmeans", "pam", "OPTICS", "agghc", "AGNES", "DIANA",
                            "MCLUST", "CMeans", "FANNY", "BaggedClust"),
  num_algo = 10,
  maxClust = 10,
  sim.indice = "Jaccard",
  returnAll = FALSE,
  Plot = TRUE,
  verbose = FALSE
)
```
Arguments

DB
Either data or dataframe of partitions.

Clust_entry
Is DB partitions (TRUE) or data (FALSE).

Clustering_selection
If DB is data, clustering algorithm to select among. Must be included in default value.

num_algo
Number of clustering algorithms to perform.

maxClust
Maximum number of clusters.

sim.indice
Index for defining best partition. Passed to clusterCrit::extCriteria(), see clusterCrit::getCriteriaNames(FALSE) for other available indexes. If more than one index are given, only the first one will be used.

returnAll
Should all partitions (TRUE) or only the best (FALSE) be returned.

Plot
Should tree be plotted.

verbose
Passed on to mclust() and other functions.

Value

A list of 2: performances and partitions. If returnAll is TRUE, both elements of the list contain results for all levels of the tree, else they only contain the results for the best level of the tree.

Examples

library(mclust)
### With clustering algorithm choices
MultiCons(iris[, 1:4],
  Clustering_selection = c("kmeans", "pam", "DIANA", "MCLUST"),
  Plot = TRUE)
### With a manual clustering entry
parts <- data.frame(factor(rep(c(1,2,3), each = 50)),
  factor(rep(c(1,2,3), times = c(100, 25, 25))),
  factor(rep(c(1,2), times = c(50, 100))),
  factor(rep(c(3, 2, 1), times = c(120, 10, 20))),
  stringsAsFactors = TRUE)
MultiCons(parts, Clust_entry = TRUE, Plot = TRUE)

partition_generation
Unsupervised partition with K selection

Description

Generates a partition using clust.algo algorithm, with k.crit for selecting the number of clusters

Usage

partition_generation(data, LOG, clust.algo, k.crit)
Arguments

- **data** dataframe to cluster
- **LOG** logical. Should all columns of the dataset be logged before applying clustering algorithms?
- **clust.algo** vector of strings: name of clustering algorithms to use (use "km" for k-means, "kmed" for K-medians, "hc" for hclust() and/or "mclust" for mclust()).
- **k.crit** string. Criterion to select the optimal number of clusters (for each imputed dataset). Use "ch" for Calinski and Harabasz criterion (not available for "mclust"), "CritCF" for CritCF or bic for BIC ("mclust" only).

Value

da dataframe with one column for each algorithm in clust.algo, containing the cluster IDs.

Examples

```
partition_generation(iris[, 1:4], LOG = FALSE,
                   clust.algo = c("km", "hc"), k.crit = "ch")
```

Description

ggplot type boxplots for each vars.cont by partition level.

Usage

```
plot_boxplot(
data,                           # The dataset.
partition.name,                # string. Name of the partition (in data). The partition variable should be a factor.
vars.cont,                      # vector of strings. variables to plot (continuous only).
vars.cont.names = NULL,         # unclass.name = "Unclassified",
include.unclass = FALSE,       # add.n = FALSE,
nc.facet = 10                   # )
)
```

vars.cont.names
Optional. Names for displaying the continuous variables. (given in the same
order than vars.cont)

unclass.name
If applicable, name for the unclassified observations in the partition.

include.unclass
Boolean, should boxplot be displayed for the unclassified or should they be ex-
cluded from the plot.

add.n
Boolean. Should the number of samples per cluster be indicated on the x axis
and color legend.

nc.facet
integer. Number of columns in the facet_wrap().

Value

ggplot object.

Examples

data(cancer, package = "survival")
cancer$status <- factor(cancer$status)
plot_boxplot(data = cancer, partition.name = "status",
vars.cont = c("age", "meal.cal", "wt.loss"))

## With unclassifieds
cancer$status.2 <- as.character(cancer$status)
cancer$status.2[sample(1:nrow(cancer), 30)] <- "Unclassif."
cancer$status.2 <- factor(cancer$status.2)
plot_boxplot(data = cancer, partition.name = "status.2",
vars.cont = c("age", "meal.cal", "wt.loss"),
unclass.name = "Unclassif.", include.unclass = TRUE)

## With unclassifieds (as NA)
cancer$status.3 <- cancer$status
cancer$status.3[sample(1:nrow(cancer), 30)] <- NA
plot_boxplot(data = cancer, partition.name = "status.3",
vars.cont = c("age", "meal.cal", "wt.loss"),
unclass.name = NA, include.unclass = TRUE, add.n = TRUE)

plot_frequency

ggplot type barplots representing frequencies for each vars.cat by
partition level.

Description

ggplot type barplots representing frequencies for each vars.cat by partition level.
Usage

```
plot_frequency(
    data,
    partition.name,
    vars.cat,
    vars.cat.names = NULL,
    binary.simplify = TRUE,
    unclass.name = "Unclassified",
    include.unclass = FALSE
)
```

Arguments

- **data**: The dataset.
- **partition.name**: string. Name of the partition (in data). The partition variable should be a factor.
- **vars.cat**: vector of strings. Variables to plot (categorical only).
- **vars.cat.names**: Optional. Names for displaying the categorical variables. (given in the same order than `vars.cat`)
- **binary.simplify**: boolean. Should only the 1st level be kept for binary variables in `vars.cat`?
- **unclass.name**: If applicable, name for the unclassified observations in the partition.
- **include.unclass**: boolean, should boxplot be displayed for the unclassified or should they be excluded from the plot.

Value

ggplot object.

Examples

```
data(cancer, package = "survival")
cancer$status <- factor(cancer$status)
plot_frequency(data = cancer, partition.name = "status",
               vars.cat = c("sex", "ph.ecog"))

## With unclassifieds

cancer$status.2 <- as.character(cancer$status)
cancer$status.2[sample(1:nrow(cancer), 30)] <- "Unclassif."
cancer$status.2 <- factor(cancer$status.2)
plot_frequency(data = cancer, partition.name = "status.2",
               vars.cat = c("sex", "ph.ecog"),
               unclass.name = "Unclassif.", include.unclass = TRUE)

## With unclassifieds (as NA)
cancer$status.3 <- cancer$status
cancer$status.3[sample(1:nrow(cancer), 30)] <- NA
plot_frequency(data = cancer, partition.name = "status.3",
               vars.cat = c("sex", "ph.ecog"),
               unclass.name = "Unclassif.", include.unclass = TRUE)
```
plot_MIpca plots only mean value while plot_MIpca_all() plots all values for the selected observations.

Description

plot_MIpca() plots only mean value while plot_MIpca_all() plots all values for the selected observations.

Usage

plot_MIpca(
  data.list,
  obs.sel,
  color.var = NULL,
  pca.varsel = NULL,
  pc.sel = c(1, 2)
)

plot_MIpca_all(
  data.list,
  obs.sel,
  pca.varsel = NULL,
  color.var = NULL,
  pc.sel = c(1, 2),
  alpha = 0.4
)

Arguments

data.list The list of the imputed datasets.
obs.sel The selection of observations to highlight. If NULL, no observations are selected; if numeric, the vector corresponds to the observations’ row number to highlight, if character, the string should be of type a condition (TRUE/FALSE) on the dataset to select the observations, where the dataset is referred to as "DATA" (ex: obs.sel = "DATA$X1>3").
color.var Either NULL to color according to obs.sel, "none" to use no color, or a vector of size nrow(data.list[[1]]) (a factor).
pca.varsel optional. A vector of strings containing the names of the variables to use for the PCA. If NULL all variables in the dataset will be used.
pc.sel  Numeric vector of size 2 containing the indexes of the principal components to plot. Default is PC1 and PC2.

alpha  Transparency level for plotting the point of the selected observations.

Value  
A ggplot object.

Examples  
```r
data(cancer, package = "survival")
cancer.imp <- MImpute(cancer[, -c(1:3)], 4)
plot_MIpca(cancer.imp, 1:10,
          pca.varsel = c("age", "sex", "ph.ecog", "meal.cal", "wt.loss"))
plot_MIpca(cancer.imp, obs.sel = NULL, color.var = factor(cancer$status),
          pca.varsel = c("age", "sex", "ph.ecog", "meal.cal", "wt.loss"))
data(cancer, package = "survival")
cancer.imp <- MImpute(cancer[, -c(1:3)], 6)
plot_MIpca_all(cancer.imp, 1:10,
               pca.varsel = c("age", "sex", "ph.ecog", "meal.cal", "wt.loss"))
```

seMIsupcox  
Semisupervised learning for a right censored endpoint

Description  
MultiCons consensus based method for MI-Semisupervised clustering. The final partition is a consensus of the Pareto-optimal solutions.

Usage  
```r
seMIsupcox(
  Impute = FALSE,
  Impute.m = 5,
  center.init = TRUE,
  center.init.N = 500,
  center.init.Ks = 2:7,
  X,
  CVE.fun = "LP",
  Y,
  nfolds = 10,
  save.path = NULL,
  Unsup.Sup.relImp = list(relImp.55 = c(0.5, 0.5)),
  plot.cons = FALSE,
  cleanup.partition = TRUE,
  min.cluster.size = 10,
  level.order = NULL,
)```
Args

**Arguments**

- **Impute**: Boolean. Default is FALSE to indicate that the user performed the imputation and provides the imputed data. If TRUE, the imputation will be performed within the call using the MImpute_surv() function. Note that if Impute is TRUE, center.init is also forced to TRUE as the center coordinates may depend on the imputation.

- **Impute.m**: Used only if Impute is TRUE; number of imputations to perform.

- **center.init**: Either a User supplied List of dataframe containing the cluster centers coordinates (for example as obtained with initiate_centers()), Or TRUE to initiate the centers within the call of the function (performed with initiate_centers()). Note that if TRUE a random initialization will be performed. For a finer tuning of the center initialization the user should generate and provide the list of centers coordinates.

- **center.init.N**: Used only if center.init is TRUE. The number to initialization to produce. Default to 500.

- **center.init.Ks**: Used only if center.init is TRUE. Vector of number of clusters to generate for the initialization. Default to 2 to 7 clusters.

- **X**: Data, in the form of a list of data.frame(s). The list should be one length 1 if data are complete or if Impute is TRUE, of should be a list of imputed dataframes if data are incomplete. If columns named "time" and "status" are present they will be discarded for the clustering.

- **CVE.fun**: string indicating how to calculate the cross validation error: only LP is available and stands for linear predictor approach (using the 'ncvreg' package).

- **Y**: Passed to CVE.fun, Outcome data: should be dataframe or matrix with 2 columns: "time" and "status".

- **nfolds**: Number of folds for cross-validation.

- **save.path**: Path indicating where objectives values for each iteration should be saved. If null the values are not saved.

- **Unsup.Sup.relImp**: List of weights for the unsupervised and supervised objectives for the Pareto optimal solution. Default is to use only one set of weights: same weight.

- **plot.cons**: Logical. Should the consensus tree be plotted?

- **cleanup.partition**: should the partition be trimmed of small clusters. (The consensus may generate small clusters of observations for which there is no consensus on the cluster assignment)

- **min.cluster.size**: if cleanup.partition == TRUE: Minimum cluster size (i.e., smaller clusters will be discarded)
level.order  if cleanup.partition == TRUE: optional. If you supply a variable the cluster levels will be ordinated according to the mean values for the variable

Unclassified if cleanup.partition == TRUE string for the label of the unclassified observations. defaults value is NA.

return.detail logical. Should the detail of imputation specific partition be returned, in supplement to the final consensus partition?

Value

A vector containing the final cluster IDs. Or if return.detail == TRUE, a list containing Consensus: the final cluster ID. Detail: the clusters obtained for each imputed dataset, Imputed.data a list containing the imputed datasets.

Examples

data(cancer, package = "survival")
cancer$status <- cancer$status - 1
cancer <- cancer[, -1]
### With imputation included
res <- seMIsupcox(X = list(cancer), Y = cancer[, c("time", "status")],
                 Impute = TRUE, Impute.m = 3, center.init = TRUE,
                 nfolds = 10, center.init.N = 20)

### With imputation and center initialization not included
## 1 imputation
cancer.imp <- MImpute_surv(cancer, 3)

## 2 Center initialization
# A low N value is used for example purposes. Higher values should be used.
N <- 20
center.number <- sample(2:6, size = N, replace = TRUE)
the.seeds <- runif(N) * 10^9
sel.col <- which(!colnames(cancer) %in% c("time", "status"))
init <- sapply(1:length(cancer.imp), function(mi.i) {
    initiate_centers(data = cancer.imp[[mi.i]][, sel.col],
        N = N, t = 1, k = center.number,
        seeds.N = the.seeds),
    USE.NAMES = TRUE, simplify = FALSE)

## 3 learning
res1 <- seMIsupcox(X = cancer.imp, Y = cancer[, c("time", "status")],
        Impute = FALSE, center.init = init, nfolds = 10,
        cleanup.partition = FALSE)
res2 <- seMIsupcox(X = cancer.imp, Y = cancer[, c("time", "status")],
        center.init = init, nfolds = 10)
Display table with comparison of the partition with categorical variables.

**Usage**

```r
table_categorical(
  data,
  partition.name,
  vars.cat,
  vars.cat.names = NULL,
  na.value = "",
  nb.dec = 1,
  text.pval = FALSE
)
```

**Arguments**

- `data` The dataset.
- `partition.name` string. Name of the partition (in data). The partition variable should be a factor.
- `vars.cat` vector of strings. variables to compare to (categorical only).
- `vars.cat.names` Optional. Names for displaying the categorical variables. (in the same order than `vars.cat`)
- `na.value` Value to use for the empty cases (e.g. "" or NA).
- `nb.dec` digit. Number of decimals for the percentage.
- `text.pval` boolean. Set to TRUE to display "p=", to FALSE to display only the value.

**Value**

table with n and percentage values per level of the partition and chi square test p-values.

**Examples**

```r
data(cancer, package = "survival")
cancer$status <- factor(cancer$status)
table_categorical(data = cancer, partition.name = "status",
  vars.cat = c("sex", "ph.ecog"))
```
Display table with comparison of the partition with continuous variables.

Usage

```r
table_continuous(
  data,
  partition.name,
  vars.cont,
  vars.cont.names = NULL,
  na.value = "",
  nb.dec = 1,
  text.pval = FALSE
)
```

Arguments

- **data**: The dataset.
- **partition.name**: string. Name of the partition (in data). The partition variable should be a factor.
- **vars.cont**: vector of strings. Variables to compare to (continuous only).
- **vars.cont.names**: Optional. Names for displaying the continuous variables. (in the same order than vars.cont)
- **na.value**: value to use for the empty cases (e.g. "" or NA).
- **nb.dec**: digit. Number of decimals for the mean and quartile values.
- **text.pval**: boolean. Set to TRUE to display "p=", to FALSE to display only the value.

Value

Table with mean and Q1 Q3 values per level of the partition and ANOVA test p-values.

Examples

```r
data(cancer, package = "survival")
cancer$status <- factor(cancer$status)
table_continuous(data = cancer, partition.name = "status",
                 vars.cont = c("age", "meal.cal", "wt.loss"))
```
Description

Unsupervised clustering for multiply imputed datasets using MultiCons() consensus (Faucheux et al. 2021 procedure)

Usage

```
unsupMI(
  Impute = FALSE,
  Impute.m = 5,
  cens.data.lod = NULL,
  cens.standards = NULL,
  cens.mice.log = 10,
  censsurv.var.log = NULL,
  censsurv.maxit = 10,
  data,
  log.data = FALSE,
  algo = "km",
  k.crit = "ch",
  comb.cons = FALSE,
  plot.cons = FALSE,
  return.detail = FALSE,
  not.to.use = c("time", "status"),
  cleanup.partition = TRUE,
  min.cluster.size = 10,
  level.order = NULL,
  Unclassified = "Unclassified"
)
```

Arguments

- **Impute**
  - Default is FALSE to indicate that the user performed the imputation and provides the imputed data. Otherwise string ("MImpute", "MImpute_surv" or "MImpute_lcens") to perform the imputation within the call using the MImpute(), MImpute_surv() or MImpute_lcens() function.

- **Impute.m**
  - Used only if Impute is not FALSE; number of imputations to perform

- **cens.data.lod**
  - Passed to MImpute_lcens() if Impute == MImpute_lcens

- **cens.standards**
  - Passed to MImpute_lcens() if Impute == MImpute_lcens

- **cens.mice.log**
  - Passed to MImpute_lcens() if Impute == MImpute_lcens

- **censsurv.var.log**
  - For MImpute_lcenssurv imputation: names of variables to log if mice.log is numeric. If NULL, all variables but those intime.status.names will be logged.
unsupMI

censsurv.maxit for MImpute_lccensurv imputation: passed to mice().
data Data, in the form of a list of data.frame(s). The list should be one length 1 if data are complete or if Impute is not FALSE, it should be a list of imputed dataframes if data are incomplete and imputed. If some columns are in not.to.use, they will be discarded for the clustering.

log.data logical. Should all columns of the dataset be logged before applying clustering algorithms?

algo vector of strings: name of clustering algorithms to use (use "km" for k-means, "kmed" for K-medians, "hc" for hclust() and/or "mclust" for mclust()).

k.crit string. Criterion to select the optimal number of clusters (for each imputed dataset). Use "ch" for Calinski and Harabasz criterion (not available for mclust), "CritCF" for CritCF or "bic" for BIC (mclust only).

comb.cons logical. Forced to FALSE if length(algo)<2. Use TRUE to perform an additional consensus from all partitions generates, whatever the algorithm.

plot.cons logical. Use TRUE to print the MultiCons tree. Note that if all partitions are identical across the imputation no consensus will be performed and therefore not plot will be obtained even if plot.cons = TRUE.

return.detail logical. Should the detail of imputation specific partition and the imputed data be returned, in the supplement to the final consensus partition?

not.to.use vector of strings: names of the columns that should be discarded for the learning step.

cleanup.partition should the partition be trimmed of small clusters. (The consensus may generate small clusters of observations for which there is no consensus on the cluster assignment)

min.cluster.size if cleanup.partition == TRUE: Minimum cluster size (i.e., smaller clusters will be discarded)

level.order if cleanup.partition == TRUE: optional. If you supply a variable the cluster levels will be ordinated according to the mean values for the variable

Unclassified if cleanup.partition == TRUE string for the label of the unclassified observations. defaults value is NA.

Value

if length(algo)>1 a vector of final cluster ID; if length(algo)>1 a data.frame with each column being the final cluster ID for the corresponding algorithm. Or if return.detail == TRUE, a list containing Consensus: the final cluster ID (or data.frame), Detail: the clusters obtained for each imputed dataset, Imputed.data a list containing the imputed datasets.

Examples

### With imputation included
data(cancer, package = "survival")
cancer$status <- cancer$status - 1
res.0 <- unsupMI(data = list(cancer), Impute = "MImpute_surv",

```r
```
### With imputation not included
#### 1 imputation
```r
cancer.imp <- MImpute_surv(cancer, 3)
```
#### 2 learning
```r
res <- unsupMI(data = cancer.imp, cleanup.partition = FALSE)
summary(factor(res))
res.1 <- unsupMI(data = cancer.imp)
summary(factor(res.1))
```
#### 2.bis learning with several algorithms
```r
res.2 <- unsupMI(data = cancer.imp, algo = c("km", "hc"), comb.cons = TRUE, plot.cons = TRUE)
```
Index

cleanUp_partition, 2
CVE_LP, 3

evaluate_partition_semisup, 4
evaluate_partition_unsup, 5

initiate_centers, 6

mice.impute.cens, 7
MImpute, 8
MImpute_lccens (MImpute), 8
MImpute_lccenssurv (MImpute), 8
MImpute_surv (MImpute), 8
MultiCons, 11

partition_generation, 12
plot_boxplot, 13
plot_frequency, 14
plot_MIpcac, 16
plot_MIpcac_all (plot_MIpcac), 16

seMIsupcox, 17

table_categorical, 20
table_continuous, 21

unsupMI, 22