Package 'sim.BA'

July 23, 2025

Title Simulation-Based Bias Analysis for Observational Studies

Version 0.1.0

Description Allows user to conduct a simulation based quantitative bias analysis using covariate structures generated with individual-level data to characterize the bias arising from unmeasured confounding. Users can specify their desired data generating mechanisms to simulate data and quantitatively summarize findings in an end-to-end application using this package.

License GPL (>= 2)

Depends R (>= 3.5.0)

Encoding UTF-8

RoxygenNote 7.3.1

Imports chk (>= 0.9.1), cobalt (>= 4.5.3), ggplot2 (>= 3.4.4), scales (>= 1.3.0), pbapply (>= 1.7-2), rlang (>= 1.1.3), stats, survival, utils

Suggests MatchIt (>= 4.5.5), WeightIt (>= 0.14.2), parallel, openxlsx (>= 4.2.5.2), knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Rishi Desai [aut, cre] (ORCID: <https://orcid.org/0000-0003-0299-7273>), Noah Greifer [aut] (ORCID: <https://orcid.org/0000-0003-3067-7154>)

Maintainer Rishi Desai <rdesai@bwh.harvard.edu>

Repository CRAN

Date/Publication 2024-04-22 14:42:35 UTC

Contents

`create_parameters` *Create parameters for data-generating models*

Description

`create_parameters()` facilitates creation of the `parameters` input to `simBA()`. This input contains information required to generate simulated dataset that will be analyzed within the simulation.

Usage

```
create_parameters(
  nbinary = 0,
  ncontinuous = 0,
  ncount = 0,
  unmeasured_conf = "u1",
  unmeasured_type = "binary",
  file = NULL
)
```

Arguments

`nbinary`, `ncontinuous`, `ncount`
: the number of binary, continuous, and count confounders to include, respectively. Default is 0.

`unmeasured_conf`
: the name of the unmeasured confounder. Default is `"u1"`.

`unmeasured_type`
: the type of variable for the unmeasured confounder. Allowable options include `"binary"`, `"continuous"`, and `"count"`. Default is `"binary"`; abbreviations allowed.

`file`
: optional; a string containing a path to a .csv or .xslx file where the output will be written to. If `NULL` (the default), no file will be written.

Value

A data.frame containing a skeleton of the parameter values, which must be filled in manually by the user. See *The* `parameters` *input object* section in the `simBA()` documentation for which columns will be present in the output. An additional column, Description, will also be produced, but it is not necessary to fill it in.

Examples

```
parameters <- create_parameters(nbinary = 6,
                                ncontinuous = 2,
                                ncount = 1,
                                unmeasured_conf = "u1",
                                unmeasured_type = "continuous")
```

parameters

---

plot.simBA *Plot the results of a simulation*

Description

plot() plots the output of a call to simBA(). The plot can contain either the estimated hazard ratios or standardized mean differences across simulations, each in a set of box plots.

Usage

```
## S3 method for class 'simBA'
plot(x, type = "balance", ...)
```

Arguments

x	a simBA object; the output of a call to simBA().
type	the type of plot to produce; allowable options include "balance" (default), "hr", and "bias". Abbreviations allowed. See Details.
...	further arguments passed to ggplot2::geom_boxplot().

Details

The balance plot plots absolute standardized mean differences. Vertical lines are placed at 0 (solid) and .1 (dashed). The hazard ratio (HR) plot plots hazard ratios on a log scale for the x-axis. Vertical lines are placed at 1 (solid) and the true marginal HR (dashed). The bias plot plots the relative error in the HR with a vertical line at 0% (indicating no error).

Value

A ggplot object, which can be modified using ggplot2 syntax.

See Also

simBA() for performing the simulation.

Examples

```
# See help("simBA") for examples.
```

simBA *Run a simulation to assess due to unmeasured confounding*

Description

simBA() runs a simulation to compute the magnitude of the bias in a hazard ratio in the presence of unmeasured confounding, possibly when proxies are available.

Usage

```
simBA(
  parameters,
  iterations = 500,
  size = 1000,
  treatment_prevalence,
  treatment_coeff,
  outcome_prevalence,
  dist = "exponential",
  unmeasured_conf,
  n_proxies = 0,
  proxy_type = "binary",
  corr = NULL,
  adj = "matching",
  estimand = "ATT",
  adj_args = list(),
  keep_data = FALSE,
  cl = NULL,
  verbose = TRUE
)
```

Arguments

parameters	either a data.frame containing information about the data generation used in the simulation or a string containing the path to a .csv or .xlsx file containing such information. See Details for what this should contain and create_parameters() to create a skeleton of this object.
iterations	the number of simulation iterations. Default is 500.
size	the size of each sample to be generated. Default is 1000.
treatment_prevalence	the desired prevalence of treatment. Should be a number between 0 and 1.
treatment_coeff	the coefficient on the treatment variable in the data-generating model for survival times.
outcome_prevalence	the desired prevalence of the outcome. This is used to specify a censoring time for the data-generating model.

`dist` the distribution to use to generate survival times. Allowable options include `"exponential"` (default) and `"weibull"`. Abbreviations allowed.

`unmeasured_conf` the name of the variable in `parameters` corresponding to the unmeasured confounder.

`n_proxies` the number of proxies for the unmeasured confounder to include in the simulation. Default is 0.

`proxy_type` when `n_proxies` is greater than 0, the type of variable the proxies should be. Allowable options include `"binary"` (default) and `"continuous"`. Abbreviations allowed.

`corr` when `n_proxies` is greater than 0, the desired correlations between the proxy variable and the unmeasured confounder in the simulation. Should be length 1 (in which case all proxies have the same correlation with the unmeasured confounder) or length equal to `n_proxies`.

`adj` string; the method used to adjust for the confounders. Allowable options include `"matching"` (the default), which uses `MatchIt::matchit()`, and `"weighting"`, which uses `WeightIt::weightit()`. Abbreviations allowed.

`estimand` string; the desired estimand to target. Allowable options include `"ATT"` (default), `"ATC"`, and `"ATE"`. Note this is also passed to the `estimand` argument of the function used for adjustment as specified by `adj` if omitted in `adj_args`.

`adj_args` a list of arguments passed to `MatchIt::matchit()` or `WeightIt::weightit()` depending on the argument to `adj`. If not supplied, the parameter defaults will be used. Take care to specify these arguments to ensure the adjustment method is as desired.

`keep_data` `logical`; whether to keep the datasets generated in each simulation. Default is `FALSE`. Setting to `TRUE` will make the output object large.

`cl` a cluster object created by `parallel::makeCluster()`, or an integer to indicate number of child-processes (integer values are ignored on Windows) for parallel evaluations. See `pbapply::pbapply()` for details. Default is `NULL` for no parallel evaluation.

`verbose` whether to print information about the progress of the simulation, including a progress bar. Default is `TRUE`.

Details

`simBA()` runs a simulation study to examine the impact of an unmeasured confounder on the bias of the marginal hazard ratio when using matching or weighting to adjust for observed confounders and, optionally, proxies of the unmeasured confounder. The user must specify the simulation data-generating model using the `parameters` argument and other arguments that control generation of the treatment, outcome, and proxies. Requirements for the `parameters` input are described below. In addition, the user must specify the form of adjustment used (matching or weighting) using the `adj` argument, the desired estimand using the `estimand` argument, and any other arguments passed to `adj_args` to control the matching/weighting method. Note by default, the ATT is targeted, even though the usual default estimand for weighting using `WeightIt::weightit()` is the ATE.

Broadly, the `parameters` input contains the name of the measured and unmeasured confounders, their variable types (binary, continuous, or count), their distributions, and their coefficients in the

treatment and outcome models. These values are used to generate a synthetic dataset of size corresponding to the `size` argument, which additionally contains the true propensity score used to simulate the treatment, the treatment itself, and the outcome (i.e., survival time and whether an event occurred). When proxies are requested (i.e., `n_proxies` set to 1 or greater), proxies for the unmeasured confounder are additionally generated and appended to the synthetic dataset.

In each iteration, a synthetic dataset is generated, and then that dataset is analyzed. First, a crude marginal hazard ratio is estimated by fitting a Cox proportional hazards model for the survival times and events as a function just of the treatment. Then, the dataset is adjusted using matching or weighting with the measured covariates, and a second hazard ratio is estimated as above, this time in the matched or weighted sample. If proxies are requested, the dataset is adjusted again using matching or weighting with the measured covariates and proxies, and a third hazard ratio is estimated as above. In addition, the balance (as measured by the standardized mean difference [SMD]) is reported for the unmeasured confounder and proxies before adjustment and after each round of matching or weighting.

The data-generating model:

The data-generating model for the outcome corresponds to a Cox proportional hazards model as described by Bender et al. (2005). The coefficients on the measured and unmeasured confounders in the outcome model are specified in the `parameters` input, and the coefficient on the treatment variable is specified by the `treatment_coeff` argument. The treatment is generated as a Bernoulli variable with probability equal to the true propensity score, which is generated according to a logistic regression model with the coefficients on the confounders specified in the `parameters` input.

The proxies, if requested, are generated such that their correlation with the unmeasured confounder is exactly equal to the values supplied to `corr`. The confounder are generated as uncorrelated variables according to the distribution supplied in the `parameters` input. Binary variables are generated as Bernoulli variables with probability equal to the supplied prevalence. Continuous variables are generated as Gaussian (Normal) variables with mean and standard deviation equal to their supplied values. Count variables are generated as Poisson variables with mean equal to its supplied value.

Some parameters are determined first by generating a dataset with one million observations. With this dataset, the intercept of the true propensity score model is selected as that which yields a treatment prevalence equal to that specified in the `treatment_prevalence` argument, and the censoring time for the outcomes is selected as that which yields an outcome event prevalence equal to that specified in the `outcome_prevalence` argument. In addition, the true marginal hazard ratio is computed using this dataset by generating potential outcomes under each treatment and fitting a Cox model of the potential outcome survival times and events as a function of the treatment under which the potential outcome was generated as recommended by Austin (2013).

The `parameters` input object:

The `parameters` input must be of a specified form in order to be processed correctly. It must be a data.frame with one row for each confounder to be generated with (at least) the following columns (which are case-sensitive):

`Variable` the name of the variable

`Type` the variable type; either binary, continuous, or count (see above for how these correspond to the distribution used to generate the variable)

`prevalence` the prevalence for binary variables (should be blank for all other variable types)

`mean` the mean for continuous and count variable (should be blank for binary variables)

`sd` the standard deviation for continuous variables (should be blank for all other variable types)

`coeff_treatment_model` the coefficient on that variable in the true propensity score model for the treatment (can be blank for any variable that doesn't affect treatment)

`coeff_outcome_model` the coefficient on that variable in the outcome model for the treatment (can be blank for any variable that doesn't affect the outcome)

The variable name supplied to `unmeasured_conf` must be present in the `parameters` input, and it must have nonzero values in both the `coeff_treatment_model` and `coeff_outcome_model` columns (or else it would not be a true confounder).

To automatically create a skeleton of the `parameters` input for you to fill in yourself, use `create_parameters()`.

Value

A `simBA` object, which contains the simulation outputs and their summaries. This includes the following components:

`sim_out` the complete simulation results, a list with an entry for each iteration including the table of log hazard ratios, the table of standardized mean differences, and the generated dataset (if `keep_data = TRUE`)

`parameters` the table of parameters supplied to `parameters` after some processing

`SMD_table` the table of average standardized mean differences for the unmeasured confounder and proxies before and after matching across all iterations

`HR_table` the table of estimated and true hazard ratios averaged across all iterations (note that log hazard ratios are averaged before exponentiating the average)

Basic `print()` and `summary()` methods are available. See `[plot.simBA()]` for plotting.

References

Austin PC. The performance of different propensity score methods for estimating marginal hazard ratios. *Statistics in Medicine*. 2013;32(16):2837-2849. doi:10.1002/sim.5705

Bender R, Augustin T, Blettner M. Generating survival times to simulate Cox proportional hazards models. *Statistics in Medicine*. 2005;24(11):1713-1723. doi:10.1002/sim.2059

See Also

`create_parameters()` for creating the `parameters` input; `plot.simBA()` for plotting the results. `MatchIt::matchit()` and `WeightIt::weightit()` for the functions used for matching and weighting, respectively, which detail the defaults used by these methods and allowable arguments that can be passed to `adj_args`.

Examples

```
# Get parameters example; can also create
# with `create_parameters()`
parameters <- read.csv(system.file("extdata", "parameters.csv",
                                   package = "sim.BA"))

  # Run simulation; adjustment via PS weighting for
  # the ATE
```

```
sim <- simBA(parameters,
             iterations = 50,
             size = 200,
             treatment_prevalence = .2,
             treatment_coef = -.25,
             outcome_prevalence = .5,
             unmeasured_conf = "u1",
             n_proxies = 2,
             proxy_type = "binary",
             corr = c(.5, .8),
             verbose = FALSE,
             # Adjustment arguments
             adj = "weighting",
             estimand = "ATE",
             adj_args = list(method = "glm"))

sim

summary(sim)

plot(sim, "balance")

plot(sim, "hr")
```

Index