Functions in SamplingStrata package - I

Marco Ballin, Giulio Barcaroli

20 May 2019

Procedural steps (categorical stratification variables)

- definition of the **sampling frame** data: identification of available auxiliary information, of the survey variables and of domains of interest;
- **manipulation of auxiliary information**: in case auxiliary variables are of the continuous type, they must be transformed into a categorical form;
- **construction of atomic strata**: on the basis of the categorical auxiliary variables available in the sampling frame, a set of strata can be constructed by calculating the Cartesian product of the values of all the auxiliary variables;
- characterization of each atomic stratum with the information related to the target variables: in order to optimise both strata and allocation of sampling units in strata, we need information on the distributions of the target variables (means and standard deviations) inside the different strata;
- **choice of the precision constraints** for each target estimate, possibly differentiated by domain;
- **optimization** of stratification and determination of required sample size and allocation in order to satisfy precision constraints on target estimates;
- analysis of the resulting optimized strata;
- association of **labels to sampling frame** units, each of them indicating the strata resulting by the optimization step;

- **selection of units** from the sampling frame;
- evaluation of the found optimal solution in terms of **expected precision and bias**.

Definition of the sampling frame data

As a first step, we have to define a frame dataframe containing the following information:

- a unique identifier of the unit ('id');
- the values of m auxiliary variables (named from X1 to Xm);
- the values of p target variables (named from Y1 to Yp);
- the value of the domain of interest for which we want to produce estimates (named 'domainvalue').

Definition of the sampling frame data

```
data(swissmunicipalities)
# Only two regions
swissmunicipalities <- swissmunicipalities[swissmunicipalities$REG < 3,]
# add a unique identifier
swissmunicipalities$id <- c(1:nrow(swissmunicipalities))
```

we get the 'swissmunicipalities' dataframe, that contains 2896 observations (each observation refers to a Swiss municipality). For reasons of processing time, here we consider only the first two regions (1502 observations). Among the others, there are the following variables (data are referred to 2003):

- REG: Swiss region
- Surfacesbois: wood area
- Surfacescult: area under cultivation
- Airbat: area with buildings
- Pop020: number of men and women aged between 0 and 19
- Pop2040: number of men and women aged between 20 and 39

Definition of the sampling frame data

Let us suppose we want to plan a survey whose target estimates are the totals of population by the first two age classes in each Swiss region. In this case, our Y variables will be:

- Y1: number of men and women aged between 0 and 19
- Y2: number of men and women aged between 20 and 39

As for the auxiliary variables (X's), we can use two of those characterising the area use (wood, and cultivated).

Finally, we want to produce estimates not only for the whole country, but also for each one of the two different regions we have selected.

Function 'buildFrameDF'

Function **buildFrameDF** permits to organize data of the sampling frame in a suitable mode for the process:

```
swissframe <- buildFrameDF(df = swissmunicipalities,
                           id = "id",
                           X = c("Surfacesbois",
                                 "Surfacescult"),
                           Y = c("Pop020",
                                 "Pop2040"),
                           domainvalue = "REG")
```

Function 'var.bin'

As the X variables are of the continuous type, first we have to reduce them in a categorical (ordinal) form. A suitable way to do so, is to apply a k-means clustering method, by using the **var.bin** function:

```
swissframe$X1 <- var.bin(swissframe$X1, bins=15)
swissframe$X2 <- var.bin(swissframe$X2, bins=15)
table(swissframe$X1)
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
## 206 209 174 154 136 113 115  99  78  64  58  44  25  19   8
```

```
table(swissframe$X2)
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
##  50 102 122 140 167 146 166 141 120  93  77  75  55  35  13
```

Function 'buildStrataDF'

The *strata* dataframe reports information regarding each atomic stratum in the population. There is one row for each stratum. The total number of strata is given by the number of different combinations of X's values in the frame. For each stratum, the following information is required:

- the identifier of the stratum (named 'stratum'), concatenation of the values of the X variables;
- the values of the m auxiliary variables (named from X1 to Xm) corresponding to those in the frame;
- the total number of units in the population (named 'N');
- a flag (named 'cens') indicating if the stratum is to be censused (=1) or sampled (=0);
- a variable indicating the cost of interviewing per unit in the stratum (named 'cost');
- for each target variable y, its mean and standard deviation, named respectively 'Mi' and 'Si');
- the value of the domain of interest to which the stratum belongs ('DOM1').

Function 'buildStrataDF'

It is possible to automatically generate the strata dataframe by invoking the **buildStrataDF** function.

Let us consider again the swissframe dataframe that we have built in previous steps. On this frame we can apply the function buildStrataDF:

```
swissstrata <- buildStrataDF(swissframe, progress = FALSE)
```

```
##
## Computations are being done on population data
##
## Number of strata:  316
## ... of which with only one unit:  70
```

Function 'buildStrataDF'

Variables 'cost' and 'cens' are initialised respectively to 1 and 0 for all strata. It is possible to give them different values:

- for variable 'cost', it is possible to differentiate the cost of interviewing per unit by assigning real values;
- for variable 'cens', it is possible to set it equal to 1 for all strata that are of the 'take-all' type (i.e. all units in that strata must be selected).

Choice of the precision constraints for each target estimate

The errors dataframe contains the accuracy constraints that are set on target estimates.

This means to define a maximum coefficient of variation for each variable and for each domain value.

Each row of this frame is related to accuracy constraints in a particular subdomain of interest, identified by the **domainvalue** value.

Choice of the precision constraints for each target estimate

In the case of the Swiss municipalities, we have chosen to define the following constraints:

```
swisserrors <- as.data.frame(list(DOM=rep("DOM1",2),
                                  CV1=c(0.08,0.10),
                                  CV2=c(0.12,0.15),
                                  domainvalue=c(1:2)
                             ))
swisserrors
```

```
##    DOM  CV1  CV2 domainvalue
## 1 DOM1 0.08 0.12           1
## 2 DOM1 0.10 0.15           2
```

Function 'checkInput'

Once having defined dataframes containing frame data, strata information and precision constraints, it is worth while to check their internal and reciprocal coherence.

It is possible to do that by using the function **checkInput**:

```
checkInput(errors = swisserrors,
           strata = swissstrata,
           sampframe = swissframe)
```

```
##
## Input data have been checked and are compliant with requirements
```

This function controls that

- the number of auxiliary variables is the same in the frame and in the strata dataframes;
- the number of target variables indicated in the frame dataframe is the same than the number of means and standard deviations in the strata dataframe,
- the number of target variables is the same than the number of coefficient of variations indicated in the errors dataframe.

Function 'optimizeStrata'

Once the strata and the constraints dataframes have been prepared, it is possible to apply the function that optimises the stratification of the frame, that is optimizeStrata. This function operates on all subdomains, identifying the best solution for each one of them. The fundamental parameters to be passed to optimizeStrata are:

- **errors**: the (mandatory) dataframe containing the precision levels expressed in terms of maximum allowable coefficients of variation that regard the estimates on target variables of the survey
- **strata**: 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
- **cens**: the (optional) dataframe containing the 'take-all' 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 'take-all' strata do exist or not. Default is FALSE
- **initialStrata**: the initial limit on the number of strata for each solution. Default is NA, and in this case it is set equal to the number of atomic 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**: indicates the probability that at each mutation the number of strata may increase with respect to the current value. Default is 0.0

Function 'optimizeStrata'

- **minnumstr**: indicates the minimum number of units that must be allocated in each stratum. Default is 2 iter: indicates the maximum number of iterations (= generations) of the genetic algorithm. Default is 50
- **pops**: dimension of each generations in terms of number of individuals to be generated. Default is 20
- **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 NA, in correspondence of which it is computed as 1/(vars+1) where vars is the length of elements in the solution
- **elitism_rate**: indicates the rate of better solutions that must be preserved from one generation to another. Default is 0.2
- **suggestions**: indicates one possible solution (from kmeans clustering or from previous runs) that will be introduced in the initial population. Default is NULL
- **parallel**: if TRUE the optimization in the different domains in carried out in parallel

Function 'optimizeStrata'

```
set.seed(123)
solution1 <- optimizeStrata(
  errors = swisserrors,
  strata = swissstrata,
  iter = 50,
  pops = 10,
  writeFiles = TRUE,
  showPlot = FALSE,
  parallel = FALSE)
```

```
##
##  *** Domain :  1   1
##  Number of strata :  157
## GA Settings
##   Population size       = 10
##   Number of Generations = 50
##   Elitism               = 2
##   Mutation Chance       = 0.00632911392405063
```

```
## Warning in sink(): non c'è nulla da eliminare
```

```
## [1] 279
```

Function ‘adjustSize’

After the optimization step, the final sample size is the result of the allocation of units in final strata. This allocation is such that the precision constraints are expected to be satisfied. Actually, three possible situations may occur:

1. the resulting sample size is acceptable;
2. the resulting sample size is to high, it is not affordable with respect to the available budget;
3. the resulting sample size is too low, the available budget permits to increase the number of units.

In the first case, no action is required. In the second case, it is necessary to reduce the number of units, by equally applying the same reduction rate in each stratum. In the third case, we could either to set more tight precision constraints, or proceed to increase the sample size by applying the same increase rate in each stratum. This increase/reduction process is iterative, as by applying the same rate we could find that in some strata there are not enough units to increase or to reduce.

Function 'adjustSize'

The function *adjustSize* permits to obtain the desired final sample size. Let us suppose that the obtained sample size is not affordable.

We can reduce it by executing the following code:

```
adjustedStrataLow <- adjustSize(size=150,
                                strata=solution1$aggr_strata,
                                cens=NULL)
```

```
##
##  170
##  159
##  158
##  158
##  Final adjusted size:  158
```

```
sum(adjustedStrataLow$SOLUZ)
```

```
## [1] 158
```

Function 'adjustSize'

Instead, if we want to increase the size because the budget allows to do this, then this is the code:

```
adjustedStrataHigh <- adjustSize(size=350,
                                 strata=solution1$aggr_strata,
                                 cens=NULL)
```

```
##
##  342
##  343
##  343
##  Final adjusted size:  343
```

```
sum(adjustedStrataHigh$SOLUZ)
```

```
## [1] 343
```

Function ‘adjustSize’

The difference between the desired sample size and the actual adjusted size depends on the number of strata in the optimized solution. Consider that the adjustment is performed in each stratum by taking into account the relative difference between the current sample size and the desired one: this produces an allocation that is expressed by a real number, that must be rounded.

The higher the number of strata, the higher the impact of the rounding in all strata on the final adjusted sample size.

Function 'expected_CV'

We can check immediately the effect of the variations on the size of the sample by using the function 'expected_CV':

```
expected_CV(solution1$aggr_strata)
```

```
##      cv(Y1) cv(Y2)
## DOM1  0.078  0.076
## DOM2  0.099  0.102
```

```
expected_CV(adjustedStrataLow)
```

```
##      cv(Y1) cv(Y2)
## DOM1  0.149  0.175
## DOM2  0.150  0.170
```

```
expected_CV(adjustedStrataHigh)
```

```
##      cv(Y1) cv(Y2)
## DOM1  0.063  0.060
## DOM2  0.081  0.084
```

```
swisserrors
```

```
##    DOM  CV1  CV2 domainvalue
## 1 DOM1 0.08 0.12           1
## 2 DOM1 0.10 0.15           2
```

Function 'updateStrata'

This function has two purposes:

1. instrumental to the processing of the sampling frame (attribution of the labels of the optimized strata to the population units);
2. analysis of the aggregation of the atomic strata obtained in the optimized solution.

The function **updateStrata** assigns the labels of the new strata to the initial one in the dataframe *strata*, and produces:

- a new dataframe named *newstrata* containing all the information in the *strata* dataframe, plus the labels of the new optimized strata;
- a table, contained in the dataset *strata_aggregation.txt*, showing in which way each optimized stratum aggregates the auxiliary variables X's.

Function 'updateStrata'

The function is invoked in this way:

```
newstrata <- updateStrata(swissstrata,
                          solution1,
                          writeFiles = TRUE)
head(newstrata,3)
```

```
##      STRATO  N        M1        M2        S1        S2 COST CENS DOM1 X1
## 1*1     1*1 11 1505.9091 2259.0000 1545.2482 2404.2180    1    0    1  1
## 1*11   1*11  1  432.0000  462.0000    0.0000    0.0000    1    0    1  1
## 1*2     1*2 17  237.2353  313.4706  390.3955  593.6508    1    0    1  1
##      X2 LABEL STRATUM
## 1*1   1     1     1*1
## 1*11 11     2    1*11
## 1*2   2     3     1*2
```

Function 'updateStrata'

Now, the atomic strata are associated to the aggregate strata defined in the optimal solution, by means of the variable *LABEL*. If we want to analyse in detail the new structure of the stratification, we can look at the *strata_aggregation.txt* file:

```
strata_aggregation <- read.delim("strata_aggregation.txt")
head(strata_aggregation)
```

```
##   DOM1 AGGR_STRATUM X1 X2
## 1    1             1  1  1
## 2    1             1  7  9
## 3    1             1  9  9
## 4    1             1  9 14
## 5    1             1 11  7
## 6    1             2  1 11
```

In this structure, for each aggregate stratum the values of the X's variables in each contributing atomic stratum are reported. It is then possible to understand the meaning of each aggregate stratum produced by the optimization.

Function 'updateFrame'

To update the frame units with new stratum labels (combination of the new values of the auxiliary variables X's), we make use of the **updateFrame** function:

```
framenew <- updateFrame(swissframe,
                        newstrata,
                        writeFiles=FALSE)
```

The execution of this function produces a dataframe framenew, and also a file (named framenew.txt) with the labels of the new strata produced by the optimization step.

Function 'selectSample'

Once the sampling frame has been added the indication of the optimized aggregated strata (variable *LABEL* in *framenew* dataframe), it is possible to select the desired sample. The allocation vector (i.e. the number of units to be selected in each stratum) is contained in variable *SOLUZ* in the *solution$aggr_strata* element of the list *solution* (or in the list which is the result of the adjustment of the sample size).

The selection of the sample is obtained by applying the function **selectSample**:

```
sample <- selectSample(framenew,
                       solution1$aggr_strata,
                       writeFiles = FALSE)
```

```
##
## *** Sample has been drawn successfully ***
##  258  units have been selected from  56  strata
##
## ==> There have been  9  take-all strata
## from which have been selected  41 units
```

Function ‘selectSample’

By indicating ‘writeFiles = TRUE’, two different .csv files are produced:

- *sample.csv* containing the units of the frame that have been selected, together with the weight that has been calculated for each one of them;
- *sample.chk.csv* containing information on the selection: for each stratum, the number of units in the population, the planned sample, the number of selected units, the sum of their weights that must equalise the number of units in the population.

Function 'evalSolution'

In order to be confident about the quality of the found solution, the function evalSolution allows to run a simulation, based on the selection of a desired number of samples from the frame to which the stratification, identified as the best, has been applied. The user can invoke this function also indicating the number of samples to be drawn:

```
eval <- evalSolution(framenew,
                     solution1$aggr_strata,
                     nsampl=500,
                     writeFiles=TRUE,
                     progress=FALSE)
```

Function 'evalSolution'

For each drawn sample, the estimates related to the Y's are calculated. Their mean and standard deviation are also computed, in order to produce the CV related to each variable in every domain. These CV's can be inspected and compared to the constraints:

```
eval$coeff_var
```

domain	cv(Y1)	cv(Y2)
1	0.0912	0.0872
2	0.1086	0.1126

```
swisserrors
```

```
##    DOM  CV1  CV2 domainvalue
## 1 DOM1 0.08 0.12           1
## 2 DOM1 0.10 0.15           2
```