Code implementation of the subsampling algorithms.

Introduction

We implemented 3 algorithms, defined "Distance procedure", the "Probability procedure", and the "Uniformity procedure" with the aim of producing sub-samples with specific distributional properties starting from an initial, collection of data which is possibly biased in terms of generalizability to a target population. In the following document we present the R code to implement the algorithms. The specific implementation described here is focused on the Italian PPS study on HAI/AMU prevalence in acute care hospitals but can easily translated to different settings.

The methods try to change the distributional characteristics of a sample by producing a sub-sample whose units are chosen according to some characteristics of interest. Two of these methods, the Distance procedure and the Probability procedure, are aimed at generating a representative sample of a target population, by using reference data at the population level with information on the distribution of relevant characteristics of the observational units. The Uniformity procedure instead generate a sample which is uniform in relation to such characteristics, therefore it does not require population reference data.

This document and all the relative material is available at https://github.com/AD-Papers-Material/SubsamplingMethods.

Input data

The algorithms take as input a database with a statistical unit for each row and the characteristics of interest as columns. Furthermore, a column with an unique ID for each unit is useful for post-hoc checks and mandatory for the Distance procedure. Our algorithms also accept the Quality Score (QS, cfr. Methods) as additional characteristics, but they can be easily modified to remove such feature.

The Probability and the Distance procedures also requires population level reference data, with the same structure. If individual level data is not available at the population, simulated data can be generated given access to the joint distribution of the considered characteristics, ensuring that the simulated dataset is large enough to limit random variation.

Our test case data use acute care hospitals as observational units and hospital size (number of acute care beds) and region of location as characteristics of interest. As a reference, the hospital are also grouped in three hospital size categories; the same categorization is used in the manuscript. We provide simulated sample data for the testing of the procedures at https://github.com/AD-Papers-Material/SubsamplingMethods.

```
# Simulated sample data retaining the real sample characteristics
str(Sample.Data, vec.len = 3)
#> 'data.frame':    143 obs. of  5 variables:
#>  $ Region: chr  "regione piemonte" "regione piemonte" "regione piemonte" ...
#>  $ Beds  : int  183 172 157 179 85 133 189 182 ...
#>  $ QS    : num  101 628 147 109 ...
#>  $ Class : chr  "< 200" "< 200" "< 200" ...
#>  $ Code  : int  1 2 3 4 5 6 7 8 ...

# Official list of Italian acute hospitals, updated to 2016
str(Reference.Data, vec.len = 3)
#> 'data.frame':    963 obs. of  4 variables:
#>  $ Code  : int  10007 10010 10012 10612 10653 10655 10003 10011 ...
#>  $ Beds  : int  258 73 22 105 9 96 337 368 ...
#>  $ Class : chr  "200 - 500" "< 200" "< 200" ...
#>  $ Region: chr  "regione piemonte" "regione piemonte" "regione piemonte" ...
```

General aspects and notation

The procedures at the moment can utilize only **two characteristics** of a sample and these need to be **categorical variables**. Therefore continuous characteristics (specifically, hospital size in this case) are discretized in quantiles by the algorithms before use. The number of quantiles to split continuous features into is an input to the algorithms.
The hospitals are then grouped into *blocks* according to the characteristics of interest: in this study, we used *location/hospital size blocks*, defined by the Italian region ($Region$) and the quantile of number of acute beds ($HSize$) the hospitals fall into. The region and the hospital size quantile allow the definition of a joint discrete probability distribution which is used by the algorithms.
For each block $block_i$ a probability $p_i = Pr(hospital|Region, HSize)$ is defined, either given a sample ($p_{i,sample}$) or the whole country ($p_{i,country}$) which indicate the fraction of hospital in a block over the total.
All the algorithms are constrained by a parameter $N_{required}$ which defines the size of the final sub-sample.
As mentioned above, the algorithms may use the Quality Score QS as further discriminant in the selection; to avoid using the QS without code modification is sufficient to assign the same value (a positive number) to all hospitals. Note that lower QS implies better data quality.

Unformity procedure

This procedure sub-samples hospitals trying to obtain an equal proportion of hospitals in every block, by iteratively choosing one hospital from each block. This is the general implementation:

- a candidate list is created from the original sample;
- the hospitals in the list are permuted randomly or ordered in ascending order by QS (the lower the score, the higher the data quality);
- the first hospital is selected;
- all other hospitals belonging to the same block of the selected hospital are removed from the candidate list;
- the process is repeated until there are still available blocks in the candidate list;
- once one hospital from each blocks has been chosen, the hospitals from all the blocks are made available again in the list, apart from those already selected in the sample;
- continue until $N_{required}$ is reached.

For the uniform procedure, we discretized the number of beds only into 4 quantiles since it was less relevant to build a precise discrete probability distribution.

```
uniform.sampling <- function(Input.Sample, n.required, n.quantiles = 4, use.QS = T){

    library(dplyr)
    library(Hmisc)

    # Prepare data by discretizing continuous variables like the number of acute
    # beds and by changing QS to a fixed value if not to be used
    Hospitals <- Input.Sample %>%
        transmute(
            Code, Region,
            Beds = Hmisc::cut2(Beds, g = n.quantiles),
            QS = if (use.QS) QS else 1)

    Selected.hospitals <- c()
    Candidates <- data.frame()

    for (i in 1:n.required) { # Until n.required is reached..

        # If the candidate list is empty, rebuilt it from the non-selected hospitals
        if (nrow(Candidates) == 0) {
```

```
            Candidates <- Hospitals %>%
                # Remove already selected hospitals
                filter(!(Code %in% Selected.hospitals)) %>%
                # Permute order, useful only if QS is not used
                sample_frac() %>%
                # Arrange by QS ascending, best hospitals first
                arrange(QS)
        }

        # Extract the first hospital of the temporary list and add it to the list of
        # selected hospitals
        Extracted.hospital <- Candidates[1,]
        Selected.hospitals <- c(Selected.hospitals, Extracted.hospital$Code)

        # Remove from the temporary list all hospitals in the same location/size block
        # of the extracted hospital
        Candidates <- Candidates %>%
            filter(
                !(Region %in% Extracted.hospital$Region),
                !(Beds %in% Extracted.hospital$Beds)
                )
    }

    # Filter the initial data by the selected hospital codes
    Input.Sample %>% filter(Code %in% Selected.hospitals)
}

## Examples

# Create a subsample of 55 hospitals using the QS
# subsample.uniform(Sample.Data, n.required = 55)

# The same but this time hospitals are chosen randomly
# subsample.uniform(Sample.Data, n.required = 55)
```

Probability procedure

This algorithm uses information from a population level list (Reference Data) to built a discrete probability distribution representative of the target population and then uses it to create a representative sub-sample. The hospitals are selected according to how representative is the block they belong to at the country level. The QS is used to weight such representativeness. The weight of the QS itself can be weighted.

- the blocks are identified in the Reference Data and for each block i the probability $p_{i,country} = Pr(hospital|Region, HSize)$ is computed as the proportion of hospitals in the block over the total;
- these probabilities are assigned to the relative blocks in the sample;
- a score is computer for each hospital j as $score_j = p_{j,country}(1 - scaled.QS_j)^w$ where:
- $p_{j,country}$ is the probability of the block of the hospital j at the country level;
- $scaled.QS_j$ is the QS of the hospital j after that all QS have been rescaled to the range [0,1], with 1 representing the worst quality score and 0 the best. $(1 - scaled.QS_j)$ reweighs the probability of being included of a hospital using the quality of the data;
- w allows scaling the importance of the QS in the selection, with $w = 0$ removing its influence;
- finally, $score_j$ is used to order the hospitals and the first $N_{required}$ get selected. In alternative, the score can be used as a weight for selecting the hospitals by random sampling.

```r
probability.sampling <- function(Input.Sample, Reference.Data, n.required,
    n.quantiles = 10, QS.weight = 1,
    method = c('arrange', 'random')){

    library(dplyr)
    library(Hmisc)
    library(magrittr)
    library(scales)

    method <- match.arg(method)

    # Definition of quantiles in the distribution of number of beds according to
    # reference data
    quantiles <- quantile(Reference.Data$Beds, seq(0, 1, length.out = n.quantiles)) %>%
        round

    P_country <- Reference.Data %>%
        count(Block = Hmisc::cut2(Beds, quantiles) %>% paste('-', Region)) %>%
        mutate(Prob = n / sum(n)) %>%
        with(magrittr::set_names(Prob, Block))

    Selection <- Input.Sample %>%
        mutate(
            # Identification of the country level blocks in the sample
            Block = Hmisc::cut2(Beds, quantiles) %>% paste('-', Region),
            # Association of P_country to the hospital in the sample
            Prob = P_country[Block],
            # Creation of the quality weight after rescaling of the QS
            QS.rescale = 1 - scales::rescale(QS),
            # Definition of the final score
            Score = Prob * QS.rescale^QS.weight
        ) %>%
        filter(!is.na(Score)) # Remove blocks that do not appear in the national list

    if (method == 'arrange') {
        Selection %>%
            arrange(desc(Score)) %>%
            head(n.required)
    } else {
        slice_sample(Selection, n = n.required, weight_by = Score)
    }
}

## Examples

# Create a subsample of 55 hospitals
# subsample.probability(Sample.Data, Reference.Data, n.required = 55)

# Set QS.weight to 0 to not use the QS in the sampling
# subsample.uniform(Sample.Data, Reference.Data, n.required = 55, QS.weight = 0)

# Create a subsample of 55 hospitals with weighted random selection
# subsample.probability(Sample.Data, Reference.Data, n.required = 55, method = 'random')
```

Distance procedure

As with the Probability procedure, the aim is to produce sub-samples that try to reproduce the distributional characteristics of a target population. The advantage of this procedure is that it is particularly appropriate when the original sample is particularly distorted in relation of the characteristics of interest. That is, some blocks are too much underrepresented in the original sample compared to target population and therefore the Probability procedure cannot find enough units in them to reproduce their relative distribution at the population level.
This procedure attempts to solve the problem by oversampling blocks which are similar to the underrepresented ones: if a block cannot provide enough units as required by the expected representativeness at the population level, samples are collected from blocks with similar characteristics. Similarity is defined through ad-hoc distance measures between blocks for each characteristics.
The procedure considers the sample characteristics of interest sequentially, implicitly giving more weight to one or another. Such priority can be passed as an argument.

This algorithm is more complex than the previous two procedures. Here's the general implementation:

- for each i block the expected number of sampled units is computed as $N_{i,expected} = \texttt{Round}(p_{i,country} \times N_{required})$, with $p_{i,country}$ being the target population level representativeness of a location/size block;
- in case $N_{required}$ is not reached, i.e., $(\sum_{j=1}^{n} N_{i,expected}) < N_{required}$, the $N_{i,expected}$ of specific blocks is increased by one unit until $N_{required}$ is achieved. The units are added to blocks whose the fractional part of $N_{i,expected}$ before rounding is higher than zero $((p_{i,country} \times N_{required}) - N_{i,expected} > 0)$, starting by the higher values (that is, those closer to be rounded up);
- for each block, starting by those requiring more hospitals, $N_{i,sample}$ hospitals belonging to it are selected;
- if $N_{i,sample} < N_{i,expected}$ (i.e., the block is underrepresented), hospitals from similar blocks are *assigned* to it, becoming not available for the other blocks. The final number of hospital assigned to a block is $N_{i,assigned}$ and may comprise hospitals actually belonging to other blocks, for example a hospital from the Piedmont region is assigned to the Lumbardy region since it is a close region. Similarity is computed via characteristic-specific algorithms (check R code online at https://github.com/AD-Papers-Material/SubsamplingMethods), and is evaluated with a priority chosen by the user; that is either hospital size similarity is considered followed by location similarity or vice-versa. In case of ties, the QS is used or random selection if the QS is equal.

At this point an initial sub-sample is achieved, in which a block is assigned to each hospital to be included in the sub-sample, while the "unassigned" label is assigned to those not to include. The initial sample may have an overrepresentativeness of blocks with more hospitals, since they will tend to cede more hospitals to underrepresented blocks. A random process attempts at further improve the sample by avoid reducing unbalance in favor of oversampling blocks, by swapping hospitals in and out of the sample if the swaps decreases the distance between the actual and the assigned block of an hospitals. The swaps are only accepted though if the overall fit of the subsample with the target population does not decrease:

- for a chosen number of iteration, a random hospital is chosen and the set of candidates hospitals to swap is generated from those with opposite assignment status (i.e.: if a hospital included in the initial sub-sample is randomly selected, than the candidates are chosen among those not included and vice-versa);
- For each candidate swap, the assigned region and size class are inverted with the randomly selected hospital and the distance between the new assigned block and the actual block of the hospitals is computed. Unassigned hospitals are always at distance of one from the actual block;
- It is chosen the candidate swap that minimize the sum of distances arranged by size and location (with order according to priority) providing an improvement in the distance in at least one of the characteristics or a decrease of the QS in the subsample. For example (considering only location for simplicity), if the random chosen hospital is from region Tuscany but is assigned to Sicily (distance: 2) and the candidate is a hospital from Sicily but unassigned (distance: 1), the swap would unassign the first hospital (distance: 1) and the other to Sicily (distance: 0): the final distance is (1 + 0) - (2 + 1) = -2, therefore implying an overall improvement of the sample;
- the swap is finally accepted if the fit of the subsample after the swap does not decrease. The fit is

evaluated as described in the statistical analysis section of the manuscript and the relative implementation is presented online.

```
subsample.distance <- function(Input.Sample, Reference.Data, n.required,
     n.quantiles = 10, priority = c('size', 'location'),
     method = c('logLik', 'spearman'), reallocate = T,
     realloc.steps = 2000, steps.to.try = 2 * n.required){

    library(dplyr)
    library(Hmisc)
    library(pbapply)
    library(parallel)

    priority <- match.arg(priority)
    method <- match.arg(method)

    # Definition of quantiles in the distribution of number of beds according to
    # reference data
    quantiles <- quantile(Reference.Data$Beds, seq(0, 1, length.out = n.quantiles)) %>%
        round()

    Hospitals <- Input.Sample %>%
        transmute(
            Code, QS,

            # Hospital size is quantized according to reference data and location/size
            # blocks are defined
            Region,
            Size.class = Hmisc::cut2(Beds, quantiles),
            Block = paste(Size.class, '-', Region),

            # To each hospital is assigned a fictitious region and size class, the
            # initial value is 'unassigned' which means not selected in the sample
            Region.assigned = 'unassigned',
            Size.class.assigned = factor('unassigned',
                levels = c(levels(Size.class), 'unassigned')),

            # The distance associated to the 'unassigned' status is 1
            Distance = 1,
            Size.diff = 1
        )

    # Blocks are identified in the refernce data too
    Reference.Data <- Reference.Data %>% mutate(
        Size.class = Hmisc::cut2(Beds, quantiles),
        Block = paste(Size.class, '-', Region)
    )

    # For each block the expected number or required hospital is computed,
    # translating the target population proportion to the required sample size
    Blocks <- count(Reference.Data, Block, name = 'N.country') %>%
        mutate(
            F.expected = N.country/sum(N.country) * n.required,
            N.expected = round(F.expected),
```

```
        # If the required sample size is not reached, additional units are added
        # to blocks closer to be rounded up
        N.expected = if (sum(N.expected) == n.required) N.expected else {
            delta <- (F.expected - N.expected)
            rank <- row_number(-delta) # rank blocks by fractional part
            missing <- 1:(n.required - sum(N.expected))
            case_when(
                rank %in% missing & delta > 0 ~ N.expected + 1,
                T ~ N.expected
            )
        }
    ) %>%
    # Join the expected block numerosity with the Reference data, to associate
    # information about region and size class
    left_join(Reference.Data[,c('Region', 'Size.class', 'Block')], by = 'Block') %>%
    distinct %>%
    # Blocks that need more hospital first
    arrange(desc(F.expected))

## Reservoir of available hospitals to assign
Available.hospitals <- Hospitals %>%
    slice_sample(prop = 1)

# First assignment: associate hospitals to block until N_expected is reached;
# if not enough hospitals for a block are available in the sample, uses
# hospitals from "similar" blocks.
Hospitals.reassigned <- pblapply(which(Blocks$N.expected != 0), function(i) {
    Block <- Blocks[i,]

    if (Block$N.expected == 0) return(NULL) # No hospitals required for this block

    # Compute "distances" of the block with all hospitals
    Available.hospitals$Distance <- get.location.distance(
        rep(Block$Region, nrow(Available.hospitals)),
        Available.hospitals$Region
    )
    Available.hospitals$Size.diff <- get.onedim.distance(
        rep(Block$Size.class, nrow(Available.hospitals)),
        Available.hospitals$Size.class
    )

    # Arrange hospitals by distance from the block, prioritizing the
    # chosen characteristic. Hospitals belonging to the block will have
    # zero distance and will be prioritize. QS is used in case of ties
    Selected.hospitals <- Available.hospitals %>% {
        if (priority == 'size') {
            arrange(., Size.diff, Distance, QS)
        } else {
            arrange(., Distance, Size.diff, QS)
        }
    } %>%
        # Select the N_expected hospitals for the block, ordered by distance.
        head(Block$N.expected) %>%
```

```
        # "Assign" the block size location to the selected hospitals
        mutate(
            Region.assigned = !!Block$Region,
            Size.class.assigned = !!Block$Size.class
            )

    # Remove the selected hospitals from those still available for sampling
    Available.hospitals <<- Available.hospitals %>%
        filter(!(Code %in% Selected.hospitals$Code))

    Selected.hospitals
}) %>% bind_rows()

score <- get.distr.fit(Hospitals.reassigned, Reference.Data, method = method)
message('First distribution score: ', score)

# Rebuild the dataset with assigned and not hospitals
Hospitals <- bind_rows(
    Hospitals.reassigned,
    Hospitals %>% filter(!(Code %in% Hospitals.reassigned$Code))
)

if (reallocate) {
    message('Reallocation')

    # Windows doesn't support forking for parallelization, and socketing is
    # actually slower
    options(mc.cores =  if (Sys.info()[['sysname']] == 'Windows') {
        1
    } else parallel::detectCores())

    steps.wo.improvements <- 0

    # Initiate random swapping of hospitals to improve the fit
    pblapply(1:realloc.steps, function(step) {

        steps.wo.improvements <<- steps.wo.improvements + 1

        # iI too many steps are gone without improvement, stops
        if (steps.wo.improvements > steps.to.try) {
            return()
        }

        # Select a random hospital
        Evaluated.hospital <- slice_sample(Hospitals, n = 1)

        # Evaluate swaps with hospital in the opposite assignment status
        Candidate.swaps <- Hospitals %>%
            filter(if (Evaluated.hospital$Region.assigned == 'unassigned') {
                Region.assigned != 'unassigned'
            } else Region.assigned == 'unassigned')

        # Evaluate all candidate swaps, computing the change in distance between
```

```
        # the assigned and the actual block. mclapply allows parallel looping on
        # Unix OS, but reverts to lapply on Windows
        Best.swap <- mclapply(1:nrow(Candidate.swaps), function(i) {
            Candidates <- bind_rows(Evaluated.hospital, Candidate.swaps[i,])

            # Compute the distances after swapping
            Candidates$swapped.dist <- get.location.distance(
                Candidates$Region, rev(Candidates$Region.assigned)
            )
            Candidates$swapped.size.diff <- get.onedim.distance(
                Candidates$Size.class, rev(Candidates$Size.class.assigned)
            )
            Candidates$swapped.QS <- rev(Candidates$QS)

            data.frame(
                Evaluated = Candidates$Code[1],
                Candidate = Candidates$Code[2],

                # Compute the delta in the distances, a negative value
                # means improvement
                delta.location = with(Candidates, sum(swapped.dist) - sum(Distance)),
                delta.size = with(Candidates, sum(swapped.size.diff) - sum(Size.diff)),
                delta.QS = with(
                    filter(Candidates, Region.assigned != 'unassigned'),
                    swapped.QS - QS
                )
            )
        }) %>% bind_rows() %>%
            # Arrange the swaps according to priority
            {
                if (priority == 'size') {
                    arrange(., delta.size, delta.location, delta.QS)
                } else {
                    arrange(., delta.location, delta.size, delta.QS)
                }
            } %>% head(1) # And select the best swap

        # If at least one of the distances real/assigned blocks is imporoved proceed
        if (with(Best.swap, delta.location < 0 | delta.size < 0 | delta.QS < 0)) {

            # Select the hospitals of the best swap and invert their assigned
            # blocks. Then recompute the distances
            Best.swap.hospitals <- Hospitals %>%
                filter(Code %in% unlist(Best.swap[,1:2])) %>%
                mutate(
                    across(c(Region.assigned, Size.class.assigned), rev),
                    Distance = get.location.distance(Region, Region.assigned),
                    Size.diff = get.onedim.distance(Size.class, Size.class.assigned)
                )

            # Remove the swapped hospitals from the sample and add them again with
            # the updated assignment
            Hospital.proposal <- Hospitals %>%
```

```
                    filter(Code %nin% Best.swap.hospitals$Code) %>%
                    rbind(Best.swap.hospitals)

                # Compute the fit with the target population before and after the swap
                score_before <- get.distr.fit(
                    filter(Hospitals, Region.assigned != 'unassigned'),
                    Reference.Data, method = method)
                score_after <- get.distr.fit(
                    filter(Hospital.proposal, Region.assigned != 'unassigned'),
                    Reference.Data, method = method)

                # If the the swap didn't decrease the fit, the swap is accepted
                if (score_after >= score_before) {

                    message('\nStep: ', step)
                    message('Improvement after ', steps.wo.improvements, ' steps')
                    message('Score after reassignment:', score_after)

                    steps.wo.improvements <<- 0

                    Hospitals <<- Hospital.proposal
                }
            }
        })

    }

    # Return the subampled data
    Input.Sample %>%
        filter(Code %in% filter(Hospitals, Region.assigned != 'unassigned')$Code)
}

## Examples

# Create a subsample of 55 hospitals prioritizing hospital size, without
# reallocation after the initial sampling

# subsample.distance(Sample.Data, Reference.Data, n.required = 55,
#   reallocate = F)

# This time prioritize location and perform reallocation. (Warning: it takes
# time!)

# subsample.distance(Sample.Data, Reference.Data, n.required = 55,
#     priority = 'location')
```