Package ‘AuxSurvey’

July 21, 2025

Title Survey Analysis with Auxiliary Discretized Variables

Version 1.0

Author Jungang Zou [aut, cre],
Yutao Liu [aut],
Sharifa Williams [aut],
Qixuan Chen [aut]

Maintainer Jungang Zou <`jungang.zou@gmail.com`>

Description
Probability surveys often use auxiliary continuous data from administrative records, but the utility of this data is diminished when it is discretized for confidentiality. We provide a set of survey estimators to make full use of information from the discretized variables. See Williams, S.Z., Zou, J., Liu, Y., Si, Y., Galea, S. and Chen, Q. (2024), Improving Survey Inference Using Administrative Records Without Releasing Individual-Level Continuous Data. Statistics in Medicine, 43: 5803-5813. <doi:10.1002/sim.10270> for details.

Encoding UTF-8

RoxygenNote 7.3.2

Depends mgcv, rstanarm, stats

Imports survey, gtools, coda, BART, dplyr, stringr, gridExtra, rlang

License Apache License (>= 2)

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2024-12-20 20:50:07 UTC

Contents

auxsurvey *Auxiliary Variables in Survey Analysis*

Description

This function provides a user-friendly interface for various estimators in survey analysis when working with discretized auxiliary variables. Probability surveys often use continuous data from administrative records as auxiliary variables, but the utility of this data is diminished when discretized for confidentiality purposes. This package offers different estimators that handle discretized auxiliary variables effectively.

Usage

```
auxsurvey(
  formula,
  auxiliary = NULL,
  samples,
  population = NULL,
  subset = NULL,
  family = gaussian(),
 method = c("sample_mean", "rake", "postStratify", "MRP", "GAMP", "linear", "BART"),
  weights = NULL,
  levels = c(0.95, 0.8, 0.5),
  stan_verbose = TRUE,
  show_plot = TRUE,
  nskip = 1000,
  npost = 1000,
  nchain = 4,
  HPD_interval = FALSE,
  seed = NULL
)
```

Arguments

`formula`	A string or formula specifying the outcome model. For non-model-based methods (e.g., sample mean, raking, post-stratification), only include the outcome variable (e.g., "~Y"). For model-based methods (e.g., MRP, GAMP, linear regression), additional fixed effect predictors can be specified, such as "Y ~ X1 + X2 + I(X^2)". For GAMP, smooth functions can be specified as "Y ~ X1 + s(X2, 10) + s(X3, by = X1)". Categorical variables are automatically treated as dummy variables in model-based methods.
`auxiliary`	A string specifying the formula for the auxiliary variables. For sample mean and BART, this should be `NULL`. For raking, post-stratification, and GAMP, this should be an additive model (e.g., "Z1 + Z2 + Z3"). For MRP, specify random effects for terms in this parameter, such as "Z1 + Z2 + Z3" or "Z1 + Z2:Z3".

`samples`	A dataframe or tibble containing all variables specified in `formula` and `auxiliary`. This is typically a subset of the population.
`population`	A dataframe or tibble containing all variables specified in `formula` and `auxiliary`. This is the entire population used for estimation.
`subset`	A character vector representing filtering conditions to select subsets of `samples` and `population`. Default is `NULL`, in which case the analysis is performed on the entire dataset. If subsets are specified, estimates for both the whole data and the subsets will be calculated.
`family`	The distribution family of the outcome variable. Supported options are: `gaussian` for continuous outcomes and `binomial` for binary outcomes.
`method`	A string specifying the model to use. Options include "sample_mean", "rake", "postStratify", "MRP", "GAMP", "linear", and "BART".
`weights`	A numeric vector of case weights. The length should match the number of cases in `samples`.
`levels`	A numeric vector specifying the confidence levels for the confidence intervals (CIs). Multiple values can be specified to calculate multiple CIs.
`stan_verbose`	A logical scalar; if `TRUE`, prints all messages when running Stan models. Default is `FALSE`. This parameter only applies to Bayesian models.
`show_plot`	A logical scalar; if `TRUE`, shows diagnostic plots for Stan models. Default is `FALSE`. This parameter only applies to Bayesian models.
`nskip`	An integer specifying the number of burn-in iterations for each chain in MCMC for Stan models. Default is `1000`. This parameter only applies to Bayesian models.
`npost`	An integer specifying the number of posterior sampling iterations for each chain in MCMC for Stan models. Default is `1000`. This parameter only applies to Bayesian models.
`nchain`	An integer specifying the number of MCMC chains for Stan models. Default is `4`. This parameter only applies to Bayesian models.
`HPD_interval`	A logical scalar; if `TRUE`, calculates the highest posterior density (HPD) intervals for the CIs of Stan models. Default is `FALSE`, in which case symmetric intervals are calculated. This parameter only applies to Bayesian models.
`seed`	An integer specifying the random seed for reproducibility. Default is `NULL`.

Details

The available estimators include:

- Weighted or unweighted sample mean
- Weighted or unweighted raking
- Weighted or unweighted post-stratification
- Bayesian methods:
  - BART (Bayesian Additive Regression Trees)
  - MRP (Multilevel Regression with Poststratification)
  - GAMP (Generalized Additive Model of Response Propensity)
  - Weighted linear regression

These Bayesian models are implemented using the **rstan** and **rstanarm** packages.

Value

A list containing the sample mean estimates and CIs for the subset and/or the whole dataset. Each element in the list includes: - `estimate`: The point estimate of the sample mean. - `CI`: Confidence intervals for the sample mean. - Other elements for each confidence level specified in `levels`.

Examples

```
## Simulate data with nonlinear association (setting 3).
data = simulate(N = 3000, discretize = 10, setting = 3, seed = 123)
population = data$population
samples = data$samples
ipw = 1 / samples$true_pi
true_mean = mean(population$Y1)

## IPW Sample Mean
IPW_sample_mean = auxsurvey("~Y1", auxiliary = NULL, weights = ipw,
                            samples = samples, population = population,
                            subset = c("Z1 == 1 & Z2 == 1"), method = "sample_mean",
                            levels = 0.95)

## Raking
rake = auxsurvey("~Y1", auxiliary = "Z1 + Z2 + Z3 + auX_10", samples = samples,
                 population = population, subset = c("Z1 == 1", "Z1 == 1 & Z2 == 1"),
                 method = "rake", levels = 0.95)

## MRP
MRP = auxsurvey("Y1 ~ 1 + Z1", auxiliary = "Z2 + Z3:auX_10", samples = samples,
                population = population, subset = c("Z1 == 1", "Z1 == 1 & Z2 == 1"),
                method = "MRP", levels = 0.95, nskip = 4000, npost = 4000,
                nchain = 1, stan_verbose = FALSE, HPD_interval = TRUE)

## GAMP
GAMP = auxsurvey("Y1 ~ 1 + Z1 + Z2 + Z3", auxiliary = "s(auX_10) + s(logit_true_pi, by = Z1)",
                 samples = samples, population = population, method = "GAMP",
                 levels = 0.95, nskip = 4000, npost = 4000, nchain = 1,
                 stan_verbose = FALSE, HPD_interval = TRUE)

## BART
BART = auxsurvey("Y1 ~ Z1 + Z2 + Z3 + auX_10", auxiliary = NULL, samples = samples,
                 population = population, method = "BART", levels = 0.95,
                 nskip = 4000, npost = 4000, nchain = 1, HPD_interval = TRUE)
```

---

postStr_wt *Weighted or Unweighted Post-Stratification Estimator*

Description

This function performs post-stratification adjustment for survey data, which adjusts the sample weights to match the marginal distributions of auxiliary variables in the population. It supports both weighted and unweighted estimations for various outcome variables, including Gaussian (continuous) and Binomial (binary) outcomes. The function computes estimates and confidence intervals (CIs) for the outcome variable using post-stratification based on the specified auxiliary variables.

Usage

```
postStr_wt(
  svysmpl,
  svypopu,
  auxVars,
  svyVar,
  subset = NULL,
  family = gaussian(),
  invlvls,
  weights = NULL
)
```

Arguments

`svysmpl`	A dataframe or tibble representing the sample data (`samples`). This should contain the outcome variable and any auxiliary variables.
`svypopu`	A dataframe or tibble representing the population data (`population`). This is used to compute the finite population correction (FPC) for post-stratification.
`auxVars`	A character vector containing the names of auxiliary variables to be used for post-stratification. These variables will be used to adjust the weights.
`svyVar`	The outcome variable for which the post-stratification estimate is calculated.
`subset`	A character vector representing filtering conditions to select subsets of the sample and population. Default is `NULL`, in which case the analysis is performed on the entire dataset. If subsets are specified, estimates for both the whole data and the subsets will also be calculated.
`family`	The distribution family of the outcome variable. Supported options are: `gaussian` for continuous outcomes and `binomial` for binary outcomes.
`invlvls`	A numeric vector specifying the confidence levels for the post-stratification estimators. If more than one value is provided, multiple CIs will be calculated.
`weights`	A numeric vector of case weights. The length should match the number of cases in `svysmpl`. These weights are used in the weighted post-stratification adjustment.

Value

A list where each element contains the post-stratification estimate and confidence intervals (CIs) for a subset or the entire dataset. The list includes: - `est`: The post-stratification estimate for the outcome variable. - `se`: The standard error of the estimate. - `tCI`: The confidence intervals for the estimate. - `sample_size`: The sample size for the subset or entire dataset. - `population_size`: The population size, if provided, including the finite population correction (FPC).

Examples

```
## Simulate data with nonlinear association (setting 3).
data = simulate(N = 3000, discretize = 3, setting = 3, seed = 123)
population = data$population  # Population data (3000 cases)
samples = data$samples        # Sample data (600 cases)
ipw = 1 / samples$true_pi    # Compute inverse probability weights

## Perform weighted post-stratification with auxiliary variables
auxVars = c("Z1", "Z2", "Z3")
Weighted_postStratify = postStr_wt(svysmpl = samples, svypopu = population, auxVars = auxVars,
                                   svyVar = "Y1", subset = NULL, family = gaussian(),
                                   invlvls = c(0.95), weights = ipw)
Weighted_postStratify

## Perform unweighted post-stratification
Unweighted_postStratify = postStr_wt(svysmpl = samples, svypopu = population, auxVars = auxVars,
                                     svyVar = "Y1", subset = NULL, family = gaussian(),
                                     invlvls = c(0.95), weights = NULL)
Unweighted_postStratify
```

---

rake_wt *Weighted or Unweighted Raking Estimator*

Description

This function estimates the weighted or unweighted raking adjustment for survey data. Raking adjusts the sample weights to match the marginal distributions of auxiliary variables in the population. It supports both weighted and unweighted estimations for a variety of outcome variables, including Gaussian (continuous) and Binomial (binary) outcomes.

Usage

```
rake_wt(
  svysmpl,
  svypopu,
  auxVars,
  svyVar,
  subset = NULL,
  family = gaussian(),
  invlvls,
  weights = NULL,
  maxiter = 50
)
```

Arguments

`svysmpl`	A dataframe or tibble representing the sample data (`samples`). This should contain the outcome variable and any auxiliary variables.
`svypopu`	A dataframe or tibble representing the population data (`population`). This is used to compute the finite population correction (FPC) for raking.
`auxVars`	A character vector containing the names of auxiliary variables to be used for raking. These variables will be used to adjust the weights.
`svyVar`	The outcome variable for which the raking estimate is calculated.
`subset`	A character vector representing filtering conditions to select subsets of the sample and population. Default is `NULL`, in which case the analysis is performed on the entire dataset. If subsets are specified, estimates for both the whole data and the subsets will be calculated.
`family`	The distribution family of the outcome variable. Supported options are: `gaussian` for continuous outcomes and `binomial` for binary outcomes.
`invlvls`	A numeric vector specifying the confidence levels for the raking estimators. If more than one value is provided, multiple CIs will be calculated.
`weights`	A numeric vector of case weights. The length should match the number of cases in `svysmpl`. These weights are used in the weighted raking adjustment.
`maxiter`	An integer specifying the maximum number of iterations for the raking algorithm. Default is 50.

Value

A list where each element contains the raking estimate and confidence intervals (CIs) for a subset or the entire dataset. The list includes: - `est`: The raking estimate for the outcome variable. - `se`: The standard error of the estimate. - `tCI`: Confidence intervals for the estimate. - `sample_size`: The sample size for the subset or entire dataset. - `population_size`: The population size, if provided, including the finite population correction (FPC).

Examples

```
## Simulate data with nonlinear association (setting 3).
data = simulate(N = 3000, discretize = 3, setting = 3, seed = 123)
population = data$population  # Population data (3000 cases)
samples = data$samples        # Sample data (600 cases)
ipw = 1 / samples$true_pi    # Compute inverse probability weights

## Perform weighted raking with auxiliary variables
auxVars = c("Z1", "Z2", "Z3")
Weighted_rake = rake_wt(svysmpl = samples, svypopu = population, auxVars = auxVars,
                        svyVar = "Y1", subset = NULL, family = gaussian(),
                        invlvls = c(0.95), weights = ipw, maxiter = 50)
Weighted_rake

## Perform unweighted raking
Unweighted_rake = rake_wt(svysmpl = samples, svypopu = population, auxVars = auxVars,
                          svyVar = "Y1", subset = NULL, family = gaussian(),
                        invlvls = c(0.95), weights = NULL, maxiter = 50)
Unweighted_rake
```

`simulate` *Simulate Survey Data with Discretized Auxiliary Variables*

Description

This function simulates survey data with discretized auxiliary variables. It generates a population dataset with continuous and binary outcomes, and includes auxiliary variables that are discretized into multiple categories. The function also generates a subset of the population as a sample, based on the propensity scores.

Usage

```
simulate(N = 3000, discretize = c(3, 5, 10), setting = c(1, 2, 3), seed = NULL)
```

Arguments

`N`	Number of population units to simulate. Default is 3000.
`discretize`	A scale specifying the number of categories for discretizing continuous variables. The function discretizes both `X` and `W` into the specified categories. Default is a number among `(3, 5, 10)`.
`setting`	A numeric value to specify the simulation setting. The settings define different relationships between the outcome variables and the covariates. Possible values are 1, 2, 3, and 4. Default is a number among `c(1, 2, 3)`.
`seed`	An optional random seed for reproducibility. Default is `NULL`.

Details

The function supports multiple simulation settings, where each setting modifies the relationships between the outcome variables and the covariates.

Value

A list containing two elements:

- `population`: A tibble with the simulated population data, including both continuous and binary outcomes, as well as auxiliary variables (both raw and discretized).
- `samples`: A tibble with the simulated sample data, where individuals are included based on their estimated propensity scores.

Examples

```
# Simulate survey data with setting 1 and discretizing variables 3 categories
data = simulate(N = 3000, discretize = 3, setting = 1, seed = 123)

# Extract population and sample datasets
population = data$population
samples = data$samples

# Examine the simulated population data
head(population)
```

`svyBayesmod` *Bayesian Survey Model Estimation*

Description

This function fits a Bayesian model using Stan for survey data. It allows you to specify the outcome formula, the function for Stan, and apply different types of survey analysis, including weighted or unweighted models, for both sample and population data. The function supports posterior estimation, confidence intervals (CIs), and MCMC diagnostics.

Usage

```
svyBayesmod(
  svysmpl,
  svypopu,
  outcome_formula,
  BayesFun,
  subset = NULL,
  family = gaussian(),
  invlvls,
  weights = NULL,
  nskip = 1000,
  npost = 1000,
  nchain = 4,
  printmod = TRUE,
  doFigure = FALSE,
  useTrueSample = FALSE,
  stan_verbose = FALSE,
  HPD_CI = FALSE,
  seed = NULL
)
```

Arguments

`svysmpl`	A dataframe or tibble representing the sample data (`samples`). This should contain the outcome variable and any additional covariates.

`svypopu`	A dataframe or tibble representing the population data (`population`). This should contain all variables in the model.
`outcome_formula`	A formula for Stan, specifying the outcome and predictors in the model.
`BayesFun`	The name of the Stan function to be used for fitting the Bayesian model.
`subset`	A character vector representing filtering conditions to select subsets of the sample and population. Default is `NULL`, in which case the analysis is performed on the entire dataset. If specified, estimates for both the whole data and the subsets will be calculated.
`family`	The distribution family for the outcome variable. Currently, the following options are supported: `gaussian` for continuous outcomes and `binomial` for binary outcomes.
`invlvls`	A numeric vector specifying the confidence levels for the credible intervals (CIs). If more than one value is specified, multiple CIs will be calculated.
`weights`	A numeric vector of case weights. The length of this vector should match the number of cases in `svysmpl`. These weights will be used in the Bayesian model for weighted estimation.
`nskip`	An integer specifying the number of burn-in iterations for each chain in the MCMC for Stan models. Default is 1000.
`npost`	An integer specifying the number of posterior sampling iterations for each chain in the MCMC for Stan models. Default is 1000.
`nchain`	An integer specifying the number of MCMC chains for Stan models. Default is 4.
`printmod`	A logical scalar; if `TRUE`, posterior estimates will be printed.
`doFigure`	A logical scalar; if `TRUE`, MCMC diagnostic plots will be generated.
`useTrueSample`	A logical scalar; if `TRUE`, the estimator will use true sample information.
`stan_verbose`	A logical scalar; if `TRUE`, MCMC information will be printed during Stan model fitting.
`HPD_CI`	A logical scalar; if `TRUE`, the calculated credible intervals will be highest posterior density intervals (HPD). Otherwise, symmetric intervals will be used. Default is `FALSE`.
`seed`	An integer specifying the random seed for reproducibility. Default is `NULL`.

Value

A list containing the Bayesian estimates and confidence intervals (CIs) for each subset or the entire dataset. Each element in the list includes: - `estimate`: The Bayesian point estimate for the outcome. - `CI`: The credible intervals for the outcome estimate. - Other elements based on the specified confidence levels in `invlvls`.

Examples

```
## Example usage with survey data:
## Simulate sample and population data
data = simulate(N = 3000, discretize = 3, setting = 3, seed = 123)
```

```
population = data$population  # Get population data
samples = data$samples        # Get sample data
ipw = 1 / samples$true_pi    # Compute inverse probability weights

## Define outcome formula and Stan function
outcome_formula = "Y1 ~ Z1 + Z2 + Z3 + (1|auX_3)"
BayesFun = "stan_glmer"

## Fit Bayesian model using weighted survey data
bayes_model = svyBayesmod(svysmpl = samples, svypopu = population,
                          outcome_formula = outcome_formula,
                          BayesFun = BayesFun, weights = ipw,
                          family = gaussian(), nskip = 2000, npost = 2000,
                        nchain = 2, printmod = TRUE, invlvls = 0.95, stan_verbose = TRUE)
```

uwt — *Weighted or Unweighted Sample Mean*

Description

This function estimates the sample mean of an outcome variable using either weighted or unweighted methods. It supports calculating the sample mean with finite population correction (FPC) when a population dataset is provided. The method can also compute confidence intervals (CIs) for the sample mean using the specified distribution family (Gaussian or Binomial).

Usage

```
uwt(
  svysmpl,
  svyVar,
  svypopu = NULL,
  subset = NULL,
  family = gaussian(),
  invlvls,
  weights = NULL
)
```

Arguments

`svysmpl`	A dataframe or tibble representing the sample data (`samples`). This should contain the outcome variable and any additional covariates.
`svyVar`	The outcome variable to estimate the sample mean for (e.g., `Y1`).
`svypopu`	A dataframe or tibble representing the population data (`population`). This is used to compute the finite population correction (FPC) when calculating the sample mean. Default is `NULL`.

`subset`	A character vector representing filtering conditions to select subsets of the sample and population. Default is `NULL`, in which case the analysis is performed on the entire dataset. If subsets are specified, estimates for both the whole data and the subsets will be calculated.
`family`	The distribution family of the outcome variable. Supported options are: `gaussian` for continuous outcomes and `binomial` for binary outcomes.
`invlvls`	A numeric vector specifying the confidence levels (CIs) for the estimators. If more than one value is provided, multiple CIs will be calculated.
`weights`	A numeric vector of case weights. The length should match the number of cases in `svysmpl`. These weights are used for calculating the weighted sample mean.

Value

A list, where each element contains the sample mean estimate and CIs for a subset or the entire data. The list includes: - `est`: The sample mean estimate. - `se`: The standard error of the sample mean estimate. - `tCI`: The confidence intervals for the sample mean. - `sample_size`: The sample size for the subset or entire dataset. - `population_size`: The population size, if a population dataset is provided (applicable to finite population correction). The list is returned for each subset specified.

Examples

```
## Simulate data with nonlinear association (setting 3).
data = simulate(N = 3000, discretize = 3, setting = 3, seed = 123)
population = data$population  # Population data (3000 cases)
samples = data$samples        # Sample data (600 cases)
ipw = 1 / samples$true_pi    # Compute inverse probability weights

## Estimate the weighted sample mean with IPW
IPW_sample_mean = uwt(svysmpl = samples, svyVar = "Y1", svypopu = population,
                      subset = c("Z1 == 1 & Z2 == 1"), family = gaussian(),
                      invlvls = c(0.95), weights = ipw)
IPW_sample_mean

## Estimate the unweighted sample mean
unweighted_sample_mean = uwt(svysmpl = samples, svyVar = "Y1", svypopu = population,
                     subset = NULL, family = gaussian(), invlvls = c(0.95), weights = NULL)
unweighted_sample_mean
```

Index