Package ‘SamplingStrata’

February 18, 2018

Type Package

Title Optimal Stratification of Sampling Frames for Multipurpose Sampling Surveys

Version 1.2

Date 2018-2-4

Author Giulio Barcaroli, Marco Ballin, Daniela Pagliuca, Egon Willighagen, Diego Zardetto

Maintainer Giulio Barcaroli <gbarcaroli@gmail.com>

Description In the field of stratified sampling design, this package offers an approach for the determination of the best stratification of a sampling frame, the one that ensures the minimum sample cost under the condition to satisfy precision constraints in a multivariate and multidomain case. This approach is based on the use of the genetic algorithm: each solution (i.e. a particular partition in strata of the sampling frame) is considered as an individual in a population; the fitness of all individuals is evaluated applying the Bethel-Chromy algorithm to calculate the sampling size satisfying precision constraints on the target estimates. Functions in the package allows to: (a) analyse the obtained results of the optimisation step; (b) assign the new strata labels to the sampling frame; (c) select a sample from the new frame accordingly to the best allocation. Functions for the execution of the genetic algorithm are a modified version of the functions in the 'genalg' package.

License GPL (>= 2)

LazyLoad yes

Depends R (>= 2.15.0), memoise

NeedsCompilation no

Repository CRAN

Date/Publication 2018-02-18 22:14:36 UTC
**R topics documented:**

adjustSize .................................................. 2
bethel .................................................. 3
buildFrameDF .............................................. 5
buildStrataDF .............................................. 6
checkInput ................................................ 7
effects .................................................. 8
optimizeStrata ............................................. 10
plotSample ............................................... 14
selectSample .............................................. 15
strata .................................................... 16
swisserrors .............................................. 17
swissframe ............................................... 18
swissmunicipalities ...................................... 19
swissstrata .............................................. 20
tuneParameters ........................................... 21
updateFrame .............................................. 24
updateStrata ............................................. 25
var.bin .................................................. 27

**Index** 29

---

adjustSize  

*Adjustment of the sample size in case it is externally given*

**Description**

The optimisation step finds the best stratification that minimises the sample size under given precision constraints. In some cases, the goal is not the minimisation of the sample size, as this value is given externally. Nonetheless, it is still possible to perform the optimisation of the stratification, and after that proceed to an adjustment of the sample size by increasing or decreasing it proportionally in each resulting stratum.

**Usage**

`adjustSize(size, strata, cens=NULL)`

**Arguments**

- `size`  
The value of the sample size given externally
- `strata`  
The new (aggregated) strata generated by the function `optimizeStrata`
- `cens`  
The strata to be censused
Value

The strata generated by the function `optimizeStrata`, where the variable 'SOLUZ' has been adjusted by taking into account the total required sample size.

Author(s)

Giulio Barcaroli

Examples

```r
## not run
library(SamplingStrata)
data(swisserrors)
data(swissstrata)
solution <- optimizeStrata (
  errors = swisserrors,
  strata = swissstrata,
  cens = NULL,
  strcens = FALSE,
  alldomains = TRUE,
  dom = dominio,
  initialStrata = nrow(swissstrata)/3,
  addStrataFactor = 0.01,
  minnumstr = 2,
  iter = 50,
  pops = 20,
  mut_chance = 0.05,
  elitism_rate = 0.2,
  highvalue = 100000000,
  suggestions = NULL,
  realAllocation = TRUE,
  writefiles = FALSE)
# sum(solution$aggr_strata$SOLUZ)
# Adjustment of total sample size (decreasing)
adjustedStrata <- adjustSize(size=300,strata=solution$aggr_strata,cens=FALSE)
sum(adjustedStrata$SOLUZ)
# Adjustment of total sample size (increasing)
adjustedStrata <- adjustSize(size=500,strata=solution$aggr_strata,cens=FALSE)
sum(adjustedStrata$SOLUZ)
## End(not run)
```

Description

Multivariate optimal allocation for different domains of interest in stratified sample design
Usage

\texttt{bethel} (paras:\texttt{stratif, errors, minnumstrat=2, maxiter=200, maxiter1=25, printa=FALSE, realAllocation=FALSE, epsilon=1e-11})

Arguments

- \texttt{errors} : Data frame of coefficients of variation for each domain
- \texttt{stratif} : Data frame of survey strata
- \texttt{minnumstrat} : Minimum number of units per strata (default=2)
- \texttt{maxiter} : Maximum number of iterations of the algorithm (default=200)
- \texttt{maxiter1} : Maximum number of iterations (default=25) of the general procedure. This kind of iteration may be required by the fact that when in a stratum the number of allocated units is greater or equal to its population, that stratum is set as “census stratum”, and the whole procedure is re-initialised
- \texttt{printa} : If \texttt{TRUE} then two attributes are added to the resulting vector. The first (‘confr’) is a comparison between results obtained with 3 different allocation methods: Bethel, proportional and equal. The second (‘outcv’) is a table reporting planned and actual CV, together with a sensitivity analysis
- \texttt{realAllocation} : If \texttt{FALSE}, the allocation is based on INTEGER values; if \texttt{TRUE}, the allocation is based on REAL values
- \texttt{epsilon} : Epsilon (default=1e-11): this value is used to compare the difference in results from one iteration to the other; if it is lower than "epsilon", then the procedure stops

Value

A vector containing the computed optimal allocation

Author(s)

Daniela Pagliuca with contributions from Teresa Buglielli and Giulio Barcaroli

Examples

\texttt{library(SamplingStrata)}
\texttt{data(strata)}
\texttt{data(errors)}
\texttt{n <- bethel(strata, errors, printa=TRUE)}
buildFrameDF

Builds the "frame" dataframe from a dataset containing information on all the units in the population of reference

Description

This function allows to build the information regarding the sampling frame of the population of reference. Mandatory variables are: - the name of the dataset containing the sampling frame of the population of reference - an identifier (Id) - a set of auxiliary variables X - a set of target variables Y - the indicator of the domain to which the unit belongs

Usage

buildFrameDF(df, id, X, Y, domainvalue)

Arguments

df  
This is the name of the dataframe containing the information on all the units in population of reference.

id  
This is the name of the identifier in the sampling frame.

X  
A character vector containing the names of the auxiliary variables in the frame dataset

Y  
A character vector containing the names of the target variables in the frame dataset

domainvalue  
The name of the variable in the frame dataset that contains the indication of the domains to which the units belong.

Value

A dataframe

Author(s)

Giulio Barcaroli

Examples

data(swissmunicipalities)
id = "Nom"
X = c("Surfacesbois","Surfacescult")
Y = c("Pop820","Pop2040")
domainvalue = "REG"
frame <- buildFrameDF(swissmunicipalities,id,X,Y,domainvalue)
head(frame)
buildStrataDF

Builds the "strata" dataframe containing information on target variables Y's distributions in the different strata, starting from sample data or from a frame

Description

This function allows to build the information regarding strata in the population required as an input by the algorithm of Bethel for the optimal allocation. In order to estimate means and standard deviations for target variables Y's, we need data coming from: (1) a previous round of the survey whose sample we want to plan; (2) sample data from a survey with variables that are proxy to the ones we are interested to; (3) a frame containing values of Y's variables (or proxy variables) for all the population. In all cases, each unit in the dataset must contain auxiliary information (X's variables) and also target variables Y's (or proxy variables) values: under these conditions it is possible to build the dataframe "strata", containing information on the distribution of Y's in the different strata (namely, means and standard deviations), together with information on strata (total population, if it is to be censused or not, the cost per single interview). If the information is contained in a sample dataset, a variable named WEIGHT is expected to be present. In case of a frame, no such variable is given, and the function will define a WEIGHT variable for each unit, whose value is always 1. Missing values for each Y variable will not be taken into account in the computation of means and standard deviations (in any case, NA's can be present in the dataset). The dataframe "strata" is written to an external file (tab delimited, extension "txt"), and will be used as an input by optimizeStrata.

Usage

buildStrataDF(dataset)

Arguments

dataset This is the name of the dataframe containing the sampling data, or frame data. It is strictly required that auxiliary information is organised in variables named as X1, X2, ... , Xm (there should be at least one of them) and the target variables are denoted by Y1, Y2, ... , Yn. In addition, in case of sample data, a variable named 'WEIGHT' must be present in the dataframe, containing the weights associated to each sampling unit

Value

A dataframe containing strata

Author(s)

Giulio Barcaroli
checkInput

Examples

data(swissframe)
strata <- buildStrataDF(swissframe)
head(strata)

cHECKS THE INPUTS TO THE PACKAGE: DATAFRAMES "ERRORS", "STRATA" AND "SAMPLING FRAME"

Description

This function checks the internal structure of the different input dataframes ("errors", "strata" and "sampling frame"), and also the correctness of the relationships among them.

Usage

checkInput(errors=NULL, strata=NULL, sampframe=NULL)

Arguments

errors Dataframe containing the precision levels expressed in terms of maximum acceptable coefficients of variation that estimates of target variables Y’s of the survey must comply.
strata Dataframe containing the information related to strata.
sampframe Dataframe containing the information related to all the units belonging to the population of interest.

Author(s)

Giulio Barcaroli

Examples

library(SamplingStrata)
data(swisserrors)
data(swissstrata)
data(swissframe)
checkInput(swisserrors, swissstrata, swissframe)
checkInput(strata=swissstrata, sampframe=swissframe)
checkInput(strata=swissstrata)
Description

Dataframe containing precision levels (expressed in terms of acceptable CV’s)

Usage

data(errors)

Format

The constraint data frame (errors) contains a row per each domain value with the following variables:

- **DOM** Type of domain code (factor)
- **CV1** Planned coefficient of variation for first variable Y1 (numeric)
- **CVj** Planned coefficient of variation for j-th variable Yj (numeric)
- **CVn** Planned coefficient of variation for last variable Yn (numeric)
- **domainvalue** Value of the domain to which the constraints refer (numeric)

Details

Note: the names of the variables must be the ones indicated above

Examples

```r
## data(errors)
## errors
```

`evalSolution` | Allows to evaluate the solution produced by the function `optimizeStrata` by selecting a number of samples from the frame with the optimal stratification, and calculating average CV’s on the target variables Y’s together with differences between estimates and the values of the parameters in the population.
evalSolution

Description
The user can indicate the number of samples that must be selected by the frame to which the optimal stratification has been applied. The allocation is the one reported in the dataframe ‘outstrata’. First, the true values of the parameters are calculated from the frame. Then, for each sample the sampling estimates are calculated, together with the differences between them and the true values of the parameters. At the end, an estimate of the CV is produced for each target variable, in order to compare them with the precision constraints set at the beginning of the optimization process. If the flag ‘writeFiles’ is set to TRUE, boxplots of distribution of the CV’s in the different domains are produced for each Y variable (‘cv.pdf’), together with boxplot of the distributions of differences between estimates and values of the parameters in the population (‘differences.pdf’).

Usage

evalSolution(frame, outstrata, nsampl=100, cens=NULL, writeFiles=FALSE)

Arguments

frame The frame to which the optimal stratification has been applied (‘framenew’)  
outstrata The new (aggregated) strata generated by the function ‘optimizeStrata’  
nsampl The number of sample to be drawn from the frame  
cens A dataframe containing units to be censused  
writeFiles A flag to write in the work directory the outputs of the function

Value
A list containing (i) the CV distribution in the domains and (ii) the bias distribution in the domains

Author(s)
Giulio Barcaroli

Examples

## Not run:
library(SamplingStrata)
data(swisserrors)
data(swissstrata)
solution <- optimizeStrata (errors = swisserrors, strata = swissstrata, cens = NULL, strcens = FALSE, alldomains = TRUE, dom = dominio, initialStrata = nrow(swissstrata)/3,
addStrataFactor = 0.01, 
minnumstr = 2, 
iter = 50, 
pops = 20, 
mut_chance = 0.05, 
elitism_rate = 0.2, 
highvalue = 100000000, 
suggestions = NULL, 
realAllocation = TRUE, 
writeFiles = FALSE)
# update sampling strata with new strata labels
swissstrata <- swissstrata[swissstrata$DOOM == dominio,]
newstrata <- updateStrata(swissstrata, solution, writeFiles = TRUE)
# update sampling frame with new strata labels
data(swissframe)
framenew <- updateFrame(frame=swissframe,newstrata=newstrata,writeFile=TRUE)
samp <- selectSample(framenew,solution$aggr_strata,writeFiles=TRUE)
# evaluate the current solution
results <- evalSolution(framenew, solution$aggr_strata, 10, cens=NULL, writeFiles = TRUE)

diff <- read.csv("differences.csv")
numY <- sum(grepl("diff", colnames(diff)))
k <- ceiling(numY/4)
close.screen(all.screens = TRUE)
for (j in 1:k) {
  split.screen(c(2, 2))
  m = 0
  for (i in (1 + 4 * (j - 1)):((4 * j)) {
    if (i <= numY) {
      m = m+1
      stmt <- paste("screen("m", ")", sep = "")
      eval(parse(text = stmt))
      stmt <- paste("boxplot(diff,i,"-dom,
        data=diff,ylab='Differences',
        xlab='Domain',col = 'orange')",sep="")
      eval(parse(text = stmt))
      stmt <- paste("mtext(expression(y"i","), side=3, adj=0, cex=1.0, line=1)",
        sep = "")
      eval(parse(text = stmt))
    }
  }
  close.screen(all.screens = TRUE)
}
## End(Not run)
**optimizeStrata**

**Description**

This function runs a set of other functions to optimise the stratification of a sampling frame.

**Usage**

```r
optimizeStrata(
  errors,  # This is the (mandatory) dataframe containing the precision levels expressed in terms of Coefficients of Variation that estimates on target variables Y’s of the survey must comply
  strata,  # This is the (mandatory) dataframe containing the information related to "atomic" strata, i.e. the strata obtained by the Cartesian product of all auxiliary variables X’s. Information concerns the identifiability of strata (values of X’s) and variability of Y’s (for each Y, mean and standard error in strata)
  cens = NULL,  # This the (optional) dataframe containing the takeall strata, those strata whose units must be selected in whatever sample. It has same structure than "strata" dataframe
  strcens = FALSE,  # Flag (TRUE/FALSE) to indicate if takeall strata do exist or not. Default is FALSE
  alldomains = TRUE,  # Flag (TRUE/FALSE) to indicate if the optimization must be carried out on all domains (default is TRUE). If it is set to FALSE, then a value must be given to parameter 'dom'
  dom = NULL,  # Indicates the domain on which the optimization must be carried. It is an integer value that has to be internal to the interval (1 <-> number of domains). If 'alldomains' is set to TRUE, it is ignored
  initialstrata = nrow(strata),  # Given initial strata
  addStrataFactor = 0.01,  # Factor used to increase the number of strata
  minnumstr = 10,  # Minimum number of strata
  iter = 20,  # Number of iterations
  pops = 20,  # Number of populations
  mut_chance = 0.05,  # Mutation rate
  elitism_rate = 0.2,  # Elitism rate
  highvalue = 1e+08,  # High value
  suggestions = NULL,  # Suggestions
  realAllocation = TRUE,  # Indicates if real allocation must be done
  writeFiles = FALSE,  # Write files or not
  showPlot = TRUE)  # Show plot or not
```

**Arguments**

- **errors**: This is the (mandatory) dataframe containing the precision levels expressed in terms of Coefficients of Variation that estimates on target variables Y’s of the survey must comply.
- **strata**: This is the (mandatory) dataframe containing the information related to "atomic" strata, i.e. the strata obtained by the Cartesian product of all auxiliary variables X’s. Information concerns the identifiability of strata (values of X’s) and variability of Y’s (for each Y, mean and standard error in strata).
- **cens**: This the (optional) dataframe containing the takeall strata, those strata whose units must be selected in whatever sample. It has same structure than "strata" dataframe.
- **strcens**: Flag (TRUE/FALSE) to indicate if takeall strata do exist or not. Default is FALSE.
- **alldomains**: Flag (TRUE/FALSE) to indicate if the optimization must be carried out on all domains (default is TRUE). If it is set to FALSE, then a value must be given to parameter 'dom'.
- **dom**: Indicates the domain on which the optimization must be carried. It is an integer value that has to be internal to the interval (1 <-> number of domains). If 'alldomains' is set to TRUE, it is ignored.
initialStrata  This is the initial limit on the number of strata for each solution. Default is the number of atomic strata. In cases of more than a domain, it has to be given as a vector, where each element indicates the number of initial strata in each domain. If the parameter ‘addStrataFactor’ is equal to zero, then ‘initialStrata’ is equivalent to the maximum number of strata to be obtained in the final solution.

addStrataFactor  This parameter indicates the probability that at each mutation the number of strata may increase with respect to the current value. Default is 0.01 (1

minnumstr  Indicates the minimum number of units that must be allocated in each stratum. Default is 2

iter  Indicated the maximum number of iterations (= generations) of the genetic algorithm. Default is 20

pops  The dimension of each generations in terms of individuals. Default is 50

mut_chance  Mutation chance: for each new individual, the probability to change each single chromosome, i.e. one bit of the solution vector. High values of this parameter allow a deeper exploration of the solution space, but a slower convergence, while low values permit a faster convergence, but the final solution can be distant from the optimal one. Default is 0.05

elitism_rate  This parameter indicates the rate of better solutions that must be preserved from one generation to another. Default is 0.2 (20

highvalue  Parameter for genetic algorithm. Not to be changed

suggestions  Optional parameter for genetic algorithm that indicates one possible solution (maybe from previous runs) that will be introduced in the initial population. Default is NULL.

realAllocation  If FALSE, the allocation is based on INTEGER values; if TRUE, the allocation is based on REAL values. Default is TRUE.

writeFiles  Indicates if the various dataframes and plots produced during the execution have to be written in the working directory. Default is FALSE.

showPlot  Indicates if the plot showing the trend in the value of the objective function has to be shown or not. Default is TRUE.

Value

A list containing (1) the vector of the solution and (2) the optimal aggregated strata

Author(s)

Giulio Barcaroli

Examples

#  This is a toy example, and can be run
#

library(SamplingStrata)
# optimisation of sampling strata

```r
solution <- optimizeStrata(
  errors = errors, strata = strata,
  cens = NULL, strcens = FALSE,
  initialStrata = nrow(strata),
  addStrataFactor = 0.01,
  minnumstr = 2,
  iter = 30,
  pops = 20,
  mut_chance = 0.05,
  elitism_rate = 0.2,
  highvalue = 100000000,
  suggestions = NULL,
  realAllocation = TRUE,
  writefiles = FALSE,
  showplot = TRUE)
sum(ceiling(solution$aggr_strata$SOLUZ))
head(solution$aggr_strata)
```

# The following example is realistic, but is time consuming

```r
library(SamplingStrata)
data(swisserrors)
data(swissstrata)

# optimisation of sampling strata

```r
solution <- optimizeStrata(
  errors = swisserrors, strata = swissstrata,
  cens = NULL, strcens = FALSE,
  initialStrata = as.numeric(table(frame$domainvalue)),
  addStrataFactor = 0.01,
  minnumstr = 2,
  iter = 60,
  pops = 20,
  mut_chance = 0.05,
  elitism_rate = 0.2,
  highvalue = 100000000,
  suggestions = NULL,
  realAllocation = TRUE,
  writefiles = FALSE,
  showplot = TRUE)
sum(ceiling(solution$aggr_strata$SOLUZ))
head(solution$aggr_strata)
```

## End(Not run)
plotSamprate

Plotting sampling rates in the different strata for each domain in the solution.

Description

Once the optimization step has been carried out, by applying this function it is possible to obtain the visualization of the proportion of sampling units in the different strata for each domain in the obtained solution.

Usage

plotSamprate(solution, dom)

Arguments

- solution: Solution obtained by executing optimizeStrata
- dom: Identification of the domain

Value

Plot

Examples

```r
## not run
library(samplingstrata)
data(swisserrors)
data(swissstrata)
# optimisation of sampling strata
solution <- optimizeStrata(
  errors = swisserrors,
  strata = swissstrata,
  cens = NULL,
  strcens = FALSE,
  initialStrata = 3000,
  addStrataFactor = 0.01,
  minnumstr = 2,
  iter = 60,
  pops = 20,
  mut_chance = 0.05,
  elitism_rate = 0.2,
  highvalue = 100000000,
  suggestions = NULL,
  writefiles = FALSE)
# plot of the sampling rates in strata of domain 1
plotSamprate(solution, 1)

## end(not run)
```
selectSample

Selection of a stratified sample from the frame with srswor method

Description

Once optimal stratification has been obtained (in the dataframe 'outstrata'), and a new frame has been built by assigning to the units of the old one the new stratum labels (by means of "updateFrame" function), it is possible to select a stratified sample from the frame with the srswor method. The result of the execution of "selectSample" function is a dataframe containing selected units, with the probabilities of inclusion. It is possible to output this dataframe in a .csv file. One more .csv file is produced ("sampling check"), containing coherency checks between (a) population in frame strata (b) population in optimised strata (c) planned units to be selected in optimised strata (d) actually selected units (e) sum of weights in each stratum.

Usage

selectSample(frameL outstrataL writeFiles = FALSE, verbatim=TRUE)

Arguments

frame This is the (mandatory) dataframe containing the sampling frame, as it has been modified by the execution of the "updateFrame" function. Name of stratum variable must be 'strato'.

outstrata This is the (mandatory) dataframe containing the information related to resulting stratification obtained by the execution of "optimizeStrata" function. Name of stratum variable must be 'strato'.

writeFiles Indicates if at the end of the processing the resulting strata will be outputted in a delimited file. Default is "FALSE".

verbatim Indicates if information on the drawn sample must be printed or not. Default is "TRUE".

Value

A dataframe containing the sample

Author(s)

Giulio Barcaroli with contribution from Diego Zardetto

Examples

```
# The following example is realistic, but is time consuming
#
## Not run:
library(SamplingStrata)
data(swisserrors)
```
data(swissstrata)
# optimisation of sampling strata
solution <- optimizeStrata(
  errors = swisserrors,
  strata = swissstrata,
  cens = NULL,
  strcens = FALSE,
  initialStrata = 3000,
  addStrataFactor = 0.01,
  minnumstr = 2,
  iter = 60,
  pops = 20,
  mut_chance = 0.05,
  elitism_rate = 0.2,
  highvalue = 100000000,
  suggestions = NULL,
  writefiles = FALSE)
# updating sampling strata with new strata labels
newstrata <- updateStrata(swissstrata, solution)
# updating sampling frame with new strata labels
data(swissframe)
framew <- updateFrame(frame=swissframe,newstrata=newstrata)
# selection of sample
sample <- selectSample(frame=framew, outstrata=solution$aggr_strata)
head(sample)

## End(Not run)

---

### strata

Dataframe containing information on strata in the frame

#### Description

Dataframe containing information on strata in the frame

#### Usage

data(strata)

#### Format

The strata data frame (strata) contains a row per stratum with the following variables:

- **stratum** Identifier of the stratum (numeric)
- **N** Number of population units in the stratum (numeric)
- **X1** Value of first auxiliary variable X1 in the stratum (factor)
- **Xi** Value of i-th auxiliary variable Xi in the stratum (factor)
- **Xk** Value of last auxiliary variable Xk in the stratum (factor)
**M1**  Mean in the stratum of the first variable \( Y_1 \) (numeric)

**Mj**  Mean in the stratum of the \( j \)-th variable \( Y_t \) (numeric)

**Mn**  Mean in the stratum of the last variable \( Y \) (numeric)

**S1**  Standard deviation in the stratum of the first variable \( Y \) (numeric)

**Sj**  Standard deviation in the stratum of the \( j \)-th variable \( Y_t \) (numeric)

**Sn**  Standard deviation in the stratum of the last variable \( Y \) (numeric)

**cens**  Flag (1 indicates a take all stratum, 0 a sampling stratum) (numeric) Default = 0

**cost**  Cost per interview in each stratum. Default = 1 (numeric)

**DOM1**  Value of domain to which the stratum belongs (factor or numeric)

### Details

Note: the names of the variables must be the ones indicated above

### Examples

```
# data(strata)
# head(strata)
```

<table>
<thead>
<tr>
<th>swisserrors</th>
<th>Precision constraints (maximum CVs) as input for Bethel allocation</th>
</tr>
</thead>
</table>

### Description

Dataframe containing precision levels (expressed in terms of acceptable CV’s)

### Usage

```
data(errors)
```

### Format

The constraint data frame (swisserrors) contains a row per each domain value with the following variables:

**DOM**  Type of domain code (factor)

**CV1**  Planned coefficient of variation for first variable \( Y_1 \) (number of men and women aged between 0 and 19) (numeric)

**CV2**  Planned coefficient of variation for second variable \( Y_2 \) (number of men and women aged between 20 and 39) (numeric)

**CV3**  Planned coefficient of variation for third variable \( Y_3 \) (number of men and women aged between 40 and 64) (numeric)

**CV4**  Planned coefficient of variation for forth variable \( Y_4 \) (number of men and women aged between 65 and over) (numeric)

**domainvalue**  Value of the domain to which the constraints refer (numeric)
### swissframe

**Dataframe containing information on all units in the population of reference that can be considered as the final sampling unit (this example is related to Swiss municipalities)**

**Description**

Dataframe containing information on all municipalities in Swiss (it is a derivation of dataframe "swissmunicipalities" in "sampling" package)

**Usage**

```r
data(swissframe)
```

**Format**

The "swissframe" dataframe contains a row per each Swiss municipality with the following variables:

- **progr** Progressive associated to the frame unit (numeric)
- **id** Name of the frame unit (character)
- **X1** Classes of total population in the municipality (factor with 18 values)
- **X2** Classes of wood area in the municipality (factor with 3 values)
- **X3** Classes of area under cultivation in the municipality (factor with 3 values)
- **X4** Classes of mountain pasture area in the municipality (factor with 3 values)
- **X5** Classes of area with buildings in the municipality (factor with 3 values)
- **X6** Classes of industrial area in the municipality (factor with 3 values)
- **Y1** Number of men and women aged between 0 and 19 (numeric)
- **Y2** Number of men and women aged between 20 and 39 (numeric)
- **Y3** Number of men and women aged between 40 and 64 (numeric)
- **Y4** Number of men and women aged between 65 and over (numeric)
- **domainvalue** Value of domain to which the municipality belongs (factor or numeric)

**Examples**

```r
# data(swisserrors)
# swisserrors
```

```r
# data(swissframe)
# head(strata)
```
Description

This population provides information about the Swiss municipalities in 2003.

Usage

data(swissmunicipalities)

Format

A data frame with 2896 observations on the following 22 variables:

- **CT**  Swiss canton.
- **REG**  Swiss region.
- **COM**  municipality number.
- **Nom**  municipality name.
- **HApoly**  municipality area.
- **Surfacesbois**  wood area.
- **Surfacescult**  area under cultivation.
- **Alp**  mountain pasture area.
- **Airbat**  area with buildings.
- **Airind**  industrial area.
- **P00BMTOT**  number of men.
- **P00BWTOT**  number of women.
- **Pop020**  number of men and women aged between 0 and 19.
- **Pop2040**  number of men and women aged between 20 and 39.
- **Pop4065**  number of men and women aged between 40 and 64.
- **Pop65P**  number of men and women aged between 65 and over.
- **H00PTOT**  number of households.
- **H00P01**  number of households with 1 person.
- **H00P02**  number of households with 2 persons.
- **H00P03**  number of households with 3 persons.
- **H00P04**  number of households with 4 persons.
- **POPTOT**  total population.

Source

Swiss Federal Statistical Office.
Examples

```r
# data(swissmunicipalities)
# hist(swissmunicipalities$POPTOT)
```

---

**swissstrata**

*Dataframe containing information on strata in the swiss municipalities frame*

---

**Description**

Dataframe containing information on strata in the swiss municipalities frame

**Usage**

```r
data(swissframe)
```

**Format**

The "swissstrata" dataframe contains a row per stratum with the following variables:

- **STRATO**  Identifier of the stratum (character)
- **N**  Number of population units in the stratum (numeric)
- **X1**  Classes of total population in the municipality (factor with 18 values)
- **X2**  Classes of wood area in the municipality (factor with 3 values)
- **X3**  Classes of area under cultivation in the municipality (factor with 3 values)
- **X4**  Classes of mountain pasture area in the municipality (factor with 3 values)
- **X5**  Classes of area with buildings in the municipality (factor with 3 values)
- **X6**  Classes of industrial area in the municipality (factor with 3 values)
- **M1**  Mean in the stratum of Y1 (number of men and women aged between 0 and 19) (numeric)
- **M2**  Mean in the stratum of Y2 (number of men and women aged between 20 and 39) (numeric)
- **M3**  Mean in the stratum of Y3 (number of men and women aged between 40 and 64) (numeric)
- **M4**  Mean in the stratum of Y4 (number of men and women aged between 64 and over) (numeric)
- **S1**  Standard deviation in the stratum of Y1 (number of men and women aged between 0 and 19) (numeric)
- **S2**  Standard deviation in the stratum of Y2 (number of men and women aged between 20 and 39) (numeric)
- **S3**  Standard deviation in the stratum of Y3 (number of men and women aged between 40 and 64) (numeric)
- **S4**  Standard deviation in the stratum of Y4 (number of men and women aged between 64 and over) (numeric)
- **cens**  Flag (1 indicates a take all stratum, 0 a sampling stratum) (numeric) Default = 0
- **cost**  Cost per interview in each stratum. Default = 1 (numeric)
- **DOM1**  Value of domain to which the stratum belongs Default = 1 (factor or numeric)
**tuneParameters**

**Examples**

```r
# data(swissstrata)
# head(swissstrata)
```

---

**tuneParameters**  
*Execution and compared evaluation of optimization runs*

---

**Description**

This function allows to execute a number of optimization runs, varying in a controlled way the values of the parameters, in order to find their most suitable values, by comparing the resulting solutions. It can be applied only to a given domain per time. Most parameters of this function are the same than those of the function 'optimizeStrata', but they are given in a vectorial format. The length of each vector is given by the number of optimizations to be run: it is therefore possible to define different combination of values of the parameters for each execution of 'optimizeStrata'. After each optimization run, from the corresponding optimized frame a given number of samples are drawn. For each of them, the estimates of the target variables Y's are computed ("precision"), together with the associated coefficients of variations, and the absolute differences between the values of the estimates and the true values in the population ("bias"). Information on the distribution of bias (differences) and precision (CV's) are outputted, and also boxplots for each of them are produced, in order to permit a compared evaluation of the different solutions found in the different runs. As the optimal solution is stored for each run, after the evaluation it is possible to use it directly, or as a "suggestion" for a new optimization with more iterations (in order to improve it).

**Usage**

```r
tuneParameters (  
  noptim,  
  nsampl,  
  frame,  
  errors = errors,  
  strata = strata,  
  cens = NULL,  
  strcens = FALSE,  
  alldomains = FALSE,  
  dom = 1,  
  initialstrata,  
  addstratafactor,  
  minnumstr,  
  iter,  
  pops,  
  mut_chance,  
  elitism_rate  
)
```
**Arguments**

- **noptim**: Number of optimization runs to be performed
- **nsampl**: Number of samples to be drawn from the optimized population frame after each optimization
- **frame**: The (mandatory) dataframe containing the sampling frame
- **errors**: This is the (mandatory) dataframe containing the precision levels expressed in terms of Coefficients of Variation that estimates on target variables Y’s of the survey must comply
- **strata**: This is the (mandatory) dataframe containing the information related to "atomic" strata, i.e. the strata obtained by the Cartesian product of all auxiliary variables X’s. Information concerns the identifiability of strata (values of X’s) and variability of Y’s (for each Y, mean and standard error in strata)
- **cens**: This the (optional) dataframe containing the takeall strata, those strata whose units must be selected in whatever sample. It has same structure than "strata" dataframe
- **strcens**: Flag (TRUE/FALSE) to indicate if takeall strata do exist or not. Default is FALSE
- **alldomains**: Flag (TRUE/FALSE) to indicate if the optimization must be carried out on all domains. It must be left to its default (FALSE)
- **dom**: Indicates the domain on which the optimization runs must be performed. It is an integer value that has to be internal to the interval (1 <-> number of domains). It is mandatory, if not indicated, the default (1) is taken.
- **initialstrata**: This is the initial limit on the number of strata for each solution. Default is 3000. This parameter has to be given in a vectorial format, whose length is given by the number of different optimisations ( = value of parameter 'noptim')
- **addStrataFactor**: This parameter indicates the probability that at each mutation the number of strata may increase with respect to the current value. Default is 0.01 (1 This parameter has to be given in a vectorial format, whose length is given by the number of different optimisations ( = value of parameter 'noptim')
- **minnumstr**: Indicates the minimum number of units that must be allocated in each stratum. Default is 2. This parameter has to be given in a vectorial format, whose length is given by the number of different optimisations ( = value of parameter 'noptim')
- **iter**: Indicated the maximum number of iterations (= generations) of the genetic algorithm. Default is 20. This parameter has to be given in a vectorial format, whose length is given by the number of different optimisations ( = value of parameter 'noptim')
- **pops**: The dimension of each generations in terms of individuals. Default is 50. This parameter has to be given in a vectorial format, whose length is given by the number of different optimisations ( = value of parameter 'noptim')
- **mut_chance**: Mutation chance: for each new individual, the probability to change each single chromosome, i.e. one bit of the solution vector. High values of this parameter allow a deeper exploration of the solution space, but a slower convergence, while low values permit a faster convergence, but the final solution can be distant from
tuneParameters

the optimal one. Default is 0.05. This parameter has to be given in a vectorial format, whose length is given by the number of different optimisations (= value of parameter 'noptim')

elitism_rate

This parameter indicates the rate of better solutions that must be preserved from one generation to another. Default is 0.2 (0.2 This parameter has to be given in a vectorial format, whose length is given by the number of different optimisations (= value of parameter 'noptim')

Author(s)

Giulio Barcaroli

Examples

```
# Not run:
#-----------------------------------------------
# data setting
library(SamplingStrata)
data(swissstrata)
data(swisserrors)
data(swissframe)
# As this function can be applied only to a given domain per time,
# we select the first domain
frame <- swissframe[, swissframe$domainvalue == 1]
strata <- swissstrata[, swissstrata$DOM == 1]
errors <- swisserrors[, swisserrors$domainvalue == 1]
#-----------------------------------------------
# parameters setting
noptim <- 8 # Number of runs
nsamp1 <- 100 # Number of samples to be drawn after each optimization
initialstrata <- ceiling(c(1:noptim)*0.1*(nrow(strata))) # Number of initial strata
addstratafactor <- rep(0.01, noptim) # Rate for increasing initial strata
minnumstr <- rep(0.5, noptim) # Minimum number of units per stratum
iter <- rep(200, noptim) # Number of iterations for each optimization
pops <- rep(20, noptim) # Number of solutions for each iteration
mut_chance <- rep(0.004, noptim) # Mutation chance
elitism_rate <- rep(0.2, noptim) # Elitism rate
#-----------------------------------------------
tuneParameters (  
  noptim,  
  nsamp1,  
  frame,  
  errors = errors,  
  strata = strata,  
  cens = NULL,  
  strcens = FALSE,  
  alldomains = FALSE,  
  dom = 1,  
  initialstrata,  
  addstratafactor,  
  minnumstr,  
  ..
)```
updateFrame

Updates the initial frame on the basis of the optimized stratification

Description

Once optimal stratification has been obtained, and new labels have been attributed to initial atomic strata ("newstrata"), it is important to report the new classification of units in the sampling frame by attributing new strata labels to each unit. By executing this function, a new frame will be obtained with the same structure of the old, but with the addition of a new stratum label. The initial frame must contain a variable named 'domainvalue' that indicates the same values of the domain that has been used with the 'optimizeStrata' function. If no domains have been defined, this variable will contains all 1’s, but it must exist.

Usage

updateFrame(frame, newstrata, writeFiles = FALSE)

Arguments

frame This is the (mandatory) dataframe containing the sampling frame.
newstrata This is the (mandatory) data frame containing the information related to the optimization applied to initial stratification (new labels applied to atomic strata). It is produced by executing the "updateStrata" function.
writeFiles Flag to write or not the new sampling frame into the working directory. Default is "FALSE"

Value

A dataframe containing the frame

Author(s)

Giulio Barcaroli
Examples

```r
## Not run
library(SamplingStrata)
data(swisserrors)
data(swissstrata)
# optimisation of sampling strata
solution <- optimizeStrata(
  errors = swisserrors,
  strata = swissstrata,
  cens = NULL,
  strcens = FALSE,
  initialStrata = 3000,
  addStrataFactor = 0.01,
  minnumstr = 2,
  iter = 60,
  pops = 20,
  mut_chance = 0.05,
  elitism_rate = 0.2,
  highvalue = 100000000,
  suggestions = NULL,
  writeFiles = FALSE)
# updating sampling strata with new strata labels
newstrata <- updateStrata(swissstrata, solution, writeFiles = TRUE)
# updating sampling frame with new strata labels
data(swissframe)
framenew <- updateFrame(frame = swissframe, newstrata = newstrata, writeFiles = TRUE)
```

## End(Not run)

**updateStrata**

Assigns new labels to atomic strata on the basis of the optimized aggregated strata

**Description**

Once optimal stratification has been obtained ("outstrata"), then we need to attribute new strata labels to each atomic stratum. By executing this function, a new dataframe "newstrata" will be obtained with the same structure of the old, ("strata") but with the addition of a new stratum label. By indicating "YES" to "writeFile" parameter, the dataframe "newstrata" will be written to a delimited file ("newstrata.txt"). Also a second delimited file ("strata_aggregation.txt") will be outputted, containing the indication of the relations bewteen atomic and aggregated strata.

**Usage**

```r
updateStrata(strata, solution, writeFiles = FALSE)
```
Arguments

strata This is the (mandatory) dataframe containing the information related to the atomic strata to which the optimisation has been applied to.

solution List obtained by the execution of the "optimizeStrata" function. The first element of the list is the vector of the indices corresponding to the optimal solution.

writeFiles Indicates if at the end of the processing the resulting strata will be outputted in a delimited file. Default is "FALSE".

Value

A dataframe containing the strata

Author(s)

Giulio Barcaroli

Examples

# # This is a toy example, and can be run
#

library(SamplingStrata)
data(errors)
data(strata)

# optimisation of sampling strata
solution <- optimizeStrata(
  errors = errors,
  strata = strata,
  cens = NULL,
  strcens = FALSE,
  initialStrata = 3000,
  addStrataFactor = 0.01,
  minnumstr = 2,
  iter = 30,
  pops = 20,
  mut_chance = 0.05,
  elitism_rate = 0.2,
  highvalue = 100000000,
  suggestions = NULL,
  writeFiles = FALSE)

# updating sampling strata with new strata labels
newstrata <- updateStrata(strata, solution)

# # The following example is realistic, but is time consuming
#
## Not run:
library(SamplingStrata)
data(swisserrors)
data(swissstrata)
var.bin

# optimisation of sampling strata
solution <- optimizeStrata(
    errors = swisserrors,
    strata = swissstrata,
    cens = NULL,
    strcens = FALSE,
    initialStrata = 3000,
    addStrataFactor = 0.01,
    minnumstr = 2,
    iter = 60,
    pops = 20,
    mut_chance = 0.05,
    elitism_rate = 0.2,
    highvalue = 10000000,
    suggestions = NULL,
    writefiles = FALSE)

# updating sampling strata with new strata labels
newstrata <- updateStrata(swissstrata, solution)

## End(Not run)

var.bin

Allows to transform a continuous variable into a categorical ordinal one by applying a modified version of the k-means clustering function in the 'stats' package.

Description

The optimization of a frame stratification is applicable only in presence of all categorical auxiliary variables in the frame. If one or more continuous auxiliary variables are in the frame, it is necessary to pre-process in order to convert them into categorical (ordinal) variables. The applied method is the "k-means" clustering method contained in the in "stats" package. This function ensures that the final result is in an ordered categorical variable.

Usage

var.bin(x,
    bins=3,
    iter.max=100)

Arguments

x Continuous variable to be transformed into a categorical one
bins Number of values of the resulting categorical variable
iter.max Maximum number of iterations of the clustering algorithm

Value

Binned variable
Examples

library(SamplingStrata)
data(swissmunicipalities)
data(swissframe)
swissframe$X1 <- var.bin(swissmunicipalities$POPTOT,bins = 18)
table(swissframe$X1)
tapply(swissmunicipalities$POPTOT,swissframe$X1,mean)
Index

*Topic datasets
   errors, 8
   strata, 16
   swisserrors, 17
   swissframe, 18
   swissmunicipalities, 19
   swissstrata, 20

*Topic survey
   adjustSize, 2
   bethel, 3
   buildFrameDF, 5
   buildStrataDF, 6
   checkInput, 7
   evalSolution, 8
   optimizeStrata, 10
   plotSamprate, 14
   selectSample, 15
   tuneParameters, 21
   updateFrame, 24
   updateStrata, 25
   var.bin, 27

adjustSize, 2

bethel, 3
buildFrameDF, 5
buildStrataDF, 6

checkInput, 7

errors, 8
evalSolution, 8

optimizeStrata, 10
plotSamprate, 14

selectSample, 15
strata, 16
swisserrors, 17
swissframe, 18

swissmunicipalities, 19
swissstrata, 20

tuneParameters, 21

updateFrame, 24
updateStrata, 25

var.bin, 27