Package 'networkreporting'

January 27, 2015

Title Tools for using network reporting estimators.

Description networkreporting has a collection of functions useful for producing estimates from data that were collected using network reporting techniques like network scale-up, indirect sampling, network reporting, and sibling history.

Version 0.0.1

Maintainer Dennis Feehan <dfeehan@princeton.edu>

Author Dennis Feehan, Matthew Salganik

License MIT + file LICENSE

URL http://dfeehan.github.io/networkreporting

VignetteBuilder knitr

Imports functional, ggplot2, plyr, reshape2, stringr

Suggests knitr

Collate 'variance_estimators.r' 'helper_functions.r' 'indirect_sampling.r' 'known_population.r' 'scale_up.r' 'summation.r' 'networkreporting_package.r' 'rds.r'

NeedsCompilation no

Repository CRAN

Date/Publication 2014-05-06 07:09:23

R topics documented:

add.kp *attach known populations to a dataframe*

Description

take a known population vector (see df.to.kpvec) and associate it with a survey dataframe. this makes it more convenient to use some of the networkreporting package's function

Usage

```
add.kp(survey.data, kp.vec, total.pop.size = NULL)
```

Arguments

survey.data	the survey dataframe
kp.vec	the known population vector
total.pop.size	(optional) the total population size to use (see below)

Details

The total.popn.size parameter is interpreted as follows:

- NA if total.popn.size is NA then work with proportions
- NULL if total.popn.size is NULL (nothing passed in), then assume that there's a total.popn.size attribute associated with the dataset we're using
- numerical value if an actual total.popn.size was passed in, use that value

Value

the survey dataframe with the known population vector attached as an attribute

See Also

df.to.kpvec

Examples

```
## Not run:

  # if kp.dat is a dataframe with columns 'kp' with known popn names
  # and 'total.size' with the total size,
  # and my.survey is the dataframe with survey responses

  my.kp.vec <- df.to.kpvec(kp.data, kp.var='kp', kp.value='total.size')
  my.survey <- add.kp(my.survey, my.kp.vec)

  # now we can call estimator functions like
  # kp.degree.estimator without having to specify known
```

```
  # populations each time

## End(Not run)
```

`bootstrap.estimates` *bootstrap.estimates*

Description

this function contains the core of the rescaled bootstrap method for estimating uncertainty in our estimates it should be designed so that it can be passed in to estimation functions as an argument

Usage

```
bootstrap.estimates(survey.data, survey.design,
  bootstrap.fn, estimator.fn, num.reps, weights = NULL,
  ..., summary.fn = NULL, verbose = TRUE,
  parallel = FALSE, paropts = NULL)
```

Arguments

`survey.data`	the dataset to use
`survey.design`	a formula describing the design of the survey (see below - TODO)
`estimator.fn`	name of a function which, given a dataset like survey data and arguments in `...`, will produce an estimate of interest
`bootstrap.fn`	name of the method to be used to take bootstrap resamples; see below
`num.reps`	the number of bootstrap replication samples to draw
`weights`	weights to use in estimation (or NULL, if none)
`summary.fn`	(optional) name of a function which, given the set of estimates produced by estimator.fn, summarizes them. if not specified, all of the estimates are returned in a list
`parallel`	if TRUE, use the plyr library's .parallel argument to produce bootstrap resamples and estimates in parallel
`paropts`	if not NULL, additional arguments to pass along to the parallelization routine
`verbose`	if TRUE, produce lots of feedback about what is going on
`...`	additional arguments which will be passed on to the estimator fn

Value

if no summary.fn is specified, then return the list of estimates produced by estimator.fn; if summary.fn is specified, then return its output

chain.data *get a dataset from a chain*

Description

take the data for each member of the given chain and assemble it together in a dataset

Usage

```
chain.data(chain)
```

Arguments

chain	the chain to build a dataset from

Value

a dataset comprised of all of the chain's members' data put together. the order of the rows in the datset is not specified.

chain.size *get the size of a chain*

Description

count the total number of respondents in the chain and return it

Usage

```
chain.size(chain)
```

Arguments

chain	the chain object

Value

the number of respondents involved in the chain

chain.vals *chain.vals*

Description

get all of the values of the given variable found among members of a chain

Usage

```
chain.vals(chain, qoi.var = "uid")
```

Arguments

chain	the chain to get values from
qoi.var	the name of the variable to get from each member of the chain

Value

a vector with all of the values of qoi.var found in this chain. (currently, the order of the values in the vector is not guaranteed)

compare.mean.ties.truth *compare.mean.ties.truth*

Description

plot the relationship between the mean number of ties in the survey dataset and the true popn sizes

Usage

```
compare.mean.ties.truth(survey.data, weights = NULL,
  known.popns = NULL)
```

Arguments

survey.data	the dataframe with the survey results
known.popns	if not NULL, a vector whose entries are the size of the known populations, and whose names are the variable names in the dataset corresponding to each one. if NULL, then assume that the survey.data dataframe has an attribute called 'known.popns' containing this vector.
weights	if not NULL, weights to use in computing the estimate. this should be the name of the column in the survey.data which has the variable with the appropriate weights. these weights should be construted so that, eg, the mean of the degrees is estimated as (1/n) * \sum_i w_i * d_i

Value

a ggplot2 object with the relationship plot

`df.to.kpvec` *turn a dataframe into a known population vector*

Description

`df.to.kpvec` takes a dataframe which has a column with known population names, and a column with known population totals, and turns it into a known population vector. if the names of the survey variables corresponding to each known population are available, they can be passed in as well

Usage

```
df.to.kpvec(kp.data, kp.var, kp.value)
```

Arguments

`kp.data`	the known population dataset
`kp.var`	the column of `kp.data` that has known population names; either a column name, a column index, or a vector of values
`kp.value`	the column of `kp.data` that has known population sizes; either a column name, a column index, or a vector of value

Value

a vector whose entries have the known population values and whose names have the corresponding `kp.var` value

See Also

add.kp

Examples

```
## Not run:
  # if kp.dat is a dataframe with columns 'kp' with known popn names
  # and 'total.size' with the total size,
  # and my.survey is the dataframe with survey responses

  my.kp.vec <- df.to.kpvec(kp.data, kp.var='kp', kp.value='total.size')
  my.survey <- add.kp(my.survey, my.kp.vec)

  # now we can call estimator functions like
  # kp.degree.estimator without having to specify known
  # populations each time

## End(Not run)
```

`estimate.degree.distns` *estimate degree distributions by trait*

Description

break down RDS degree distributions by trait, and return an object which has the degrees for each trait as well as functions to draw degrees from each trait.

Usage

```
estimate.degree.distns(survey.data, d.hat.vals, traits,
  keep.vars = NULL)
```

Arguments

`survey.data`	the respondent info
`d.hat.vals`	the variable that contains the degrees for each respondent
`traits`	a vector of the names of the columns of `survey.data` which refer to the traits
`keep.vars`	additional vars to return along with degrees

Details

one of the items returned as a result is a function, `draw.degrees.fn`, which takes one argument, `traits`. this is a vector of traits and, for each entry in this vector, `draw.degress.fn` returns a draw from the empirical distribution of degrees among respondents with that trait. so, `draw.degrees.fn(c("0.0", "0.1", "0.1")` would return a degree drawn uniformly at random from among the observed degrees of respondents with trait "0.0" and then two degrees from respondents with trait "0.1"

Value

an object with

- `distns` a list with one entry per trait value; each
- `draw.degrees.fn` a function which gets called with one
- `keep.vars` the name of the other vars that are kept (if any)

`estimate.error` *given an estimated subpopn size or prevalence and the correct value, produce some measurements of how close the esimate is*

Description

given an estimated subpopn size or prevalence and the correct value, produce some measurements of how close the esimate is

Usage

```
estimate.error(estimate, truth)
```

Arguments

`estimate`	the estimate
`truth`	the correct answer

Value

a vector whose entries have various summaries of fit

`estimate.mixing` *construct a mixing model from GoC/RDS data*

Description

given a dataset with the respondents and a dataset on the parents (in many cases the same individuals), and a set of relevant traits, estimate mixing parameters and return a markov model

Usage

```
estimate.mixing(survey.data, parent.data, traits)
```

Arguments

`survey.data`	the respondent info
`parent.data`	the parent info
`traits`	the names of the traits to build the model on

Value

a list with two entries:

- `mixing.df` the data used to estimate the mixing
- `choose.next.state.fn` a function which can be passed a vector of states and will return a draw of a subsequent state each entry in the vector
- `mixing.df` a dataframe (long-form) representation of the transition counts used to estimate the transition probabilities
- `states` a list with an entry for each state. within each state's entry are
  - `trans.probs` a vector of estimated transition probabilities
  - `trans.fn` a function which, when called, randomly chooses a next state with probabilities given by the transition probs.

`example.survey` *Example household survey data*

Description

Example of a household survey dataset, used in unit tests and vignettes for the `networkreporting` package.

`gwsm.estimator` *indirect estimator (generalized weight share method / gwsm)*

Description

compute gwsm estimate of the population size

Usage

```
gwsm.estimator(survey.data, gwsm.col = "mult")
```

Arguments

`survey.data`	the dataframe with the survey results
`gwsm.col`	the name or index of the column that contains, for each respondent, the individual value of the number known divided by the sum of the multiplicities

Value

the multiplicity estimate of the hidden population's size (as a prevalence)

hm.q — *Names of ARD questions for known populations in example household survey data*

Description

A vector containing the names of the columns of which correspond to aggregate relational data (ARD) questions asked about groups of known size in `hhsurvey.data`

is.child.ct — *determine whether or not one id is a parent of another*

Description

this function allows us to determine which ids are directly descended from which other ones. it is the only part of the code that relies on the ID format used by the Curitiba study (Salganik et al 2011); by modifying this function, it shold be possible to adapt this code to another study

Usage

```
is.child.ct(id, seed.id)
```

Arguments

`id`	the id of the potential child
`seed.id`	the id of the potential parent

Value

TRUE if `id` is the direct descendant of `seed.id` and FALSE otherwise

knownpop.dat — *Size of example known populations*

Description

A dataframe with the size of the known populations corresponding to the survey data in `hhsurvey.data`

kp.degree.estimator *kp.degree.estimator*

Description

compute an estimate of the respondents' degrees using the known population method

Usage

```
kp.degree.estimator(survey.data, known.popns = NULL,
  total.popn.size = NULL, missing = "ignore",
  verbose = FALSE)
```

Arguments

survey.data	the dataframe with the survey results
known.popns	if not NULL, a vector whose entries are the size of the known populations, and whose names are the variable names in the dataset corresponding to each one. if NULL, then assume that the survey.data dataframe has an attribute called 'known.popns' containing this vector.
total.popn.size	the size of the entire population. if NULL, this function returns proportions; if not NULL, it returns absolute numbers (ie, the proportions * total popn size)
missing	if "ignore", then proceed with the analysis without doing anything about missing values. if "complete.obs" then, for each row, use only the known populations that have no missingness for the computations. care must be taken in using this second option
verbose	if TRUE, print messages to the screen

Details

note that this function does not take survey weights, since these estimates are not for total degree, but just for the individual degree of each respondent

Value

a vector with an estimate of the degree for each row in survey.data. if missing=="ignore", then the degree for rows that have missingness in the 'how many X' questions will be set to NA

Examples

```
data(hhsurvey)
 kp.vec <- df.to.kpvec(knownpop.dat, kp.var='known.popn', kp.value='size')
 example.survey <- add.kp(example.survey, kp.vec)
 d.hat <- kp.degree.estimator(example.survey,
                              missing="complete.obs",
                              total.popn.size=NA)
```

make.chain *build an RDS seed's chain from the dataset*

Description

note that this assumes that the chain is a tree (no loops)

Usage

```
make.chain(seed.id, survey.data,
  is.child.fn = is.child.ct)
```

Arguments

seed.id	the id of the seed whose chain we wish to build from the dataset
survey.data	the dataset
is.child.fn	a function which takes two ids as arguments; it is expected to return TRUE if the second argument is the parent of the first, and FALSE otherwise. it defaults to is.child.ct

Value

info

max.depth *get the height (maximum depth) of a chain*

Description

get the height (maximum depth) of a chain

Usage

```
max.depth(chain)
```

Arguments

chain	the chain object

Value

the maximum depth of the chain

`mc.sim` *run a markov model*

Description

run a given markov model for n time steps, starting at a specified state

Usage

```
mc.sim(mm, start, n)
```

Arguments

`mm`	the markov model object returned by `estimate.mixing`
`start`	the name of the state to start in
`n`	the number of time-steps to run through

Details

this uses the markov model produced by estimate.mixing

Value

a vector with the state visited at each time step. the first entry has the starting state

`MU284` *MU284 population*

Description

Data used in unit tests for variance estimation. See TODO-Sarndal TODO-sampling package TODO-doc describing unit tests

`MU284.boot.res.summ` *MU284 bootstrap summary results*

Description

summary of the MU284 bootstrap resamples, used in unit tests

`MU284.estimator.fn` *function used in unit tests for MU284 bootstrap resamples*

Description

this function is used as part of the unit tests for the bootstrap resampling code

`MU284.surveys` *MU284 bootstrap results*

Description

list of datasets, each with one bootstrap sample of the MU284 population; this is used in unit tests

`multiplicity.estimator` *multiplicity.estimator*

Description

compute multiplicity estimate of the population size

Usage

```
multiplicity.estimator(survey.data, mult.col = "mult")
```

Arguments

`survey.data`	the dataframe with the survey results
`mult.col`	the name or index of the column that contains, for each respondent, the individual value of the number known divided by the sum of the multiplicities

Value

the multiplicity estimate of the hidden population's size (as a prevalence)

`networkreporting` *Network reporting estimators*

Description

`networkreporting` has methods for analyzing data that were collected using network reporting techniques. It includes estimators appropriate for indirect sampling, network scale-up, network reporting, and sibling history methods.

nsum.estimator *nsum.estimator*

Description

compute network scale-up (nsum) estimate of the hidden population's size. if the degree ratio and information transmission rate are both 1 (the defaults), this is the Killworth estimator (Killworth et al 1998, "A social network approach...")

Usage

```
nsum.estimator(survey.data, d.hat.vals = "d",
  y.vals = "y", total.popn.size = NULL, deg.ratio = 1,
  tx.rate = 1, weights = NULL, killworth.se = FALSE,
  missing = "ignore", verbose = FALSE, ...)
```

Arguments

survey.data	the dataframe with survey results
d.hat.vals	the name or index of the column that contains each respondent's estimated degree
y.vals	the name or index of the column that contains the count of hidden popn members known
total.popn.size	NULL, NA, or a size
weights	if not NULL, weights to use in computing the estimate. this should be the name of the column in the survey.data which has the variable with the appropriate weights. these weights should be construted so that, eg, the mean of the degrees is estimated as (1/n) * \sum_i w_i * d_i
deg.ratio	the degree ratio, \frac\bard_T\bard; defaults to 1
tx.rate	the information transmission rate; defaults to 1
killworth.se	if not NA, return the Killworth et al estimate of
missing	if "ignore", then proceed with the analysis without doing anything about missing values. if "complete.obs" then only use rows that have no missingness for the computations (listwise deletion). care must be taken in using this second option
verbose	if TRUE, print messages to the screen
...	extra parameters to pass on to the bootstrap fn, if applicable

Value

the nsum estimate of the hidden population's size (as a prevalence or an absolute number, depending on total.popn.size)

Examples

```
data(hhsurvey)
 kp.vec <- df.to.kpvec(knownpop.dat, kp.var='known.popn', kp.value='size')
 example.survey <- add.kp(example.survey, kp.vec)
 d.hat <- kp.degree.estimator(example.survey,
                              missing="complete.obs",
                              total.popn.size=10e6)
 example.survey$d.hat <- d.hat
 sw.estimate <- nsum.estimator(example.survey,
                               d.hat.vals="d.hat",
                               y.vals="sex.workers",
                               weights="indweight",
                               missing="complete.obs",
                               total.popn.size=10e6)
```

`nsum.internal.validation` *nsum.internal.validation*

Description

use a hold-one-out method to estimate the predictive accuracy of the network scale-up estimator on the known populations

Usage

```
nsum.internal.validation(survey.data, known.popns = NULL,
  total.popn.size = NULL, degrees = NULL,
  missing = "ignore", kp.method = FALSE, weights = NULL,
  killworth.se = FALSE, return.plot = FALSE,
  verbose = FALSE, bootstrap = FALSE, ...)
```

Arguments

`survey.data`	the dataframe with the survey results
`known.popns`	if not NULL, a vector whose entries are the size of the known populations, and whose names are the variable names in the dataset corresponding to each one. if NULL, then assume that the survey.data dataframe has an attribute called 'known.popns' containing this vector.
`total.popn.size`	the size of the entire population. if NA, this function works with proportions; if NULL, it looks for the 'total.popn.size' attribute of the dataset `survey.data`; if not NULL or NA, it works with absolute numbers (ie, the proportions * total popn size)
`degrees`	if not NULL, then the name or index of the column in the datset containing the degree estimates. if NULL, then use the known population method to estimate the degrees (see `kp.degree.estimator`)

missing	if "ignore", then proceed with the analysis without doing anything about missing values. if "complete.obs" then only use rows that have no missingness for the computations (listwise deletion). care must be taken in using this second option
kp.method	if TRUE, then we're using known population method estimates of the degrees. this means we have to recompute the degrees each time we hold out a known subgroup. if the degrees come from another estimator, like the summation method, then we don't need to do that since we don't use the ARD questions in coming up with the degree estimate.
weights	if not NULL, weights to use in computing the estimate. this should be the name of the column in the survey.data which has the variable with the appropriate weights. these weights should be construted so that, eg, the mean of the degrees is estimated as (1/n) * \sum_i w_i * d_i
killworth.se	if TRUE, return the Killworth et al estimate
return.plot	if TRUE, make and return a ggplot2 plot object
verbose	if TRUE, report more detailed information about what's going on
bootstrap	if TRUE, use bootstrap.estimates to take bootstrap resamples in order to obtain intervals around each estimate. in this case, you are expected to also pass in at least bootstrap.fn, survey.design, and num.reps
...	additional arguments, which are passed on to bootstrap.estimates if bootstrap is TRUE

Details

given a set of estimated degrees, responses to a group of ARD questions, and the total size of the populations that the ARD questions ask about, this function estimates the accuracy of the network scale-up method by dropping each known population in turn, using the non-dropped populations to compute the degree and an estimate of the size of the known population, and comparing the result to the actual size of the known population

Value

a list with a dataset containing the subpopn-specific estimates, as well as several summaries of the accuracy of those estimates, including mae (mean absolute error), mse (mean squared error), rmse (root mean squared error), and are (average relative error)

---

parent.data *Example game of contacts parents (nominators)*

Description

Example of a game of contacts dataset, collected via RDS, organized from the perspective of the parents (ie, those who nominated subsequent respondents). See also survey.data. Used in unit tests and vignettes for the networkreporting package.

rds.boot.draw.chain *draw RDS bootstrap resamples for one chain*

Description

this function uses the algorithm described in the supporting online material for Weir et al 2012 (Weir et al 2012, "A comparison of respondent-driven...") to take bootstrap resamples of one chain from an RDS dataset

Usage

```
rds.boot.draw.chain(chain, mm, dd, parent.trait,
  idvar = "uid")
```

Arguments

chain	the chain to draw resamples for
mm	the mixing model to use
dd	the degree distns to use
parent.trait	a vector whose length is the number of bootstrap reps we want
idvar	the name of the variable used to label the columns of the output (presumably some id identifying the row in the original dataset they come from – see below)

Value

a list of dataframes with one entry for each respondent in the chain. each dataframe has one row for each bootstrap replicate. so if we take 10 bootstrap resamples of a chain of length 50, there will be 50 entries in the list that is returned. each entry will be a dataframe with 10 rows.

rds.chain.boot.draws *draw RDS bootstrap resamples*

Description

draw boostrap resamples for an RDS dataset, using the algorithm described in the supporting online material of (Weir et al 2012, "A comparison of respondent-driven...")

Usage

```
rds.chain.boot.draws(chains, mm, dd, num.reps,
  keep.vars = NULL)
```

Arguments

chains	a list whose entries are the chains we want to resample
mm	the mixing model
dd	the degree distributions
num.reps	the number of bootstrap resamples we want
keep.vars	if not NULL, then the names of variables from the original dataset we want appended to each bootstrap resampled dataset (default is NULL)

Value

a list of length num.reps; each entry in the list has one bootstrap-resampled dataset

---

rds.mc.boot.draws *draw RDS bootstrap resamples using the algorithm in Salganik 2006, "Variance estimation..."*

Description

this algorithm picks a respondent from the survey to be a seed uniformly at random. it then generates a bootstrap draw by simulating the markov process forward for n steps, where n is the size of the draw required.

Usage

```
rds.mc.boot.draws(chains, mm, dd, num.reps)
```

Arguments

chains	a list with the chains constructed from the survey using make.chain
mm	the mixing model
dd	the degree distributions
num.reps	the number of bootstrap resamples we want

Details

if you wish the bootstrap dataset to end up with variables from the original dataset other than the traits and degree, then you must specify this when you construct dd using the 'estimate.degree.distns function.

Value

a list of length num.reps; each entry in the list has one bootstrap-resampled dataset

rdsII.estimator *rdsII.estimator*

Description

compute an estimate for the prevalence of a trait from an RDS sample, using the estimator described in Volz and Heckathorn 2008

Usage

```
rdsII.estimator(survey.data, d.hat.vals, y.vals,
  missing = "ignore", verbose = FALSE)
```

Arguments

survey.data	the dataframe with RDS survey results
d.hat.vals	the name or index of the column that contains each respondent's estimated degree
y.vals	the name or index of the column that contains the quantity of interest. if this is a dichotomous trait, it should be 0 / 1
missing	if "ignore", then proceed with the analysis without doing anything about missing values. if "complete.obs" then only use rows that have no missingness for the computations (listwise deletion). care must be taken in using this second option
verbose	if TRUE, print messages to the screen

Details

NOTE: we have no weights for now, right? RDS doesn't get used with weights?

Value

the RDS-II estimate of the average of the quantity of interest

rescaled.bootstrap.sample *rescaled.bootstrap.sample*

Description

given a survey dataset and a description of the survey design (ie, which combination of vars determines primary sampling units, and which combination of vars determines strata), take a bunch of bootstrap samples for the rescaled bootstrap estimator (see, eg, Rust and Rao 1996).

Usage

```
rescaled.bootstrap.sample(survey.data, survey.design,
  parallel = FALSE, paropts = NULL, num.reps = 1)
```

Arguments

survey.data	the dataset to use
survey.design	a formula describing the design of the survey (see below)
num.reps	the number of bootstrap replication samples to draw
parallel	if TRUE, use parallelization (via plyr)
paropts	an optional list of arguments passed on to plyr to control details of parallelization

Details

Note that we assume that the formula uniquely specifies PSUs. This will always be true if the PSUs were selected without replacement. If they were selected with replacement, then it will be necessary to make each realization of a given PSU in the sample a unique id. Bottom line: the code below assumes that all observations within each PSU (as identified by the design formula) are from the same draw of the PSU.

The rescaled bootstrap technique works by adjusting the estimation weights based on the number of times each row is included in the resamples. If a row is never selected, it is still included in the returned results, but its weight will be set to 0. It is therefore important to use estimators that make use of the estimation weights on the resampled datasets.

We always take m_i = n_i - 1, according to the advice presented in Rao and Wu (1988) and Rust and Rao (1996).

survey.design is a formula of the form
weight ~ psu_vars + strata(strata_vars), where weight is the variable with the survey weights and psu is the variable denoting the primary sampling unit

Value

a list with num.reps entries. each entry is a dataset which has at least the variables index (the row index of the original dataset that was resampled) and weight.scale (the factor by which to multiply the sampling weights in the original dataset).

---

srs.bootstrap.sample *srs.bootstrap.sample*

Description

given a survey dataset and a description of the survey design (ie, which combination of vars determines primary sampling units, and which combination of vars determines strata), take a bunch of bootstrap samples under a simple random sampling (with repetition) scheme

Usage

```
srs.bootstrap.sample(survey.data, num.reps = 1,
  parallel = FALSE, paropts = NULL, ...)
```

Arguments

survey.data	the dataset to use
num.reps	the number of bootstrap replication samples to draw
parallel	if TRUE, use parallelization (via plyr)
paropts	an optional list of arguments passed on to plyr to control details of parallelization
...	ignored, but useful because it allows params like which are used in other bootstrap designs, to be passed in without error

Details

survey.design is not needed; it's included as an argument to make it easier to drop srs.bootstrap.sample into the place of other bootstrap functions, which do require information about the survey design

Value

a list with num.reps entries. each entry is a dataset which has at least the variables index (the row index of the original dataset that was resampled) and weight.scale (the factor by which to multiply the sampling weights in the original dataset).

---

summation.degree.estimator

summation.degree.estimator

Description

compute an estimate of the respondents' degrees using the summation method (McCarty et al 2001)

Usage

```
summation.degree.estimator(survey.data, sum.q = NULL,
  missing = "ignore")
```

Arguments

survey.data	the dataframe with the survey results
sum.q	if not NULL, a vector whose entries are the variable names in the dataset corresponding to each summation question. if NULL, then assume that the survey.data dataframe has an attribute called 'sum.qs' containing this vector.
missing	if "ignore", then proceed with the analysis without doing anything about missing values. other options are not yet implemented.

Details

This is the method first described in McCarty et al 2001, "Comparing two methods for estimating network size"

Note that the summation degree estimator for the case where there is missing data is not yet implemented. (In fact, I don't think that there is a known estimator for this case.)

Value

a vector with an estimate of the degree for each row in survey.data. if na.rm=TRUE, then the degree for rows that have missingness in the summation questions will be set to NA

---

survey.data *Example game of contacts survey data*

Description

Example of a game of contacts survey dataset, collected via RDS. See also `parent.data`. Used in unit tests and vignettes for the `networkreporting` package.

---

target.q *Names of ARD questions for hidden populations in example household survey data*

Description

A vector containing the names of the columns of which correspond to aggregate relational data (ARD) questions asked about hidden groups (ie, groups of unknown size) in `hhsurvey.data`

---

tn1 *Toy network example number 1*

Description

Toy network example, useful for unit tests in the `networkreporting` package

---

tn2 *Toy network example number 2*

Description

Toy network example, useful for unit tests in the `networkreporting` package

tn3 *Toy network example number 3*

Description

Toy network example, useful for unit tests in the networkreporting package

tn4 *Toy network example number 4*

Description

Toy network example, useful for unit tests in the networkreporting package

tnr1 *Toy network with attributes, example number 1*

Description

Toy network example, useful for unit tests in the networkreporting package

tnr2 *Toy network with attributes, example number 2*

Description

Toy network example, useful for unit tests in the networkreporting package

tnr3 *Toy network with attributes, example number 3*

Description

Toy network example, useful for unit tests in the networkreporting package

topcode.data *topcode a group of variables*

Description

this function uses `topcode.var` to topcode a set of variables. it's useful for topcoding a whole set of aggregated relational data ("how many X are you connected to?") questions in the same way.

Usage

```
topcode.data(survey.data, vars, max, to.na = NULL,
  ignore = NA)
```

Arguments

`survey.data`	the dataset with the survey responses
`vars`	a vector with the names or indices of the columns in the dataframe that are to be topcoded
`max`	the maximum value; all values > max are recoded to max
`ignore`	a vector of values to leave unchanged
`to.na`	a vector of values to recode to NA (this happens before topcoding)

Value

the topcoded vector

Examples

```
## Not run:
   data(hh.survey) # example data included with the package
   example.survey <- topcode.data(example.survey,
                                  vars=known.popn.vars,
                                  max=30)

## End(Not run)
```

topcode.var *topcode a vector of numerical values*

Description

this function topcodes one vector; it's used by the `topcode` function to topcode a set of columns in a data frame

Usage

```
topcode.var(x, max, to.na = NULL, ignore = NA)
```

Arguments

x	the vector of values to topcode
max	the maximum value; all values > max are recoded to max
to.na	a vector of values to recode to NA (this happens before topcoding)
ignore	a vector of values to leave unchanged

Value

the topcoded vector

---

tot.pop.size *Size of example total population*

Description

A numeric with the size of the total population to be used with the survey data in `hhsurvey.data`

---

total.degree.estimator *total.degree*

Description

Estimate the total degree of the population network from sample degrees

Usage

```
total.degree.estimator(survey.data, d.hat.vals = "d",
  weights = NULL, missing = "ignore")
```

Arguments

survey.data	the dataframe with survey results
d.hat.vals	the name or index of the column that contains each respondent's estimated degree
weights	if not NULL, weights to use in computing the estimate. this should be the name of the column in the survey.data which has the variable with the appropriate weights. these weights should be construted so that, eg, the mean of the degrees is estimated as (1/n) * \sum_i w_i * d_i
missing	if "ignore", then proceed with the analysis without doing anything about missing values. if "complete.obs" then only use rows that have no missingness for the computations (listwise deletion). care must be taken in using this second option

Details

This computes the weighted sum of the respondents' estimated degrees. For now, this function doesn't worry about missing values OR about differences between the frame and the universe.

Value

the estimated total degree

Examples

```
data(hhsurvey)
 kp.vec <- df.to.kpvec(knownpop.dat, kp.var='known.popn', kp.value='size')
 example.survey <- add.kp(example.survey, kp.vec)
 d.hat <- kp.degree.estimator(example.survey,
                              missing="complete.obs",
                              total.popn.size=NA)
 example.survey$d.hat <- d.hat
 tot.d.hat <- total.degree.estimator(example.survey,
                                     d.hat.vals="d.hat",
                                     weights="indweight",
                                     missing="complete.obs")
```

---

toy.networks *Toy network data*

Description

List with 4 toy network examples, useful for unit tests in the networkreporting package

---

toy.nr.networks *Toy network response networks*

Description

List with 3 toy network examples with attributes attached to each node; useful for unit tests in the networkreporting package.

Index