Package 'surveysd'

November 17, 2017

Type Package

Title Estimates standard errors in complex surveys

Version 0.1.0

Author Johannes Gussenbauer, Alexander Kowarik, Matthias Till

Maintainer Johannes Gussenbauer `<Johannes.Gussenbauer@statistik.gv.at>`

Description Estimate point estimates and their standard errors in complex household surveys using bootstrap replicates. Bootstraping is layed out for surveys with rotating panel design.

Encoding UTF-8

LazyData true

License GPL (>= 2)

Imports Rcpp (>= 0.12.12),data.table,survey,simPop

LinkingTo Rcpp

RoxygenNote 6.0.1

R topics documented:

`calc.stError` *Calcualte point estimates and their standard errors using bootstrap weights.*

Description

Calculate point estimates as well as standard errors of variables in surveys. Standard errors are estimated using bootstrap weights (see `bootstrap.rep` and `recalib`). In addition the standard error of an estimate can be calcualted using the survey data for 3 or more consecutive years, which results in a reduction of the standard error.

Usage

```
calc.stError(dat,weights="hgew",b.weights=paste0("w",1:1000),year="jahr",var="povmd60",
                    fun="weightedRatio",cross_var=NULL,year.diff=NULL,
                    year.mean=3,bias=FALSE,add.arg=NULL,size.limit=20,cv.limit=10)
```

Arguments

`dat`	either data.frame or data.table containing the survey data. Surveys can be a panel survey or rotating panel survey, but does not need to be. For rotating panel survey bootstrap weights can be created using `bootstrap.rep` and `recalib`.
`weights`	character specifying the name of the column in `dat` containing the original sample weights. Used to calculate point estimates.
`b.weights`	character vector specifying the names of the columns in `dat` containing bootstrap weights. Used to calculate standard errors.
`year`	character specifying the name of the column in `dat` containing the sample years.
`var`	character vector containing variable names in `dat` on which `fun` shall be applied for each sample year.
`fun`	character specifying the function which will be applied on `var` for each sample year. Possible arguments are `weightedRatio,weightedSum,sampSize,popSize` as well as any other function which returns a double or integer and uses weights as its second argument.
`cross_var`	character vectors or list of character vectors containig variables in `dat`. For each list entry `dat` will be split in subgroups according to the containing variables as well as `year`. The pointestimates are then estimated for each subgroup seperately. If `cross_var=NULL` the data will split into sample years by default.
`year.diff`	character vectors, defining years for which the differences in the point estimate as well it's standard error is calculated. Each entry must have the form of `"year1 - year2"`. Can be NULL
`year.mean`	odd integer, defining the range of years over which the sample mean of point estimates is additionally calcualted.
`bias`	boolean, if TRUE the sample mean over the point estimates of the bootstrap weights is returned.
`add.arg`	character specifying additional arguments for `fun`. Can be `NULL`.
`size.limit`	integer defining a lower bound on the number of observations on `dat` in each group defined by `year` and the entries in `cross_var`. Warnings are returned if the number of observations in a subgroup falls below `size.limit`. In addition the concerned groups are available in the function output.
`cv.limit`	non-negativ value defining a upper bound for the standard error in relation to the point estimate. If this relation exceed `cv.limit`, for a point estimate, they are flagged and available in the function output.

Details

`calc.stError` takes survey data (`dat`) and returns point estimates as well as their standard Errors defined by `fun` and `var` for each sample year in `dat`. `dat` must be household data where household members correspond to multiple rows with the same household identifier. The data should at least containt the following columns:

- Column indicating the sample year;

- Column indicating the household ID;
- Column containing the household sample weights;
- Columns which contain the bootstrap weights (see output of recalib);
- Columns listed in var as well as in cross_var

For each variable in var as well as sample year the function fun is applied using the original as well as the bootstrap sample weights.
The point estimate is then selected as the result of fun when using the original sample weights and it's standard error is estimated with the result of fun using the bootstrap sample weights.

fun can be any function which returns a double or integer and uses sample weights as it's second argument. The predifined options are weightedRatio,weightedSum,sampSize and popSize, for wich sampSize and popSize indicate the sample and population size respectively.

For the option weightedRatio a weighted ratio (in %) of var is calculated for var equal to 1, e.g sum(weight[var==1])/sum(weight[!is.na(var)])*100.

If cross_var is not NULL but a vector of variables from dat then fun is applied on each subset of dat defined by all combinations of values in cross_var.
For instance if cross_var = "sex" with "sex" having the values "Male" and "Female" in dat the point estimate and standard error is calculated on the subsets of dat with only "Male" or "Female" value for "sex". This is done for each value of year. For variables in cross_var which have NAs in dat the subset containing the missings will be discarded.
When cross_var is a list of character vectors subsets of dat and the following estimation of the point estimate, including the estimate for the standard error, are calculated for each list entry.

When defining year.diff the difference of point estimates between years as well their standard errors are calculated.
The entries in year.diff must have the form of *year1 - year2* which means that the results of the point estimates for *year2* will be substracted from the results of the point estimates for *year1*.

Specifying year.mean leads to an improvement in standard error by averaging the results for the point estimates, using the bootstrap weights, over year.mean years. Setting, for instance, year.mean = 3 the results in averaging these results over each consecutive set of 3 years.
Estimating the standard error over these averages gives in an improved estimate of the standard error for the central year, which was used for averaging.

Setting bias to TRUE returns the calculation of a mean over the results from the bootstrap replicates. In the output the corresponding columns is labeled *_mean* at the end.

If fun needs more arguments they can be set in add.arg.

The parameter size.limit indicates a lower bound of the sample size for subsets in dat created by cross_var. If the sample size of a subset falls below size.limit a warning will be displayed.
In addition all subsets for which this is the case can be selected from the output of calc.stError with $smallGroups.
With the parameter cv.limit one can set an upper bound on the coefficient of variantion. Estimates which exceed this bound are flagged with TRUE and are available in the function output with $cvHigh. cv.limit must be a positive integer and is treated internally as %, e.g. for cv.limit=1 the estimate will be flagged if the coefficient of variantion exceeds 1%.

When specifying year.mean, the decrease in standard error for choosing this method is internally calcualted and a rough estimate for an implied increase in sample size is available in the output with `$stEDecrease`. The rough estimate for the increase in sample size uses the fact that for a sample of size n the sample estimate for the standard error of most point estimates converges with a factor $1/\sqrt{n}$ against the true standard error σ.

Value

Returns a list containing:

- `Estimates`: data.table containing yearly, differences and/or k year averages for estimates of `fun` applied to `var` as well as the corresponding standard errors, which are calculated using the bootstrap weights.
- `smallGroups`: data.table containing groups for which the number of observation falls below `size.limit`.
- `cvHigh`: data.table containing a boolean variable which indicates for each estimate if the estimated standard error exceeds `cv.limit`.
- `stEDecrease`: data.table indicating for each estimate the theoretical increase in sample size which is gained when averaging over k years. Only returned if `year.mean` is not `NULL`.

Author(s)

Johannes Gussenbauer, Alexander Kowarik, Statistics Austria

See Also

`bootstrap.rep`
`recalib`

Examples

```
# read in and prepare data
library(data.table)
dat <- data.table(read_sas("O:/B/3-AP/Analyse/sonstiges/
                            bundesländerschätzungen 2008-2018/daten/bldaten0816.sas7bdat"))

dat <- bootstrap.rep(dat,REP=20,hid="hid",weights="hgew",strata="bundesld",
                     year="jahr",totals=NULL,boot.names=NULL)
dat <- recalib(dat,hid="hid",weights="hgew",b.rep=paste0("w",1:20),
               year="jahr",conP.var=c("ksex","kausl","al","erw","pension"),
               conH.var=c("bundesld","hsize","recht"))

# or load file with calibrated bootstrap weights
# load("dat_calibweight.RData")

# estimate weightedRatio for povmd60 per year
err.est <- calc.stError(dat,weights="hgew",b.weights=paste0("w",1:20),year="jahr",var="povmd60",
                       fun="weightedRatio",cross_var=NULL,year.diff=NULL,year.mean=NULL)

# estimate weightedRatio for povmd60 per year and sex
cross_var <- "sex"
err.est <- calc.stError(dat,weights="hgew",b.weights=paste0("w",1:20),
                        year="jahr",var="povmd60",fun="weightedRatio",
                        cross_var=cross_var,year.diff=NULL,year.mean=NULL)

# use average over 3 years for standard error estimation
```

```
err.est <- calc.stError(dat,weights="hgew",b.weights=paste0("w",1:20),year="jahr",var="povmd60",
                        fun="weightedRatio",cross_var=cross_var,year.diff=NULL,year.mean=3)

# get estimate for difference of year 2016 and 2013
year.diff <- c("2016-2013")
err.est <- calc.stError(dat,weights="hgew",b.weights=paste0("w",1:20),year="jahr",var="povmd60",
                       fun="weightedRatio",cross_var=cross_var,year.diff=year.diff,year.mean=3)

# apply function to multiple variables and define different subsets
var <- c("povmd60","arose")
cross_var <- list("sex","bundesld",c("sex","bundesld"))
err.est <- calc.stError(dat,weights="hgew",b.weights=paste0("w",1:20),year="jahr",var=var,
                       fun="weightedRatio",cross_var=cross_var,year.diff=year.diff,year.mean=3)

# use a function from an other package that has sampling weights as its second argument
# for example ging() from laeken
library(laeken)

# set up help function that returns only the gini index
help_gini <- function(x,w){
 return(gini(x,w)$value)
}

err.est <- calc.stError(dat,weights="hgew",b.weights=paste0("w",1:20),year="jahr",var="epinc_real",
                       fun="help_gini",cross_var=cross_var,year.diff=year.diff,year.mean=3)
```

draw.bootstrap *Draw bootstrap replicates*

Description

Draw bootstrap replicates from survey data with rotating panel design. Survey information, like ID, sample weights, strata and population totals per strata, should be specified to ensure meaningfull survey bootstraping.

Usage

```
draw.bootstrap(dat,REP=1000,hid="hid",weights="hgew",strata="bundesld",
                        year="jahr",totals=NULL,boot.names=NULL)
```

Arguments

`dat`	either data.frame or data.table containing the survey data with rotating panel design.
`REP`	integer indicating the number of bootstrap replicates.
`hid`	character specifying the name of the column in `dat` containing the household ID.
`weights`	character specifying the name of the column in `dat` containing the sample weights.
`strata`	character vector specifying the name of the column in `dat` by which the population was stratified.
`year`	character specifying the name of the column in `dat` containing the sample years.

`totals`	(optional) character specifying the name of the column in `dat` containing the the totals per strata. If totals is `NULL`, the sum of weights per strata will be calcualted and named 'fpc'.
`boot.names`	character indicating the leading string of the column names for each bootstrap replica. If NULL defaults to "w".

Details

`draw.bootstrap` takes `dat` and draws `REP` bootstrap replicates from it. `dat` must be household data where household members correspond to multiple rows with the same household identifier. The data should at least containt the following columns:

- Column indicating the sample year;
- Column indicating the household ID;
- Column containing the household sample weights;
- Columns by which population was stratified during the sampling process.

A column for the totals in each strat can be included, but is only optional. If it is not included, e.g `totals=NULL`, this column will be calculated and added to `dat` using `strata` and `weights`.
The bootstrap replicates are drawn for each survey year (`year`) using the function `as.svrepdesign` from the package `survey`. Afterwards the bootstrap replicates for each household are carried forward from the first year the household enters the survey to all the censecutive years it stays in the survey.
This ensures that the bootstrap replicates follow the same logic as the sampled households, making the bootstrap replicates more comparable to the actual sample units.

Value

the survey data with the number of REP bootstrap replicates added as columns.

Returns a data.table containing the original data as well as the number of `REP` columns containing the bootstrap replicates for each repetition.
The columns of the bootstrap replicates are by default labeled "w*Number*" where *Number* goes from 1 to `REP`. If the column names of the bootstrap replicates should start with a different character or string the parameter `boot.names` can be used.

Author(s)

Johannes Gussenbauer, Alexander Kowarik, Statistics Austria

See Also

`data.table` for more information on data.table objects.
`svydesign` for more information on how to create survey-objects.
`as.svrepdesign` for more information on how bootstrap replicates are drawn from survey-objects.

Examples

```
# read in data (must be changed..)
dat <- data.table(read_sas("O:/B/3-AP/Analyse/sonstiges/
                                  bundesländerschätzungen 2008-2018/daten/bldaten0816.sas7bdat"))

# create 20 bootstrap replicates using the column "bundesld" as strata
dat_boot <- draw.bootstrap(dat=copy(dat),REP=20,hid="hid",weights="hgew",
                          strata="bundesld",year="jahr")

# do the same with more strata
dat_boot <- draw.bootstrap(dat=copy(dat),REP=20,hid="hid",weights="hgew",
                          strata=c("bundesld","sex","hsize"),year="jahr")

# change column names for bootstrap replicates
dat_boot <- draw.bootstrap(dat=copy(dat),REP=20,hid="hid",weights="hgew",
                          strata=c("bundesld"),year="jahr",boot.names="replicate")

# save bootstrap replicates as .RData
save(dat_boot,file="dat_replicates.RData")
# or .csv-file
write.csv2(dat_boot,file="dat_replicates.csv",row.names=FALSE)
```

`recalib` *Calibrate weights*

Description

Calibrate weights for bootstrap replicates by using iterative proportional updating to match population totals on various household and personal levels.

Usage

```
recalib(dat, hid = "hid", weights = "hgew", b.rep = paste0("w", 1:1000),
  year = "jahr", conP.var = c("ksex", "kausl", "al", "erw", "pension"),
  conH.var = c("bundesld", "hsize", "recht"), ...)
```

Arguments

`dat`	either data.frame or data.table containing the sample survey for various years.
`hid`	character specifying the name of the column in `dat` containing the household ID.
`weights`	character specifying the name of the column in `dat` containing the sample weights.
`b.rep`	character specifying the names of the columns in `dat` containing bootstrap weights which should be recalibratet
`year`	character specifying the name of the column in `dat` containing the sample years.
`conP.var`	character vector containig person-specific variables to which weights should be calibrated. for which contingency tables for the population tables are calculatet per `year` and
`conH.var`	character vector containig household-specific variables to which weights should be calibrated.
`...`	additional arguments passed on to function `ipu2` from the `simPop` package.

Details

`recalib` takes survey data (`dat`) containing the bootstrap replicates generated by `bootstrap.rep` and calibrates weights for each bootstrap replication according to population totals for person- or household-specific variables.
`dat` must be household data where household members correspond to multiple rows with the same household identifier. The data should at least containt the following columns:

- Column indicating the sample year;
- Column indicating the household ID;
- Column containing the household sample weights;
- Columns which contain the bootstrap replicates (see output of `bootstrap.rep`);
- Columns indicating person- or household-specific variables for which sample weight should be adjusted.

For each year and each variable in `conP.var` and/or `conH.var` contingency tables are estimated to get margin totals on personal- and/or household-specific variables in the population.
Afterwards the bootstrap replicates are multiplied with the original sample weight and the resulting product ist then adjusted using `ipu2` to match the previously calcualted contingency tables. In this process the columns of the bootstrap replicates are overwritten by the calibrated weights.

Value

Returns a data.table containing the survey data as well as the calibrated weights for the bootstrap replicates, which are labeled like the bootstrap replicates.

Author(s)

Johannes Gussenbauer, Alexander Kowarik, Statistics Austria

See Also

`ipu2` for more information on iterative proportional fitting.

Examples

```
# read in data (need to be changed)
library(data.table)
dat <- data.table(read_sas("O:/B/3-AP/Analyse/sonstiges/
                             bundesländerschätzungen 2008-2018/daten/bldaten0816.sas7bdat"))
# draw bootstrap replicates
dat <- bootstrap.rep(dat,REP=20,hid="hid",weights="hgew",
                     strata="bundesld",year="jahr",totals=NULL,boot.names=NULL)

# or load data with replicates if they have already been saved
# load("dat_replicates.RData")

# calibrate weight for bootstrap replicates
# use sex for person-specific and hsize for household-specific marginals
dat_calib <- recalib(dat=copy(dat),hid="hid",weights="hgew",b.weights=paste0("w",1:20),
                     year="jahr",conP.var=c("sex"),conH.var=c("hsize"))


# do the same but expand person- and household specific variables
```

```
dat_calib <- recalib(dat=copy(dat),hid="hid",weights="hgew",b.weights=paste0("w",1:20),
                     year="jahr",conP.var=c("sex","ageX"),conH.var=c("bundesld","hsize"))


# for many variables (household- or person-specific)
# use increase maxIter to get convergence
dat_calib <- recalib(dat=copy(dat),hid="hid",weights="hgew",b.weights=paste0("w",1:20),
                     year="jahr",conP.var=c("ksex","age","bildung","kausl","al","erw","pension"),
                     conH.var=c("bundesld","hsize","recht"),maxIter=100)

# save calibrated bootstrap weights as .RData
save(dat_calib,file="dat_calibweight.RData")
# or .csv-file
write.csv2(dat_calib,file="dat_calibweight.csv",row.names=FALSE)
```

Index