Package 'PStrata'

January 20, 2025

Type Package

Title Principal Stratification Analysis in R

Version 0.0.5

Date 2023-12-02

Encoding UTF-8

Maintainer Bo Liu <bl226@duke.edu>

Description Estimating causal effects in the presence of post-treatment confounding using principal stratification. 'PStrata' allows for customized monotonicity assumptions and exclusion restriction assumptions, with automatic full Bayesian inference supported by 'Stan'. The main function to use in this package is PStrata(), which provides posterior estimates of principal causal effect with uncertainty quantification. Visualization tools are also provided for diagnosis and interpretation. See Liu and Li (2023) <arXiv:2304.02740> for details.

Depends R (>= 3.5.0)

Collate PStrata-package.R PSFormula.R PStrataInfo.R survival.R prior.R make_standata.R PSObject.R make_stancode.R PSSample.R PStrata.R PSOutcome.R sim_data_normal.R sim_data_Cox.R PSContrast.R

License GPL (>= 2)

Suggests R.rsp

VignetteBuilder R.rsp

Imports ggplot2, rstan, lme4, abind, dplyr, purrr, stringr, stats

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Bo Liu [aut, cre],
Fan Li [ctb]

Repository CRAN

Date/Publication 2023-12-03 03:10:02 UTC

Contents

`PStrata-package` *PStrata: Principal STRATification Analysis for Data with Post-Randomization Confounding*

Description

The **PStrata** package is designed for estimating causal effects in the presense of post-treatment confounding using principal stratification. It provides an interface to fit the Bayesian principal stratification model, which is a complex mixture model, using **Stan**, a C++ package for obtaining full Bayesian inference. The formula syntax is an extended version of the syntax applied in many regression functions and packages, such as `lm`, `glm` and **lme4**, to provide a simple interface. A wide variety of distributions and link functions are supported, allowing users to fit linear, binary or count data, and survival models with principal stratification. Further modeling options include multiple post-treatment confounding variables and cluster random effects. The monotonicity and exclusion restriction assumptions can be easily applied, and prior specifications are flexible and encourage users to reflect their prior belief. In addition, all parameters can be inferred from the posterior distribution, which enables further analysis other than provided by the package. A frequentist weighting-based triply-robust estimator is also implemented for both ordinary outcomes and survival outcomes.

Details

The Bayesian principal stratification analysis relies on two models, the principal stratum model and the outcome model. The main function of **PStrata** is `PStrata`, which uses formula syntax to specify these models. Based on the supplied formulas, data and additional information allowing users to specify assumptions and prior distributions, it automatically generates the Stan code via `make_stancode` and `make_standata`, and fits the model using **Stan**.

The estimated probability for each principal stratum and the estimated mean response are calculated with **Stan** as it is faster and more space-efficient. However, a large number of post-processing methods can also be applied. `summary` is perfectly suited for an overview of the estimated parameters, and `plot` provides visualization of the principal stratification and the outcome distribution.

Because **PStrata** heavily relies on **Stan** for posterior sampling, a C++ compiler is required. The program **Rtools** (available on https://cran.r-project.org/bin/windows/Rtools/) comes with a C++ compiler for Windows. On Mac, Xcode is suggested. For further instructions on how to get the compilers running, please refer to the prerequisites section at the RStan-Getting-Started page.

References

The Stan Development Team. Stan Modeling Language User's Guide and Reference Manual. https://mc-stan.org/users/documentation/

Stan Development Team (2020). RStan: the R interface to Stan. R package version 2.21.2. https://mc-stan.org/

See Also

`PStrata`

`make_stancode` ***Stan** Code for* **PStrata** *Models*

Description

Generate the **Stan** code corresponding to the model, which is read by **Stan** to do sampling.

Usage

```
make_stancode(PSobject, filename = NULL, debug = FALSE)
```

Arguments

`PSobject`	an object of class `PSobject`
`filename`	(optional) string. If not `NULL`, the stan file will be saved via `cat` in a text file named after the string supplied.
`debug`	only for testing in development mode. Will be removed in future release.

Value

A string, which can be printed on screen using `cat`.

make_standata *Data for* **PStrata** *Models*

Description

Generate data for **PStrata** models to be passed to **Stan**

Usage

```
make_standata(PSobject)
```

Arguments

PSobject	an object of class PSObject

Value

a named list of objects containing the required data to fit a **PStrata** model with **Stan**.

prior *Prior functions*

Description

Define prior functions used in PStrata.

Usage

```
prior_flat()

prior_normal(mu = 0, sigma = 1)

prior_t(mu = 0, sigma = 1, df = 1)

prior_cauchy(mu = 0, sigma = 1)

prior_lasso(mu = 0, sigma = 1)

prior_logistic(mu = 0, sigma = 1)

prior_chisq(df = 1)

prior_inv_chisq(df = 1)

prior_exponential(beta = 1)
```

```
prior_gamma(alpha = 1, beta = 1)

prior_inv_gamma(alpha = 1, beta = 1)

prior_weibull(alpha = 1, sigma = 1)
```

Arguments

`mu`, `sigma`, `df`, `alpha`, `beta`
: parameters for the prior distribution

Value

A list, including the following items.

name name of the distribution

type type of the distribution, one character string of "real" or "positive"

args a named list of all the input parameters

call a function call object of the prior distribution on the parameters

---

`PSContrast` *Contrast of potential outcome for principal stratification analysis*

Description

Create an object that represents contrast of potential outcomes by treatment arms, strata or time points.

Usage

```
PSContrast(
  outcome,
  S = NULL,
  Z = NULL,
  T = NULL,
  type = c("all", "sequential", "cycle")
)
```

Arguments

`outcome`	an object of class `PSoutcome` or `PSContrast`
`S`	a vector denoting which strata to take contrasts. Default is `NULL` indicating no contrasts are taken. Set to 'TRUE' to take contrasts between all strata.
`Z`	a vector denoting which treatment arms to take contrasts. Default is `NULL` indicating no contrasts are taken. Set to 'TRUE' to take contrasts between all treatment arms.

T	a vector denoting which time points to take contrasts. Default is NULL indicating no contrasts are taken. Set to 'TRUE' to take contrasts between all time points. This is used only when 'object' is obtained under survival outcome.
type	Either "all" (default), "sequential" or "cycle". If "all", every pairwise contrasts are taken. If "sequential", contrasts are taken over every consecutive pairs. If "cycle", contrasts are taken over every consecutive pairs and also between the first and the last levels.

Value

An S3 object of class PSContrast and PSOutcome, containing

outcome_array	A num_strata * num_treatment * num_iter array of contrast if the outcome type is non-survival or a num_strata * num_treatment * num_time_points * num_iter array of contrast if the outcome type is survival.
is.survival	A boolean value, whether the outcome type is survival.
time_points	The time points at which the outcome is evaluated, if the outcome type is survival.

The S3 method summary and plot can be applied to the returned object.

PSFormula *Set up a model formula for use in* **PStrata**

Description

Set up a model formula for use in **PStrata** package allowing users to specify the treatment indicator, the post-randomization confounding variables, the outcome variable, and possibly the covariates. For survival outcome, a censoring indicator is also specified. Users can also define (potentially non-linear) transforms of the covariates and include random effects for clusters.

Usage

```
PSFormula(formula, data)
```

Arguments

formula	an object of class formula (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given in 'Details'.
data	a data frame containing the variables named in formula.

Details

Two models are required for the principal stratification analysis: the principal stratum model and the outcome model.

General formula structure: For the principal stratum model, the `formula` argument accepts formulas of the following syntax:

`treatment + postrand ~ terms`

The `treatment` variable refers to the name of the binary treatment indicator. The `postrand` variable refers to the name of the binary post-randomization confounding variable. The `terms` part includes all of the predictors used for the principal stratum model.

For the outcome model, the `formula` argument accepts formulas of the similar syntax:

`response [+ observed] ~ terms`

The `response` variable refers to the name of the outcome variable. The `terms` part includes all of the predictors used for the outcome model. The `observed` variable shall not be used for ordinary response. When the true response is subject to right censoring (also called survival outcome in relevant literature), the `response` variable should refer to the observed or censored response, and the `observed` variable should be an indicator of whether the true response is observed. For example, suppose the true time for an event is T and the time of censoring is C, Then, the `response` variable should refer to $\min(T, C)$, the actual time of the event or censoring, whichever comes earlier, and the indicator `observed` is 1 if $T < C$ and 0 otherwise.

The `terms` specified in the principal stratum model and the outcome model can be different.

Multiple post-randomization confounding variables: If multiple post-randomization confounding variables exist, one can specify all of them using the following syntax:

`treatment + postrand_1 + postrand_2 + ... + postrand_n ~ terms`

The post-randomization confounding variables are provided in place of `postrand_1` to `postrand_n`. Up to this version, all of these variables should be binary indicators. Note that the order of these post-randomization confounding variables will not affect the result of the estimation of the parameters, but it will be important in specifying other parameters, such as `strata` and `ER` (see `PStrata`).

Non-linear transformation of the predictors: The syntax for the predictors follow the conventions as used in `link{formula}`. The part `terms` consists of a series of terms concatenated by `+`, each term being the name of a variable, or the interaction of several variables separated by `:`.

Apart from `+` and `:`, a number of other operators are also useful. The `*` operator is a short-hand for factor crossing: `a*b` is interpreted as `a + b + a:b`. The `^` operator means factor crossing to a specific degree. For example, `(a + b + c)^2` is interpreted as `(a + b + c) * (a + b + c)`, which is identical to `a + b + c + a:b + a:c + b:c`. The `-` operator removes specified terms, so that `(a + b + c)^2 - a:b` is identical to `a + b + c + a:c + b:c`. The `-` operator can be also used to remove the intercept term, such as `x - 1`. One can also use `x + 0` to remove the intercept term.

Arithmetic expressions such as `a + log(b)` are also legal. However, arithmetic expressions may contain special symbols that are defined for other use, such as `+`, `*`, `^` and `-`. To avoid confusion, the function `I()` can be used to bracket portions where the operators should be interpreted in arithmetic sense. For example, in `x + I(y + z)`, the term `y + z` is interpreted as the sum of `y` and `z`.

Group level random effect: When effects assumed to vary across grouping variables are considered, one can specify such effects by adding terms in the form of `gterms | group`, where `group`

refers to the group indicator (usually a `factor`), and `gterms` specifies the terms whose coefficients are group-specific, drawn from a population normal distribution.

The most common situation for group level random effect is to include group-specific intercepts to account for unmeasured confounding. For example, `x + y + (1 | g)` specifies a model with population predictors `x` and `y`, as well as random intercept for each level of `g`.

For more complex random effect structures, refer to `lme4::lmer`. However, structures other than simple random intercepts and slopes may lead to unexpected behaviors.

Value

`PSFormula` returns an object of class `PSFormula`, which is a `list` containing for following components.

`full_formula` input formula as is

`data` input data frame

`fixed_eff_formula` input formula with only fixed effects

`response_names` character vector with names of variables that appear on the left hand side of input formula

`has_random_effect` logical indicating whether random effects are specified in the input formula

`has_intercept` logical indicating whether the input formula has an intercept

`fixed_eff_names` character vector with names of all variables included as fixed effects

`fixed_eff_count` integer indicating the number of variables (factors are converted to and counted as dummy variables)

`fixed_eff_matrix` fixed-effect design matrix

`random_eff_list` a list containing information for each random effect. Such information is a list with the corresponding design matrix, the term names and the factor levels.

See Also

`formula`, `lmer`.

Examples

```
df <- data.frame(
  X = 1:10,
  Z = c(0,0,0,0,0,1,1,1,1,1),
  D = c(0,0,0,1,1,1,0,0,1,1),
  R = c(1,1,1,1,2,2,2,3,3,3)
 )
PSFormula(Z + D ~ X + I(X^2) + (1 | R), df)
```

PSObject *Create a Principal Stratification Object*

Description

Create an object containing essential information to create the Stan file and data for Stan to draw posterior samples. Such information includes the specified model for principal stratum and outcome, the type of outcome, assumptions, and prior specification, etc.

Usage

```
PSObject(
  S.formula,
  Y.formula,
  Y.family,
  data = NULL,
  strata = NULL,
  ER = NULL,
  prior_intercept = prior_flat(),
  prior_coefficient = prior_normal(),
  prior_sigma = prior_inv_gamma(),
  prior_alpha = prior_inv_gamma(),
  prior_lambda = prior_inv_gamma(),
  prior_theta = prior_normal(),
  survival.time.points = 50
)
```

Arguments

`S.formula`, `Y.formula`
: an object of class "`PSFormula`" (or an object of class "`formula`" that can be coerced to that class with `data` provided) specifying the model for principal stratum and outcome respectively. See `PSFormula` for details.

`Y.family`
: an object of class "`family`": specifying the parametric family of the model for the response and the link function. See the documentation for `glm` for details on how such model fitting takes place. Supported families and corresponding link functions are presented in 'Details' below.

`data`
: (optional) a data frame object. This is required when either `S.formula` or `Y.formula` is a `formula` object, to coerce it into a `PSFormula` object. When this happens, the data frame should contain all of the variables with names given in `S.formula` or `Y.formula`.

`strata`, `ER`
: arguments to define the principal strata. See `PStrataInfo` for details.

  Alternatively, one can pass an object of class `PStrataInfo` to `strata`, and `ER` will be ignored.

`prior_intercept`, `prior_coefficient`, `prior_sigma`, `prior_alpha`, `prior_lambda`, `prior_theta`
: prior distribution for corresponding parameters in the model.

`survival.time.points`
: a vector of time points at which the estimated survival probability is evaluated (only used when the type of outcome is survival), or an integer specifying the number of time points to be chosen. By default, the time points are chosen with equal distance from 0 to the 90% quantile of the observed outcome.

Details

The supported `family` objects include two types: native families for ordinary outcome and `survival` family for survival outcome.

For ordinary outcome, the below families and links are supported. See `family` for more details.

family	link
`binomial`	`logit`, `probit`, `cauchit`, `log`, `cloglog`
`gaussian`	`identity`, `log`, `inverse`
`Gamma`	`inverse`, `identity`, `log`
`poisson`	`log`, `identity`, `log`
`inverse.gamma`	`1/mu^2`, `inverse`, `identity`, `log`

The `quasi` family is not supported for the current version of the package.

For survival outcome, the `family` object is created by `survival(method = "Cox", link = "identity")`, where `method` can be either `"Cox"` for Weibull-Cox model or `"AFT"` for accelerated failure time model. See `survival` for more details. For the current version, only `"identity"` is used as the link function.

The `gaussian` family and the `survival` family with `method = "AFT"` introduce an additional parameter `sigma` for the standard deviation, whose prior distribution is specified by `prior_sigma`. Similarly, `prior_alpha` specifies the prior distribution of `alpha` for `Gamma` family, `prior_lambda` specifies the prior distribution of `theta` for `inverse.gaussian` family, and `prior_theta` specifies the prior distribution of `theta` for `survival` family with `method = "Cox"`.

The models for principal stratum `S.formula` and response `Y.formula` also involve a linear combination of terms, where the prior distribution of the intercept and coefficients are specified by `prior_intercept` and `prior_coefficient` respectively.

Value

A list, containing important information describing the principal stratification model.

`S.formula`, `Y.formula`
: A `PSFormula` object converted from the input `S.formula` and `Y.formula`

`Y.family`
: Same as input.

`is.survival`
: A boolean value. `TRUE` if `Y.family` is `survival_Cox` or `survival_AFT`.

`strata_info`
: A `PStrataInfo` object converted from the input `strata` and `ER`.

prior_intercept, prior_coefficient, prior_sigma, prior_alpha, prior_lambda, prior_theta
: Same as input.

survival.time.points
: A list of time points at which the estimated survival probability is evaluated.

SZDG_table
: A matrix. Each row corresponds to a valid (stratum, treatment, confounder, group) combination.

Z_names
: A character vector. The names of the levels of the treatment.

Examples

```
df <- data.frame(
  Z = rbinom(10, 1, 0.5),
  D = rbinom(10, 1, 0.5),
  Y = rnorm(10),
  X = 1:10
)

PSObject(
  S.formula = Z + D ~ X,
  Y.formula = Y ~ X,
  Y.family = gaussian("identity"),
  data = df,
  strata = c(n = "00*", c = "01", a = "11*")
)

#------------------------------

PSObject(
  S.formula = Z + D ~ 1,
  Y.formula = Y ~ 1,
  Y.family = gaussian("identity"),
  data = sim_data_normal,
  strata = c(n = "00*", c = "01", a = "11*")
)
```

PSOutcome *Estimated potential outcome for principal stratification analysis*

Description

Create an object useful to present the potential outcomes under each treatment arm for each principal stratum. Contrasts between treatment arms or principal strata are easy to obtain from this object.

Usage

```
PSOutcome(PStrataObj, type = c("probability", "RACE"))
```

Arguments

PStrataObj	an object of class PStrata or PStrata_survival
type	whether the causal estimand is survival probability or RACE, ignored for non-survival outcomes.

Value

An S3 object of type PSOutcome, containing

outcome_array	A num_strata * num_treatment * num_iter array of mean outcome if the outcome type is non-survival or a num_strata * num_treatment * num_time_points * num_iter array of mean outcome if the outcome type is survival.
is.survival	A boolean value, whether the outcome type is survival.
time_points	The time points at which the outcome is evaluated, if the outcome type is survival.

The S3 method summary and plot can be applied to the returned object.

PSSample *Sample with* Stan

Description

Sample from the posterior distribution by calling stan. Check stan for details of the arguments.

Usage

```
PSSample(
  file,
  model_name = "anon_model",
  model_code = "",
  fit = NA,
  data = list(),
  pars = NA,
  chains = 4,
  iter = 2000,
  warmup = floor(iter/2),
  thin = 1,
  init = "random",
  seed = sample.int(.Machine$integer.max, 1),
  algorithm = c("NUTS", "HMC", "Fixed_param"),
  control = NULL,
  sample_file = NULL,
  diagnostic_file = NULL,
  save_dso = TRUE,
  verbose = FALSE,
```

```
  include = TRUE,
  cores = getOption("mc.cores", 1L),
 open_progress = interactive() && !isatty(stdout()) && !identical(Sys.getenv("RSTUDIO"),
   "1"),
  ...,
  boost_lib = NULL,
  eigen_lib = NULL
)
```

Arguments

file	The path to the Stan program to use. file should be a character string file name or a connection that R supports containing the text of a model specification in the Stan modeling language. A model may also be specified directly as a character string using the model_code argument, but we recommend always putting Stan programs in separate files with a .stan extension. The stan function can also use the Stan program from an existing stanfit object via the fit argument. When fit is specified, the file argument is ignored.
model_name	A string to use as the name of the model; defaults to "anon_model". However, the model name will be derived from file or model_code (if model_code is the name of a character string object) if model_name is not specified. This is not a particularly important argument, although since it affects the name used in printed messages, developers of other packages that use **rstan** to fit models may want to use informative names.
model_code	A character string either containing the model definition or the name of a character string object in the workspace. This argument is used only if arguments file and fit are not specified.
fit	An instance of S4 class stanfit derived from a previous fit; defaults to NA. If fit is not NA, the compiled model associated with the fitted result is re-used; thus the time that would otherwise be spent recompiling the C++ code for the model can be saved.
data	A named list or environment providing the data for the model, or a character vector for all the names of objects to use as data. See the **Passing data to Stan** section below.
pars	A character vector specifying parameters of interest to be saved. The default is to save all parameters from the model. If include = TRUE, only samples for parameters named in pars are stored in the fitted results. Conversely, if include = FALSE, samples for all parameters *except* those named in pars are stored in the fitted results.
chains	A positive integer specifying the number of Markov chains. The default is 4.
iter	A positive integer specifying the number of iterations for each chain (including warmup). The default is 2000.
warmup	A positive integer specifying the number of warmup (aka burnin) iterations per chain. If step-size adaptation is on (which it is by default), this also controls the number of iterations for which adaptation is run (and hence these warmup

samples should not be used for inference). The number of warmup iterations should be smaller than `iter` and the default is `iter/2`.

`thin` A positive integer specifying the period for saving samples. The default is 1, which is usually the recommended value. Unless your posterior distribution takes up too much memory we do *not* recommend thinning as it throws away information. The tradition of thinning when running MCMC stems primarily from the use of samplers that require a large number of iterations to achieve the desired effective sample size. Because of the efficiency (effective samples per second) of Hamiltonian Monte Carlo, rarely should this be necessary when using Stan.

`init` Specification of initial values for all or some parameters. Can be the digit `0`, the strings `"0"` or `"random"`, a function that returns a named list, or a list of named lists:

`init="random"` **(default):** Let Stan generate random initial values for all parameters. The seed of the random number generator used by Stan can be specified via the `seed` argument. If the seed for Stan is fixed, the same initial values are used. The default is to randomly generate initial values between `-2` and `2` *on the unconstrained support*. The optional additional parameter `init_r` can be set to some value other than `2` to change the range of the randomly generated inits.

`init="0"`, `init=0`**:** Initialize all parameters to zero on the unconstrained support.

inits via list: Set inital values by providing a list equal in length to the number of chains. The elements of this list should themselves be named lists, where each of these named lists has the name of a parameter and is used to specify the initial values for that parameter for the corresponding chain.

inits via function: Set initial values by providing a function that returns a list for specifying the initial values of parameters for a chain. The function can take an optional parameter `chain_id` through which the `chain_id` (if specified) or the integers from 1 to `chains` will be supplied to the function for generating initial values. See the **Examples** section below for examples of defining such functions and using a list of lists for specifying initial values.

When specifying initial values via a `list` or `function`, any parameters for which values are not specified will receive initial values generated as described in the `init="random"` description above.

`seed` The seed for random number generation. The default is generated from 1 to the maximum integer supported by R on the machine. Even if multiple chains are used, only one seed is needed, with other chains having seeds derived from that of the first chain to avoid dependent samples. When a seed is specified by a number, `as.integer` will be applied to it. If `as.integer` produces `NA`, the seed is generated randomly. The seed can also be specified as a character string of digits, such as `"12345"`, which is converted to integer.

Using R's `set.seed` function to set the seed for Stan will not work.

`algorithm` One of the sampling algorithms that are implemented in Stan. The default and preferred algorithm is `"NUTS"`, which is the No-U-Turn sampler variant of Hamiltonian Monte Carlo (Hoffman and Gelman 2011, Betancourt 2017). Currently the other options are `"HMC"` (Hamiltonian Monte Carlo), and `"Fixed_param"`.

When "Fixed_param" is used no MCMC sampling is performed (e.g., for simulating with in the generated quantities block).

control A named list of parameters to control the sampler's behavior. It defaults to NULL so all the default values are used. First, the following are adaptation parameters for sampling algorithms. These are parameters used in Stan with similar names here.

- adapt_engaged (logical)
- adapt_gamma (double, positive, defaults to 0.05)
- adapt_delta (double, between 0 and 1, defaults to 0.8)
- adapt_kappa (double, positive, defaults to 0.75)
- adapt_t0 (double, positive, defaults to 10)
- adapt_init_buffer (integer, positive, defaults to 75)
- adapt_term_buffer (integer, positive, defaults to 50)
- adapt_window (integer, positive, defaults to 25)

In addition, algorithm HMC (called 'static HMC' in Stan) and NUTS share the following parameters:

- stepsize (double, positive, defaults to 1) Note: this controls the *initial* stepsize only, unless adapt_engaged=FALSE.
- stepsize_jitter (double, [0,1], defaults to 0)
- metric (string, one of "unit_e", "diag_e", "dense_e", defaults to "diag_e")

For algorithm NUTS, we can also set:

- max_treedepth (integer, positive, defaults to 10)

For algorithm HMC, we can also set:

- int_time (double, positive)

For test_grad mode, the following parameters can be set:

- epsilon (double, defaults to 1e-6)
- error (double, defaults to 1e-6)

sample_file An optional character string providing the name of a file. If specified the draws for *all* parameters and other saved quantities will be written to the file. If not provided, files are not created. When the folder specified is not writable, tempdir() is used. When there are multiple chains, an underscore and chain number are appended to the file name.

diagnostic_file An optional character string providing the name of a file. If specified the diagnostics data for *all* parameters will be written to the file. If not provided, files are not created. When the folder specified is not writable, tempdir() is used. When there are multiple chains, an underscore and chain number are appended to the file name.

save_dso Logical, with default TRUE, indicating whether the dynamic shared object (DSO) compiled from the C++ code for the model will be saved or not. If TRUE, we can draw samples from the same model in another R session using the saved DSO (i.e., without compiling the C++ code again). This parameter only takes effect if fit is not used; with fit defined, the DSO from the previous run is used. When

save_dso=TRUE, the fitted object can be loaded from what is saved previously and used for sampling, if the compiling is done on the same platform, that is, same operating system and same architecture (32bits or 64bits).

verbose — TRUE or FALSE: flag indicating whether to print intermediate output from Stan on the console, which might be helpful for model debugging.

include — Logical scalar defaulting to TRUE indicating whether to include or exclude the parameters given by the pars argument. If FALSE, only entire multidimensional parameters can be excluded, rather than particular elements of them.

cores — The number of cores to use when executing the Markov chains in parallel. The default is to use the value of the "mc.cores" option if it has been set and otherwise to default to 1 core. However, we recommend setting it to be as many processors as the hardware and RAM allow (up to the number of chains). See detectCores if you don't know this number for your system.

open_progress — Logical scalar that only takes effect if cores > 1 but is recommended to be TRUE in interactive use so that the progress of the chains will be redirected to a file that is automatically opened for inspection. For very short runs, the user might prefer FALSE.

... — Other optional parameters:

- chain_id (integer)
- init_r (double, positive)
- test_grad (logical)
- append_samples (logical)
- refresh(integer)
- save_warmup(logical)
- deprecated: enable_random_init(logical)

chain_id can be a vector to specify the chain_id for all chains or an integer. For the former case, they should be unique. For the latter, the sequence of integers starting from the given chain_id are used for all chains.

init_r is used only for generating random initial values, specifically when init="random" or not all parameters are initialized in the user-supplied list or function. If specified, the initial values are simulated uniformly from interval [-init_r, init_r] rather than using the default interval (see the manual of (cmd)Stan).

test_grad (logical). If test_grad=TRUE, Stan will not do any sampling. Instead, the gradient calculation is tested and printed out and the fitted stanfit object is in test gradient mode. By default, it is FALSE.

append_samples (logical). Only relevant if sample_file is specified *and* is an existing file. In that case, setting append_samples=TRUE will append the samples to the existing file rather than overwriting the contents of the file.

refresh (integer) can be used to control how often the progress of the sampling is reported (i.e. show the progress every refresh iterations). By default, refresh = max(iter/10, 1). The progress indicator is turned off if refresh <= 0.

Deprecated: enable_random_init (logical) being TRUE enables specifying initial values randomly when the initial values are not fully specified from the user.

save_warmup (logical) indicates whether to save draws during the warmup phase and defaults to TRUE. Some memory related problems can be avoided by setting it to FALSE, but some diagnostics are more limited if the warmup draws are not stored.

boost_lib — The path for an alternative version of the Boost C++ to use instead of the one in the **BH** package.

eigen_lib — The path for an alternative version of the Eigen C++ library to the one in **RcppEigen**.

Value

An object of S4 class rstan::stanfit.

PStrata — *Principal Stratification Analysis for Data with Post-Randomization Intervention*

Description

Perform pincipal stratification analysis when there are confounding variables after randomization

Usage

```
PStrata(
  PSobject = NULL,
  S.formula,
  Y.formula,
  Y.family,
  data = NULL,
  strata = NULL,
  ER = NULL,
  prior_intercept = prior_flat(),
  prior_coefficient = prior_normal(),
  prior_sigma = prior_inv_gamma(),
  prior_alpha = prior_inv_gamma(),
  prior_lambda = prior_inv_gamma(),
  prior_theta = prior_normal(),
  survival.time.points = 50,
  filename = NULL,
  ...
)
```

Arguments

PSobject — an object of class PSObject. If left blank, the object is constructed using the following arguments. See PSObject for details.

`S.formula`, `Y.formula`
: an object of class "`PSFormula`" (or an object of class "`formula`" that can be coerced to that class with `data` provided) specifying the model for principal stratum and outcome respectively. See `PSFormula` for details.

`Y.family`
: an object of class "`family`": specifying the parametric family of the model for the response and the link function. See the documentation for `glm` for details on how such model fitting takes place. Supported families and corresponding link functions are presented in 'Details' below.

`data`
: (optional) a data frame object. This is required when either `S.formula` or `Y.formula` is a `formula` object, to coerce it into a `PSFormula` object. When this happens, the data frame should contain all of the variables with names given in `S.formula` or `Y.formula`.

`strata`, `ER`
: arguments to define the principal strata. See `PStrataInfo` for details.
Alternatively, one can pass an object of class `PStrataInfo` to `strata`, and `ER` will be ignored.

`prior_intercept`, `prior_coefficient`, `prior_sigma`, `prior_alpha`, `prior_lambda`, `prior_theta`
: prior distribution for corresponding parameters in the model.

`survival.time.points`
: a vector of time points at which the estimated survival probability is evaluated (only used when the type of outcome is survival), or an integer specifying the number of time points to be chosen. By default, the time points are chosen with equal distance from 0 to the 90% quantile of the observed outcome.

`filename`
: (optional) string. If not `NULL`, the stan file will be saved via `cat` in a text file named after the string supplied.

`...`
: additional parameters to be passed into `PSSample`.

Value

An object of class `PStrata` or `PStrata_survival`, which is a list containing

`PSobject`
: An object of `PSObject`.

`post_samples`
: An object of class `rstan::stanfit` returned by **Stan**.

Examples

```
require(abind)
PSobj <- PSObject(
  S.formula = Z + D ~ 1,
  Y.formula = Y ~ 1,
  Y.family = gaussian("identity"),
  data = sim_data_normal,
  strata = c(n = "00*", c = "01", a = "11*")
)

PStrata(PSobj, cores = 2, chains = 2, iter = 200)

# Another example for survival data
```

```
PSobj <- PSObject(
  S.formula = Z + D ~ 1,
  Y.formula = Y + delta ~ 1,
  Y.family = survival("Cox"),
  data = sim_data_Cox,
  strata = c(`never-taker` = "00*", complier = "01", `always-taker` = "11*")
)

PStrata(PSobj, cores = 2, chains = 2, iter = 200)
```

`PStrataInfo` *Create an object that defines the principal strata*

Description

`PStrataInfo` is a class of object that defines all principal strata to be considered, by specifying the potential value of each post-randomization confounding variable under each treatment arm.

Usage

```
PStrataInfo(strata, ER = NULL)
```

Arguments

`strata`	a list or a vector defining all principal strata. Details of the syntax are given in 'Details' below.
`ER`	a vector indicating on which strata exclusion restriction is assumed. Details are given in 'Details' below.

Details

Since definition of the principal strata appears fundamental and essential in principal stratification analyses, the creation of such an object is designed to be user-friendly - various ways are accommodated to create a `PStrataInfo` object, some possibly preferable over others under different settings.

There are mainly two ways to easily create a `PStrataInfo` object.

By string: To define the principal strata by strings, the `strata` argument should receive a named vector, each component being the description of one strata with the name of that strata. The naming does not affect the actual inference, but informative names can be helpful for users to distinguish among strata.

Each stratum is defined by the potential values of the post-randomization confounding variable D under each treatment arm. By convention, assume that the K treatment arms are numbered from 0 to K-1. Then, each stratum is defined by the tuple $(D(0), \ldots, D(K-1))$, which can be written compactly as a string. For example, under binary treatment, the never-takers (i.e. $D(0) = D(1) = 0$) can be represented by string `"00"` and the compliers (i.e. $D(0) = 0, D(1) = 1$) can be represented by string `"01"`. Note that the value that the post-randomization confounding variable

can take is limited between 0 to 9 for the string to be parsed correctly. This should be more than enough in most of the applications, and in cases where a number above 10 is needed, please create the `PStrataInfo` object by matrix (see below).

When multiple post-randomization confounding variables exist, the string for each confounding variable is concatenated with the symbol "`|`". For example, if D_0 and D_1 are both binary post-randomization confounding variables, the stratum defined by $D_0(0) = D_0(1) = 0, D_1(0) = 0, D_1(1) = 1$ can be represented by string `"00|11"`. The order of these confounding variables should be the same as they appear in the `S.formula` parameter in `PSObject`.

A common assumption in practice is the exclusion restriction (ER) assumption, which assumes that the causal effect of the treatment on the outcome is totally realized through the post-randomization confounding variables. For example, the ER assumption on the stratum of never-takers can be interpreted as the outcome is identically distributed across the treated and control group, because all causal effect of the treatment is realized through the post-randomization variable, which is the same (0) under both treatment arms. To assume ER for some stratum, simply put an asterisk "`*`" at the end of the string, such as "00*" for the never-taker stratum. *Note that under the context of multiple post-randomization variables, the package treats all such variables as a unity. The outcome is assumed to be identical under different treatment arms only when all post-randomization variables remain the same under these treatment arms.*

Another way to specify the stratum where ER is assumed is to use the `ER` argument. It either takes a logical vector of the same length of `strata` with `TRUE` indicating ER is assumed and `FALSE` otherwise, or takes a character vector with the names of all strata where ER is to be assumed upon. When names to the strata are not provided in `strata`, the strata can be referred to by their canonical name, which is the string used to define the stratum with asterisks removed. For example, the strata `"00|11*"` can be referred to with name "00|11".

By matrix: To define the principal strata by matrices, the `strata` argument should receive a named list, each component being a matrix. The number of rows matches the number of post-randomization variables, and the number of columns matches that of possible treatment arms. For any fixed row i, column j stores the potential value of the i-th post-randomization variable under treatment arm j.

When this approach is used, there is no shorthand to specify ER assumption. The `ER` argument is required to do this.

Warning: When ER assumption is specified in both `strata` and `ER` argument, the shorthand notation for ER in `strata` is ignored, and a warning is given regardless of whether the specification given by `strata` and `ER` actually match.

Value

an object of class `PSStrataInfo`, which is a list of the following components.

num_strata number of principal strata defined

num_treatment number of treatment arms

num_postrand_var number of post-randomization variables

max_postrand_level integer vector, the biggest number used by each post-randomization variable

strata_matrix integer matrix, each row corresponding to one stratum and each column corresponding to one treatment arm. The matrix is designed only for internal use.

ER_list logical vector, each component corresponding to one stratum, indicating whether ER is assumed for the specific stratum

strata_names character vector, the names of all strata

Examples

```
PStrataInfo(strata = c(n = "00*", c = "01", a = "11"))
PStrataInfo(
  strata = list(n = c(0, 0), c = c(0, 1), a = c(1, 1)),
  ER = c(TRUE, FALSE, FALSE)
)
PStrataInfo(
  strata = list(n = c(0, 0), c = c(0, 1), a = c(1, 1)),
  ER = c("n")
)
```

---

sim_data_Cox *Simulated Dataset for Survival Outcome (Cox Model)*

Description

A dataset generated for illustration of the principal stratification analysis. This dataset represents the common case of non-compliance.

Usage

```
sim_data_Cox
```

Format

'sim_data_Cox' A data frame with 1,000 rows and 7 columns:

S Principal Strata: "never taker", "complier" or "always taker"

Z Randomized treatment arm: 0 = control, 1 = treatment

D Actual treatment arm: 0 = control, 1 = treatment

T True outcome: event time

C Censor time

delta Event indicator. 1 means true outcome is observed; 0 means otherwise

Y The observed event time or censor time

Details

The dataset represents the scenario where actual treatment might not be in compliance with the randomized (assigned) treatment. Defiers and always-takers are ruled out, leaving two strata, "never-taker" and "complier" randomly sampled with probability 0.3, 0.7 respectively. The assigned treatment Z is randomized with 0.5 probability for either arm. The true event time T is given by the following Weibull-Cox distribution

never-taker $Y \sim Weibull - Cox(theta = 1, mu = 0.3)$

complier $Y \sim Weibull - Cox(theta = 1, mu = 2 - 0.6 * Z)$

and the censor time C is uniformly drawn between 0.5 and 2.

The exclusion restriction assumption holds for never-takers in this generated dataset.

---

sim_data_normal *Simulated Dataset for Normal Outcome*

Description

A dataset generated for illustration of the principal stratification analysis. This dataset represents the common case of non-compliance.

Usage

```
sim_data_normal
```

Format

'sim_data_normal' A data frame with 1,000 rows and 4 columns:

S Principal Strata: "never taker", "complier" or "always taker"

Z Randomized treatment arm: 0 = control, 1 = treatment

D Actual treatment arm: 0 = control, 1 = treatment

Y Outcome

Details

The dataset represents the scenario where actual treatment might not be in compliance with the randomized (assigned) treatment. Defiers are ruled out, leaving three strata, "never taker", "complier" and "always taker" randomly sampled with probability 0.3, 0.2 and 0.5 respectively. The assigned treatment Z is randomized with 0.5 probability for either arm. The outcome Y is given by the following.

never taker $Y \sim N(3, 1)$

complier $Y \sim N(-1 - Z, 0.5)$

always taker $Y \sim N(1, 2)$

The exclusion restriction assumption holds for never takers and always takers in this generated dataset.

`survival` *The family function for survival data*

Description

Construct a family object for survival data

Usage

```
survival(method = "Cox", link = "identity")
```

Arguments

`method`	the parametric method used for survival data. Can be Cox or AFT.
`link`	a link function, currently only identity is implemented and used

Value

A `family` object

Index