Package ‘estimatr’

April 1, 2024

Type Package

Title Fast Estimators for Design-Based Inference

Version 1.0.4


URL https://declaredesign.org/r/estimatr/,
https://github.com/DeclareDesign/estimatr

BugReports https://github.com/DeclareDesign/estimatr/issues

License MIT + file LICENSE

Depends R (&gt;= 3.6.0)

Imports Formula, generics, methods, Rcpp (&gt;= 0.12.16), rlang (&gt;= 0.2.0)

LinkingTo Rcpp, RcppEigen

Encoding UTF-8

RoxygenNote 7.3.1

LazyData true

Suggests fabricatr (&gt;= 0.10.0), randomizr (&gt;= 0.20.0), AER, clubSandwich, emmeans (&gt;= 1.4), estimability, margins, modelsummary, prediction, RcppEigen, sandwich, stargazer, testthat, car

Enhances texreg

NeedsCompilation yes

Author Graeme Blair [aut, cre],
Jasper Cooper [aut],
Alexander Coppock [aut],
Macartan Humphreys [aut],
Luke Sonnet [aut],

1
Description

Usage
alo_star_men
Format

A data frame with educational treatments and outcomes:

- **gpa0** high school GPA
- **sfsp** financial incentives and support treatment
- **ssp** support only treatment
- **GPA_year1** college GPA year 1
- **GPA_year2** college GPA year 2

Details

This data was originally taken from the following paper, subset to men who showed up to college, were in one of the arms with the support condition, and had GPA data for their first year in college. 


Source

https://www.aeaweb.org/articles?id=10.1257/app.1.1.136

---

**commarobust** | Build lm_robust object from lm fit

**Description**

Build lm_robust object from lm fit

**Usage**

```r
commarobust(model, se_type = NULL, clusters = NULL, ci = TRUE, alpha = 0.05)
```

**Arguments**

- **model**: an lm model object
- **se_type**: The sort of standard error sought. If clusters is not specified the options are "HC0", "HC1" (or "stata", the equivalent), "HC2" (default), "HC3", or "classical". If clusters is specified the options are "CR0", "CR2" (default), or "stata". Can also specify "none", which may speed up estimation of the coefficients.
- **clusters**: A vector corresponding to the clusters in the data.
- **ci**: logical. Whether to compute and return p-values and confidence intervals, TRUE by default.
- **alpha**: The significance level, 0.05 by default.
Value

an `lm_robust` object.

Examples

```r
lmo <- lm(mpg ~ hp, data = mtcars)

# Default HC2
commarobust(lmo)

commarobust(lmo, se_type = "HC3")

commarobust(lmo, se_type = "stata", clusters = mtcars$carb)
```

`declaration_to_condition_pr_mat`

Builds condition probability matrices for Horvitz-Thompson estimation from `randomizr` declaration

Description

Builds condition probability matrices for Horvitz-Thompson estimation from `randomizr` declaration

Usage

```r
declaration_to_condition_pr_mat(
  ra_declaration,
  condition1 = NULL,
  condition2 = NULL,
  prob_matrix = NULL
)
```

Arguments

- `ra_declaration`: An object of class "ra_declaration", generated by the `declare_ra` function in `randomizr`. This object contains the experimental design that will be represented in a condition probability matrix.
- `condition1`: The name of the first condition, often the control group. If NULL, defaults to first condition in randomizr declaration. Either both `condition1` and `condition2` have to be specified or both left as NULL.
- `condition2`: The name of the second condition, often the treatment group. If NULL, defaults to second condition in randomizr declaration. Either both `condition1` and `condition2` have to be specified or both left as NULL.
- `prob_matrix`: An optional probability matrix to override the one in `ra_declaration`
Details

This function takes a "ra_declaration", generated by the declare_ra function in randomizr and returns a 2n*2n matrix that can be used to fully specify the design for horvitz_thompson estimation. This is done by passing this matrix to the condition_pr_mat argument of horvitz_thompson.

Currently, this function can learn the condition probability matrix for a wide variety of randomizations: simple, complete, simple clustered, complete clustered, blocked, block-clustered.

A condition probability matrix is made up of four submatrices, each of which corresponds to the joint and marginal probability that each observation is in one of the two treatment conditions.

The upper-left quadrant is an n*n matrix. On the diagonal is the marginal probability of being in condition 1, often control, for every unit (Pr(Z_i = Condition1) where Z represents the vector of treatment conditions). The off-diagonal elements are the joint probabilities of each unit being in condition 1 with each other unit, Pr(Z_i = Condition1, Z_j = Condition1) where i indexes the rows and j indexes the columns.

The upper-right quadrant is also an n*n matrix. On the diagonal is the joint probability of a unit being in condition 1 and condition 2, often the treatment, and thus is always 0. The off-diagonal elements are the joint probability of unit i being in condition 1 and unit j being in condition 2, Pr(Z_i = Condition1, Z_j = Condition2).

The lower-left quadrant is also an n*n matrix. On the diagonal is the joint probability of a unit being in condition 1 and condition 2, and thus is always 0. The off-diagonal elements are the joint probability of unit i being in condition 2 and unit j being in condition 1, Pr(Z_i = Condition2, Z_j = Condition1).

The lower-right quadrant is an n*n matrix. On the diagonal is the marginal probability of being in condition 2, often treatment, for every unit (Pr(Z_i = Condition2)). The off-diagonal elements are the joint probability of each unit being in condition 2 together, Pr(Z_i = Condition2, Z_j = Condition2).

Value

a numeric 2n*2n matrix of marginal and joint condition treatment probabilities to be passed to the condition_pr_mat argument of horvitz_thompson. See details.

See Also

permutations_to_condition_pr_mat

Examples

# Learn condition probability matrix from complete blocked design
library(randomizr)
n <- 100
dat <- data.frame(
  blocks = sample(letters[1:10], size = n, replace = TRUE),
  y = rnorm(n)
)

# Declare complete blocked randomization
bl_declaration <- declare_ra(blocks = dat$blocks, prob = 0.4, simple = FALSE)
# Get probabilities
block_pr_mat <- declaration_to_condition_pr_mat(bl_declaration, 0, 1)
# Do randomization
dat$z <- conduct_ra(bl_declaration)

horvitz_thompson(y ~ z, data = dat, condition_pr_mat = block_pr_mat)

# When you pass a declaration to horvitz_thompson, this function is called
# Equivalent to above call
horvitz_thompson(y ~ z, data = dat, ra_declaration = bl_declaration)

difference_in_means  

**Design-based difference-in-means estimator**

**Description**

Difference-in-means estimators that selects the appropriate point estimate, standard errors, and degrees of freedom for a variety of designs: unit randomized, cluster randomized, block randomized, block-cluster randomized, matched-pairs, and matched-pair cluster randomized designs.

**Usage**

difference_in_means(
  formula,
  data,
  blocks,
  clusters,
  weights,
  subset,
  se_type = c("default", "none"),
  condition1 = NULL,
  condition2 = NULL,
  ci = TRUE,
  alpha = 0.05
)

**Arguments**

- **formula**: an object of class formula, as in `lm`, such as `Y ~ Z` with only one variable on the right-hand side, the treatment.
- **data**: A data frame.
- **blocks**: An optional bare (unquoted) name of the block variable. Use for blocked designs only.
- **clusters**: An optional bare (unquoted) name of the variable that corresponds to the clusters in the data; used for cluster randomized designs. For blocked designs, clusters must nest within blocks.
weights  the bare (unquoted) names of the weights variable in the supplied data.
subset An optional bare (unquoted) expression specifying a subset of observations to be used.
se_type An optional string that can be one of c("default", "none”). If “default” (the default), it will use the default standard error estimator for the design, and if "none" then standard errors will not be computed which may speed up run time if only the point estimate is required.
condition1 value in the treatment vector of the condition to be the control. Effects are estimated with condition1 as the control and condition2 as the treatment. If unspecified, condition1 is the "first" condition and condition2 is the "second" according to levels if the treatment is a factor or according to a sort if it is a numeric or character variable (i.e if unspecified and the treatment is 0s and 1s, condition1 will by default be 0 and condition2 will be 1). See the examples for more.
condition2 value in the treatment vector of the condition to be the treatment. See condition1.
ci logical. Whether to compute and return p-values and confidence intervals, TRUE by default.
alpha The significance level, 0.05 by default.

Details
This function implements a difference-in-means estimator, with support for blocked, clustered, matched-pairs, block-clustered, and matched-pair clustered designs. One specifies their design by passing the blocks and clusters in their data and this function chooses which estimator is most appropriate.

If you pass only blocks, if all blocks are of size two, we will infer that the design is a matched-pairs design. If they are all size four or larger, we will infer that it is a regular blocked design. If you pass both blocks and clusters, we will similarly infer whether it is a matched-pairs clustered design or a block-clustered design the number of clusters per block. If the user passes only clusters, we will infer that the design was cluster-randomized. If the user specifies neither the blocks nor the clusters, a regular Welch’s t-test will be performed.

Importantly, if the user specifies weights, the estimation is handed off to lm_robust with the appropriate robust standard errors as weighted difference-in-means estimators are not implemented here. More details of the about each of the estimators can be found in the mathematical notes.

Value
Returns an object of class "difference_in_means".

The post-estimation commands functions summary and tidy return results in a data.frame. To get useful data out of the return, you can use these data frames, you can use the resulting list directly, or you can use the generic accessor functions coef and confint.

An object of class "difference_in_means" is a list containing at least the following components:

coefficients the estimated difference in means
std.error the estimated standard error
statistic the t-statistic
difference_in_means

df the estimated degrees of freedom
p.value the p-value from a two-sided t-test using coefficients, std.error, and df
conf.low the lower bound of the 1 - alpha percent confidence interval
conf.high the upper bound of the 1 - alpha percent confidence interval
term a character vector of coefficient names
alpha the significance level specified by the user
N the number of observations used
outcome the name of the outcome variable
design the name of the design learned from the arguments passed

References


See Also

lm_lin

Examples

library(fabricatr)
library(randomizr)
# Get appropriate standard errors for unit-randomized designs

# ----------
# 1. Unit randomized
# ----------
dat <- fabricate(
  N = 100,
  Y = rnorm(100),
  Z_comp = complete_ra(N, prob = 0.4),
)
table(dat$Z_comp)
difference_in_means(Y ~ Z_comp, data = dat)

# ----------
# 2. Cluster randomized
# ----------
# Accurates estimates and standard errors for clustered designs
dat$clust <- sample(20, size = nrow(dat), replace = TRUE)
dat$Z_clust <- cluster_ra(dat$clust, prob = 0.6)
```r
difference_in_means

table(dat$Z_clust, dat$clust)
summary(difference_in_means(Y ~ Z_clust, clusters = clust, data = dat))

# 3. Block randomized

dat$block <- rep(1:10, each = 10)
dat$Z_block <- block_ra(dat$block, prob = 0.5)
table(dat$Z_block, dat$block)
difference_in_means(Y ~ Z_block, blocks = block, data = dat)

# 4. Block cluster randomized

dat$small_clust <- rep(1:50, each = 2)
dat$big_blocks <- rep(1:5, each = 10)
dat$Z_blcl <- block_and_cluster_ra(
  blocks = dat$big_blocks,
  clusters = dat$small_clust
)
difference_in_means(
  Y ~ Z_blcl,
  blocks = big_blocks,
  clusters = small_clust,
  data = dat
)

# 5. Matched-pairs

dat$pairs <- rep(1:50, each = 2)
dat$Z_pairs <- block_ra(dat$pairs, prob = 0.5)
table(dat$pairs, dat$Z_pairs)
difference_in_means(Y ~ Z_pairs, blocks = pairs, data = dat)

# 6. Matched-pair cluster randomized

dat$small_clust <- rep(1:50, each = 2)
dat$cluster_pairs <- rep(1:25, each = 4)
table(dat$cluster_pairs, dat$small_clust)
dat$Z_mpcl <- block_and_cluster_ra(
  blocks = dat$cluster_pairs,
  clusters = dat$small_clust
)
```
clusters = dat$small_clust
)

difference_in_means(
  Y ~ Z_mpc1,
  blocks = cluster_pairs,
  clusters = small_clust,
  data = dat
)

# Other examples
# ------------
# Also works with multi-valued treatments if users specify
# comparison of interest
dat$Z_multi <- simple_ra(
  nrow(dat),
  conditions = c("Treatment 2", "Treatment 1", "Control"),
  prob_each = c(0.4, 0.4, 0.2)
)

difference_in_means(
  Y ~ Z_multi,
  condition1 = "Treatment 2",
  condition2 = "Control",
  data = dat
)
difference_in_means(
  Y ~ Z_multi,
  condition1 = "Treatment 1",
  condition2 = "Control",
  data = dat
)

# Specifying weights will result in estimation via lm_robust()
dat$w <- runif(nrow(dat))
difference_in_means(Y ~ Z_comp, weights = w, data = dat)
lm_robust(Y ~ Z_comp, weights = w, data = dat)

---

Description

Fast procedures for small set of commonly-used, design-appropriate estimators with robust standard errors and confidence intervals. Includes estimators for linear regression, instrumental variables regression, difference-in-means, Horvitz-Thompson estimation, and regression improving precision.

**Author(s)**

**Maintainer:** Graeme Blair <graeme.blair@gmail.com>

**Authors:**
- Jasper Cooper <jjc2247@columbia.edu>
- Alexander Coppock <alex.coppock@yale.edu>
- Macartan Humphreys <macartan@gmail.com>
- Luke Sonnet <luke.sonnet@gmail.com>

**Other contributors:**
- Neal Fultz <nfultz@gmail.com> [contributor]
- Lily Medina <lilymiru@gmail.com> [contributor]
- Russell Lenth <russell-lenth@uiowa.edu> [contributor]

**See Also**

Useful links:
- [https://declaredesign.org/r/estimatr/](https://declaredesign.org/r/estimatr/)
- [https://github.com/DeclareDesign/estimatr](https://github.com/DeclareDesign/estimatr)

---

### estimatr_glancers

**Glance at an estimatr object**

**Description**

Glance at an estimatr object

**Usage**

```r
## S3 method for class 'lm_robust'
glance(x, ...)

## S3 method for class 'lh_robust'
glance(x, ...)

## S3 method for class 'iv_robust'
glance(x, ...)

## S3 method for class 'difference_in_means'
```
```r
# S3 method for class 'horvitz_thompson'
glance(x, ...)
```

### Arguments

- **x**
  - An object returned by one of the estimators
- **...**
  - extra arguments (not used)

### Value

For `glance.lm_robust`, a data.frame with columns:

- **r.squared**
  - the $R^2$
  
  $R^2 = 1 - \frac{\text{Sum}(e[i]^2)}{\text{Sum}((y[i] - y^*)^2)}$,

  where $y^*$ is the mean of $y[i]$ if there is an intercept and zero otherwise, and $e[i]$ is the $i$th residual.

- **adj.r.squared**
  - the $R^2$ but penalized for having more parameters

- **se_type**
  - the standard error type specified by the user

- **statistic**
  - the value of the F-statistic

- **p.value**
  - p-value from the F test

- **df.residual**
  - residual degrees of freedom

- **nobs**
  - the number of observations used

For `glance.lh_robust`, we glance the `lm_robust` component only. You can access the linear hypotheses as a data.frame directly from the `lh` component of the `lh_robust` object.

For `glance.iv_robust`, a data.frame with columns:

- **r.squared**
  - The $R^2$ of the second stage regression

- **adj.r.squared**
  - The $R^2$ but penalized for having more parameters

- **df.residual**
  - residual degrees of freedom

- **N**
  - the number of observations used

- **se_type**
  - the standard error type specified by the user

- **statistic**
  - the value of the F-statistic

- **p.value**
  - p-value from the F test

- **statistic.weakinst**
  - the value of the first stage F-statistic, useful for the weak instruments test; only reported if there is only one endogenous variable

- **p.value.weakinst**
  - p-value from the first-stage F test, a test of weak instruments; only reported if there is only one endogenous variable

- **statistic.endogeneity**
  - the value of the F-statistic for the test of endogeneity; often called the Wu-Hausman statistic, with robust standard errors, we employ the regression based test
```
p.value.endogeneity
   p-value from the F-test for endogeneity
statistic.overid
   the value of the chi-squared statistic for the test of instrument correlation with
   the error term; only reported with overidentification
p.value.overid  p-value from the chi-squared test; only reported with overidentification

For glance.difference_in_means, a data.frame with columns:
design        the design used, and therefore the estimator used
df            the degrees of freedom
nobs          the number of observations used
nbblocks      the number of blocks, if used
nclusters     the number of clusters, if used
condition2    the second, "treatment", condition
condition1    the first, "control", condition

For glance.horvitz_thompson, a data.frame with columns:
nobs          the number of observations used
se_type       the type of standard error estimator used
condition2    the second, "treatment", condition
condition1    the first, "control", condition

See Also
generics::glance(), lm_robust(), lm_lin(), iv_robust(), difference_in_means(), horvitz_thompson()

estimatr_tidiers  Tidy an estimatr object

Description
Tidy an estimatr object

Usage
## S3 method for class 'lm_robust'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

## S3 method for class 'iv_robust'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

## S3 method for class 'difference_in_means'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)
## S3 method for class 'horvitz_thompson'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

## S3 method for class 'lh_robust'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

## S3 method for class 'lh'
tidy(x, conf.int = TRUE, conf.level = NULL, ...)

### Arguments

- **x**
  - An object returned by one of the estimators

- **conf.int**
  - Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to ‘TRUE’.

- **conf.level**
  - The confidence level to use for the confidence interval if ‘conf.int = TRUE’. Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.

- **...**
  - extra arguments (not used)

### Value

A data.frame with columns for coefficient names, estimates, standard errors, confidence intervals, p-values, degrees of freedom, and the name of the outcome variable

### See Also

- `generics::tidy()`, `lm_robust()`, `iv_robust()`, `difference_in_means()`, `horvitz_thompson()`

---

### extract.robust_default

Extract model data for texreg package

### Description

Prepares a "lm_robust" or "iv_robust" object for the texreg package. This is largely a clone of the extract.lm method.

### Usage

```
extract.robust_default(
  model,
  include.ci = TRUE,
  include.rsquared = TRUE,
  include.adjrs = TRUE,
  include.nobs = TRUE,
  include.fstatistic = FALSE,
)```
extract-default

```r
include.rmse = TRUE,
include.nclusts = TRUE,
...
)

extract.lm_robust(
  model,
  include.ci = TRUE,
  include.rsquared = TRUE,
  include.adjrs = TRUE,
  include.nobs = TRUE,
  include.fstatistic = FALSE,
  include.rmse = TRUE,
  include.nclusts = TRUE,
  ...
)

extract.iv_robust(
  model,
  include.ci = TRUE,
  include.rsquared = TRUE,
  include.adjrs = TRUE,
  include.nobs = TRUE,
  include.fstatistic = FALSE,
  include.rmse = TRUE,
  include.nclusts = TRUE,
  ...
)
```

**Arguments**

- `model`: an object of class `lm_robust` or `iv_robust`
- `include.ci`: logical. Defaults to TRUE
- `include.rsquared`: logical. Defaults to TRUE
- `include.adjrs`: logical. Defaults to TRUE
- `include.nobs`: logical. Defaults to TRUE
- `include.fstatistic`: logical. Defaults to TRUE
- `include.rmse`: logical. Defaults to TRUE
- `include.nclusts`: logical. Defaults to TRUE if clusters in model
- `...`: unused
gen_pr_matrix_cluster  Generate condition probability matrix given clusters and probabilities

Description
Generate condition probability matrix given clusters and probabilities

Usage
gen_pr_matrix_cluster(clusters, treat_probs, simple)

Arguments
clusters  A vector of clusters
treat_probs  A vector of treatment (condition 2) probabilities
simple  A boolean for whether the assignment is a random sample assignment (TRUE, default) or complete random assignment (FALSE)

Value
A numeric 2n*2n matrix of marginal and joint condition treatment probabilities to be passed to the condition_pr_mat argument of horvitz_thompson.

See Also
declaration_to_condition_pr_mat

horvitz_thompson  Horvitz-Thompson estimator for two-armed trials

Description
Horvitz-Thompson estimators that are unbiased for designs in which the randomization scheme is known

Usage
horvitz_thompson(
  formula,
data,
blocks,
clusters,
simple = NULL,
condition_prs,
condition_pr_mat = NULL,
ra_declaration = NULL,
subset,
condition1 = NULL,
condition2 = NULL,
se_type = c("youngs", "constant", "none"),
    ci = TRUE,
    alpha = 0.05,
    return_condition_pr_mat = FALSE
)

Arguments

formula  an object of class formula, as in `lm`, such as `Y ~ Z` with only one variable on the
right-hand side, the treatment.
data      A data.frame.
blocks    An optional bare (unquoted) name of the block variable. Use for blocked designs only. See details.
clusters  An optional bare (unquoted) name of the variable that corresponds to the clusters in the data; used for cluster randomized designs. For blocked designs, clusters
must be within blocks.
simple    logical, optional. Whether the randomization is simple (TRUE) or complete (FALSE). This is ignored if blocks are specified, as all blocked designs use complete
randomization, or either ra_declaration or condition_pr_mat are passed. Otherwise, it defaults to TRUE.
condition_prs An optional bare (unquoted) name of the variable with the condition 2 (treatment) probabilities. See details. May also use a single number for the condition 2
probability if it is constant.
condition_pr_mat An optional 2n * 2n matrix of marginal and joint probabilities of all units in
condition1 and condition2. See details.
ra_declaration An object of class "ra_declaration", from the declare_ra function in the
randomizr package. This is the third way that one can specify a design for this estimator. Cannot be used along with any of condition_prs, blocks, clusters, or condition_pr_mat. See details.
subset    An optional bare (unquoted) expression specifying a subset of observations to be used.
condition1 value in the treatment vector of the condition to be the control. Effects are estimated with condition1 as the control and condition2 as the treatment. If
unspecified, condition1 is the "first" condition and condition2 is the "second" according to levels if the treatment is a factor or according to a sort if it is a numeric or character variable (i.e if unspecified and the treatment is 0s and 1s, condition1 will by default be 0 and condition2 will be 1). See the examples for more.
condition2 value in the treatment vector of the condition to be the treatment. See condition1.
se_type can be one of c("younsg", "constant", "none") and corresponds the estimator of the standard errors. Default estimator uses Young’s inequality (and is conservative) while the other uses a constant treatment effects assumption and only works for simple randomized designs at the moment. If "none" then standard errors will not be computed which may speed up run time if only the point estimate is required.

ci logical. Whether to compute and return p-values and confidence intervals, TRUE by default.

alpha The significance level, 0.05 by default.

return_condition_pr_mat logical. Whether to return the condition probability matrix. Returns NULL if the design is simple randomization, FALSE by default.

Details

This function implements the Horvitz-Thompson estimator for treatment effects for two-armed trials. This estimator is useful for estimating unbiased treatment effects given any randomization scheme as long as the randomization scheme is known.

In short, the Horvitz-Thompson estimator essentially reweights each unit by the probability of it being in its observed condition. Pivotal to the estimation of treatment effects using this estimator are the marginal condition probabilities (i.e., the probability that any one unit is in a particular treatment condition). Pivotal to estimating the variance whenever the design is more complicated than simple randomization are the joint condition probabilities (i.e., the probabilities that any two units have a particular set of treatment conditions, either the same or different). The estimator we provide here considers the case with two treatment conditions.

Users interested in more details can see the mathematical notes for more information and references, or see the references below.

There are three distinct ways that users can specify the design to the function. The preferred way is to use the declare_ra function in the randomizr package. This function takes several arguments, including blocks, clusters, treatment probabilities, whether randomization is simple or not, and more. Passing the outcome of that function, an object of class "ra_declaration" to the ra_declaration argument in this function, will lead to a call of the declaration_to_condition_pr_mat function which generates the condition probability matrix needed to estimate treatment effects and standard errors. We provide many examples below of how this could be done.

The second way is to pass the names of vectors in your data to condition_prs, blocks, and clusters. You can further specify whether the randomization was simple or complete using the simple argument. Note that if blocks are specified the randomization is always treated as complete. From these vectors, the function learns how to build the condition probability matrix that is used in estimation.

In the case where condition_prs is specified, this function assumes those probabilities are the marginal probability that each unit is in condition2 and then uses the other arguments (blocks, clusters, simple) to learn the rest of the design. If users do not pass condition_prs, this function learns the probability of being in condition2 from the data. That is, none of these arguments are specified, we assume that there was a simple randomization where the probability of each unit being in condition2 was the average of all units in condition2. Similarly, we learn the block-level probability of treatment within blocks by looking at the mean number of units in condition2 if condition_prs is not specified.
The third way is to pass a `condition_pr_mat` directly. One can see more about this object in the documentation for `declaration_to_condition_pr_mat` and `permutations_to_condition_pr_mat`. Essentially, this $2n \times 2n$ matrix allows users to specify marginal and joint marginal probabilities of units being in conditions 1 and 2 of arbitrary complexity. Users should only use this option if they are certain they know what they are doing.

**Value**

Returns an object of class "horvitz_thompson".

The post-estimation commands functions `summary` and `tidy` return results in a `data.frame`. To get useful data out of the return, you can use these data frames, you can use the resulting list directly, or you can use the generic accessor functions `coef` and `confint`.

An object of class "horvitz_thompson" is a list containing at least the following components:

- `coefficients` the estimated difference in totals
- `std.error` the estimated standard error
- `statistic` the z-statistic
- `df` the estimated degrees of freedom
- `p.value` the p-value from a two-sided z-test using `coefficients` and `std.error`
- `conf.low` the lower bound of the $1 - \alpha$ percent confidence interval
- `conf.high` the upper bound of the $1 - \alpha$ percent confidence interval
- `term` a character vector of coefficient names
- `alpha` the significance level specified by the user
- `nobs` the number of observations used
- `outcome` the name of the outcome variable
- `condition_pr_mat` the condition probability matrix if `return_condition_pr_mat` is TRUE

**References**


**See Also**

`declare_ra`
Examples

```r
# Set seed
set.seed(42)

# Simulate data
n <- 10
dat <- data.frame(y = rnorm(n))

library(randomizr)

#----------
# 1. Simple random assignment
#----------

dat$p <- 0.5
dat$z <- rbinom(n, size = 1, prob = dat$p)

# If you only pass condition_prs, we assume simple random sampling
horvitz_thompson(y ~ z, data = dat, condition_prs = p)
# Assume constant effects instead
horvitz_thompson(y ~ z, data = dat, condition_prs = p, se_type = "constant")

# Also can use randomizr to pass a declaration
srs_declaration <- declare_ra(N = nrow(dat), prob = 0.5, simple = TRUE)
horvitz_thompson(y ~ z, data = dat, ra_declaration = srs_declaration)

#----------
# 2. Complete random assignment
#----------

dat$z <- sample(rep(0:1, each = n/2))
# Can use a declaration
crs_declaration <- declare_ra(N = nrow(dat), prob = 0.5, simple = FALSE)
horvitz_thompson(y ~ z, data = dat, ra_declaration = crs_declaration)
# Can precompute condition_pr_mat and pass it
# (faster for multiple runs with same condition probability matrix)
crs_pr_mat <- declaration_to_condition_pr_mat(crs_declaration)
horvitz_thompson(y ~ z, data = dat, condition_pr_mat = crs_pr_mat)

#----------
# 3. Clustered treatment, complete random assignment
#-----------

# Simulating data
dat$cl <- rep(1:4, times = c(2, 3, 3))
dat$prob <- 0.5
clust_crs_decl <- declare_ra(N = nrow(dat), clusters = dat$cl, prob = 0.5)
dat$z <- conduct_ra(clust_crs_decl)
# Easiest to specify using declaration
ht_cl <- horvitz_thompson(y ~ z, data = dat, ra_declaration = clust_crs_decl)
# Also can pass the condition probability and the clusters
ht_cl_manual <- horvitz_thompson(
  y ~ z,
  condition_prs = dat$prob,
  ra_declaration = clust_crs_decl)
```

---

**horvitz_thompson**
Two-Stage Least Squares Instrumental Variables Regression

Description

This function estimates an instrumental variables regression using two-stage least squares with a variety of options for robust standard errors.

Usage

```r
iv_robust(
  formula,
  data,
  weights,
  subset,
  clusters,
  fixed_effects,
  se_type = NULL,
  ci = TRUE,
  alpha = 0.05,
  diagnostics = FALSE,
  return_vcov = TRUE,
  try_cholesky = FALSE
)
```
Arguments

- **formula**: an object of class formula of the regression and the instruments. For example, the formula \( y \sim x_1 + x_2 \mid z_1 + z_2 \) specifies \( x_1 \) and \( x_2 \) as endogenous regressors and \( z_1 \) and \( z_2 \) as their respective instruments.

- **data**: A data.frame

- **weights**: the bare (unquoted) names of the weights variable in the supplied data.

- **subset**: An optional bare (unquoted) expression specifying a subset of observations to be used.

- **clusters**: An optional bare (unquoted) name of the variable that corresponds to the clusters in the data.

- **fixed_effects**: An optional right-sided formula containing the fixed effects that will be projected out of the data, such as \( \sim \text{blockID} \). Do not pass multiple fixed effects with intersecting groups. Speed gains are greatest for variables with large numbers of groups and when using "HC1" or "stata" standard errors. See "Details".

- **se_type**: The sort of standard error sought. If **clusters** is not specified the options are "HC0", "HC1" (or "stata", the equivalent), "HC2" (default), "HC3", or "classical". If **clusters** is specified the options are "CR0", "CR2" (default), or "stata". Can also specify "none", which may speed up estimation of the coefficients.

- **ci**: logical. Whether to compute and return p-values and confidence intervals, TRUE by default.

- **alpha**: The significance level, 0.05 by default.

- **diagnostics**: logical. Whether to compute and return instrumental variable diagnostic statistics and tests.

- **return_vcov**: logical. Whether to return the variance-covariance matrix for later usage, TRUE by default.

- **try_cholesky**: logical. Whether to try using a Cholesky decomposition to solve least squares instead of a QR decomposition, FALSE by default. Using a Cholesky decomposition may result in speed gains, but should only be used if users are sure their model is full-rank (i.e., there is no perfect multi-collinearity)

Details

This function performs two-stage least squares estimation to fit instrumental variables regression. The syntax is similar to that in \texttt{ivreg} from the \texttt{AER} package. Regressors and instruments should be specified in a two-part formula, such as \( y \sim x_1 + x_2 \mid z_1 + z_2 + z_3 \), where \( x_1 \) and \( x_2 \) are regressors and \( z_1 \), \( z_2 \), and \( z_3 \) are instruments. Unlike \texttt{ivreg}, you must explicitly specify all exogenous regressors on both sides of the bar.

The default variance estimators are the same as in \texttt{lm_robust}. Without clusters, we default to HC2 standard errors, and with clusters we default to CR2 standard errors. 2SLS variance estimates are computed using the same estimators as in \texttt{lm_robust}, however the design matrix used are the second-stage regressors, which includes the estimated endogenous regressors, and the residuals used are the difference between the outcome and a fit produced by the second-stage coefficients and the first-stage (endogenous) regressors. More notes on this can be found at the mathematical appendix.
If fixed effects are specified, both the outcome, regressors, and instruments are centered using the method of alternating projections (Halperin 1962; Gaure 2013). Specifying fixed effects in this way will result in large speed gains with standard error estimators that do not need to invert the matrix of fixed effects. This means using "classical", "HC0", "HC1", "CR0", or "stata" standard errors will be faster than other standard error estimators. Be wary when specifying fixed effects that may result in perfect fits for some observations or if there are intersecting groups across multiple fixed effect variables (e.g. if you specify both "year" and "country" fixed effects with an unbalanced panel where one year you only have data for one country).

If diagnostics are requested, we compute and return three sets of diagnostics. First, we return tests for weak instruments using first-stage F-statistics (diagnostic_first_stage_fstatistic). Specifically, the F-statistics reported compare the model regressing each endogeneous variable on both the included exogenous variables and the instruments to a model where each endogenous variable is regressed only on the included exogenous variables (without the instruments). A significant F-test for weak instruments provides evidence against the null hypothesis that the instruments are weak. Second, we return tests for the endogeneity of the endogenous variables, often called the Wu-Hausman test (diagnostic_endogeneity_test). We implement the regression test from Hausman (1978), which allows for robust variance estimation. A significant endogeneity test provides evidence against the null that all the variables are exogenous. Third, we return a test for the correlation between the instruments and the error term (diagnostic_overid_test). We implement the Wooldridge (1995) robust score test, which is identical to Sargan’s (1958) test with classical standard errors. This test is only reported if the model is overidentified (i.e. the number of instruments is greater than the number of endogenous regressors), and if no weights are specified.

Value

An object of class "iv_robust".

The post-estimation commands functions summary and tidy return results in a data.frame. To get useful data out of the return, you can use these data frames, you can use the resulting list directly, or you can use the generic accessor functions coef, vcov, confint, and predict.

An object of class "iv_robust" is a list containing at least the following components:

- coefficients: the estimated coefficients
- std.error: the estimated standard errors
- statistic: the t-statistic
- df: the estimated degrees of freedom
- p.value: the p-values from a two-sided t-test using coefficients, std.error, and df
- conf.low: the lower bound of the 1 - alpha percent confidence interval
- conf.high: the upper bound of the 1 - alpha percent confidence interval
- term: a character vector of coefficient names
- alpha: the significance level specified by the user
- se_type: the standard error type specified by the user
- res_var: the residual variance
- nobs: the number of observations used
- k: the number of columns in the design matrix (includes linearly dependent columns!)
rank the rank of the fitted model
vcov the fitted variance covariance matrix
r.squared the $R^2$ of the second stage regression
adj.r.squared the $R^2$ of the second stage regression, but penalized for having more parameters.
rank
fstatistic a vector with the value of the second stage F-statistic with the numerator and denominator degrees of freedom
firststage_fstatistic a vector with the value of the first stage F-statistic with the numerator and denominator degrees of freedom, useful for a test for weak instruments
weighted whether or not weights were applied
call the original function call
fitted.values the matrix of predicted means

We also return terms with the second stage terms and terms_regressors with the first stage terms, both of which used by predict. If fixed_effects are specified, then we return proj_fstatistic, proj_r.squared, and proj_adj.r.squared, which are model fit statistics that are computed on the projected model (after demeaning the fixed effects).

We also return various diagnostics when `diagnostics` == TRUE. These are stored in diagnostic_firststage_fstatistic, diagnostic_endogeneity_test, and diagnostic_overid_test. They have the test statistic, relevant degrees of freedom, and p.value in a named vector. See 'Details' for more. These are printed in a formatted table when the model object is passed to summary().

References


Examples

```r
library(fabricatr)
dat <- fabricate(
  N = 40,
  Y = rpois(N, lambda = 4),
  Z = rbinom(N, 1, prob = 0.4),
  D = Z * rbinom(N, 1, prob = 0.8),
  X = rnorm(N),
  G = sample(letters[1:4], N, replace = TRUE)
)

# Instrument for treatment 'D' with encouragement 'Z'
tidy(iv_robust(Y ~ D + X | Z + X, data = dat))

# Instrument with Stata's `ivregress 2sls , small rob` HC1 variance
tidy(iv_robust(Y ~ D | Z, data = dat, se_type = "stata"))
```
# With clusters, we use CR2 errors by default
dat$cl <- rep(letters[1:5], length.out = nrow(dat))
tidy(iv_robust(Y ~ D | Z, data = dat, clusters = cl))

# Again, easy to replicate Stata (again with `small` correction in Stata)
tidy(iv_robust(Y ~ D | Z, data = dat, clusters = cl, se_type = "stata"))

# We can also specify fixed effects, that will be taken as exogenous regressors
# Speed gains with fixed effects are greatest with "stata" or "HC1" std.errors
tidy(iv_robust(Y ~ D | Z, data = dat, fixed_effects = ~ G, se_type = "HC1"))

---

**lh_robust**  
*Linear Hypothesis for Ordinary Least Squares with Robust Standard Errors*

**Description**
This function fits a linear model with robust standard errors and performs linear hypothesis test.

**Usage**

```r
lh_robust(..., data, linear_hypothesis)
```

**Arguments**

- `...` Other arguments to be passed to `lm_robust`
- `data` A `data.frame`
- `linear_hypothesis` A character string or a matrix specifying combination, to be passed to the hypothesis.matrix argument of `car::linearHypothesis` See `linearHypothesis` for more details.

**Details**
This function is a wrapper for `lm_robust` and for `linearHypothesis`. It first runs `lm_robust` and next passes "lm_robust" object as an argument to `linearHypothesis`.

**Value**

An object of class "lh_robust" containing the two following components:

- `lm_robust` an object as returned by `lm_robust`.
- `lh` A data frame with most of its columns pulled from `linearHypothesis`' output.

The only analysis directly performed by `lh_robust` is a t-test for the null hypothesis of no effects of the linear combination of coefficients as specified by the user. All other output components are either extracted from `linearHypothesis` or `lm_robust`.

The original output returned by `linearHypothesis` is added as an attribute under the "linear_hypothesis" attribute.
Examples

```r
library(fabricatr)
dat <- fabricate(
  N = 40,
  y = rpois(N, lambda = 4),
  x = rnorm(N),
  z = rbinom(N, 1, prob = 0.4),
  clusterID = sample(1:4, 40, replace = TRUE)
)

# Default variance estimator is HC2 robust standard errors
lhro <- lh_robust(y ~ x + z, data = dat, linear_hypothesis = "z + 2x = 0")

# The linear hypothesis argument can be specified equivalently as:
lh_robust(y ~ x + z, data = dat, linear_hypothesis = "z = 2x")
lh_robust(y ~ x + z, data = dat, linear_hypothesis = "2*x + 1*z")
lh_robust(y ~ x + z, data = dat, linear_hypothesis = "z + 2x = 0")

# Also recovers other sorts of standard errors just as specified in \code{\link{lm_robust}}
lh_robust(y ~ x + z, data = dat, linear_hypothesis = "z + 2x = 0", se_type = "classical")
lh_robust(y ~ x + z, data = dat, linear_hypothesis = "z + 2x = 0", se_type = "HC1")

# Can tidy() main output and subcomponents in to a data.frame
lhro <- lh_robust(y ~ x + z, data = dat, linear_hypothesis = "z + 2x = 0")
tidy(lhro)
tidy(lhro$lm_robust)
tidy(lhro$lh)

# Can use summary() to get more statistics on the main output and subcomponents.
summary(lhro)
summary(lhro$lm_robust)
summary(lhro$lh)
```

---

**lm_lin**

Linear regression with the Lin (2013) covariate adjustment

### Description

This function is a wrapper for `lm_robust` that is useful for estimating treatment effects with pre-treatment covariate data. This implements the method described by Lin (2013).

### Usage

```r
lm_lin(
  formula,
  covariates,
  data,
```
*lm_lin*

weights, subset, clusters, se_type = NULL, ci = TRUE, alpha = 0.05, return_vcov = TRUE, try_cholesky = FALSE)

**Arguments**

- **formula**
  an object of class formula, as in *lm*, such as \(Y \sim Z\) with only one variable on the right-hand side, the treatment

- **covariates**
  a right-sided formula with pre-treatment covariates on the right hand side, such as \(\sim x_1 + x_2 + x_3\).

- **data**
  A data.frame

- **weights**
  the bare (unquoted) names of the weights variable in the supplied data.

- **subset**
  An optional bare (unquoted) expression specifying a subset of observations to be used.

- **clusters**
  An optional bare (unquoted) name of the variable that corresponds to the clusters in the data.

- **se_type**
  The sort of standard error sought. If *clusters* is not specified the options are "HC0", "HC1" (or "stata", the equivalent), "HC2" (default), "HC3", or "classical". If *clusters* is specified the options are "CR0", "CR2" (default), or "stata" are permissible.

- **ci**
  logical. Whether to compute and return p-values and confidence intervals, TRUE by default.

- **alpha**
  The significance level, 0.05 by default.

- **return_vcov**
  logical. Whether to return the variance-covariance matrix for later usage, TRUE by default.

- **try_cholesky**
  logical. Whether to try using a Cholesky decomposition to solve least squares instead of a QR decomposition, FALSE by default. Using a Cholesky decomposition may result in speed gains, but should only be used if users are sure their model is full-rank (i.e., there is no perfect multi-collinearity)

**Details**

This function is simply a wrapper for *lm_robust* and implements the Lin estimator (see the reference below). This method pre-processes the data by taking the covariates specified in the `covariates` argument, centering them by subtracting from each covariate its mean, and interacting them with the treatment. If the treatment has multiple values, a series of dummies for each value is created and each of those is interacted with the demeaned covariates. More details can be found in the Getting Started vignette and the mathematical notes.
Value

An object of class "lm_robust".

The post-estimation commands functions `summary` and `tidy` return results in a data.frame. To get useful data out of the return, you can use these data frames, you can use the resulting list directly, or you can use the generic accessor functions `coef`, `vcov`, `confint`, and `predict`. Marginal effects and uncertainty about them can be gotten by passing this object to `margins` from the `margins` package.

Users who want to print the results in TeX of HTML can use the `extract` function and the `texreg` package.

An object of class "lm_robust" is a list containing at least the following components:

- `coefficients`  
  the estimated coefficients
- `std.error`  
  the estimated standard errors
- `statistic`  
  the t-statistic
- `df`  
  the estimated degrees of freedom
- `p.value`  
  the p-values from a two-sided t-test using `coefficients`, `std.error`, and `df`
- `conf.low`  
  the lower bound of the 1 - alpha percent confidence interval
- `conf.high`  
  the upper bound of the 1 - alpha percent confidence interval
- `term`  
  a character vector of coefficient names
- `alpha`  
  the significance level specified by the user
- `se_type`  
  the standard error type specified by the user
- `res_var`  
  the residual variance
- `N`  
  the number of observations used
- `k`  
  the number of columns in the design matrix (includes linearly dependent columns!)
- `rank`  
  the rank of the fitted model
- `vcov`  
  the fitted variance covariance matrix
- `r.squared`  
  The $R^2$

$$R^2 = 1 - \frac{\text{Sum}(e[i]^2)}{\text{Sum}((y[i] - y^*)^2)},$$

where $y^*$ is the mean of $y[i]$ if there is an intercept and zero otherwise, and $e[i]$ is the $i$th residual.

- `adj.r.squared`  
  The $R^2$ but penalized for having more parameters, `rank`
- `weighted`  
  whether or not weights were applied
- `call`  
  the original function call
- `fitted.values`  
  the matrix of predicted means

We also return `terms` and `contrasts`, used by `predict`, and `scaled_center` (the means of each of the covariates used for centering them).

References


See Also

lm_robust

Examples

library(fabricatr)
library(randomizr)

dat <- fabricate(
    N = 40,
    x = rnorm(N, mean = 2.3),
    x2 = rpois(N, lambda = 2),
    x3 = runif(N),
    y0 = rnorm(N) + x,
    y1 = rnorm(N) + x + 0.35
)

dat$z <- complete_ra(N = nrow(dat))
dat$y <- ifelse(dat$z == 1, dat$y1, dat$y0)

# Same specification as lm_robust() with one additional argument
lmlin_out <- lm_lin(y ~ z, covariates = ~ x, data = dat)
tidy(lmlin_out)

# Works with multiple pre-treatment covariates
lm_lin(y ~ z, covariates = ~ x + x2, data = dat)

# Also centers data AFTER evaluating any functions in formula
lmlin_out2 <- lm_lin(y ~ z, covariates = ~ x + log(x3), data = dat)
lmlin_out2$scaled_center["log(x3)"]
mean(log(dat$x3))

# Works easily with clusters
dat$clusterID <- rep(1:20, each = 2)
dat$z_clust <- cluster_ra(clusters = dat$clusterID)

lm_lin(y ~ z_clust, covariates = ~ x, data = dat, clusters = clusterID)

# Works with multi-valued treatments
dat$z_multi <- sample(1:3, size = nrow(dat), replace = TRUE)

lm_lin(y ~ z_multi, covariates = ~ x, data = dat)

# Stratified estimator with blocks
dat$blockID <- rep(1:5, each = 8)
dat$z_block <- block_ra(blocks = dat$blockID)

lm_lin(y ~ z_block, ~ factor(blockID), data = dat)

## Not run:
# Can also use 'margins' package if you have it installed to get
# marginal effects
library(margins)
lmlout <- lm_lin(y ~ z_block, ~ x, data = dat)
summary(margins(lmlout))

# Can output results using 'texreg'
library(texreg)
texregobj <- extract(lmlout)

## End(Not run)

---

**lm_robust**

*Ordinary Least Squares with Robust Standard Errors*

**Description**

This formula fits a linear model, provides a variety of options for robust standard errors, and conducts coefficient tests.

**Usage**

```r
lm_robust(
  formula,
  data,
  weights,
  subset,
  clusters,
  fixed_effects,
  se_type = NULL,
  ci = TRUE,
  alpha = 0.05,
  return_vcov = TRUE,
  try_cholesky = FALSE
)
```

**Arguments**

- `formula`: an object of class formula, as in `lm`
- `data`: A `data.frame`
- `weights`: the bare (unquoted) names of the weights variable in the supplied data.
- `subset`: An optional bare (unquoted) expression specifying a subset of observations to be used.
- `clusters`: An optional bare (unquoted) name of the variable that corresponds to the clusters in the data.
- `fixed_effects`: An optional right-sided formula containing the fixed effects that will be projected out of the data, such as `~ blockID`. Do not pass multiple fixed effects with intersecting groups. Speed gains are greatest for variables with large numbers of groups and when using "HC1" or "stata" standard errors. See 'Details'.
se_type  The sort of standard error sought. If clusters is not specified the options are "HC0", "HC1" (or "stata", the equivalent), "HC2" (default), "HC3", or "classical". If clusters is specified the options are "CR0", "CR2" (default), or "stata". Can also specify "none", which may speed up estimation of the coefficients.

ci  logical. Whether to compute and return p-values and confidence intervals, TRUE by default.

alpha  The significance level, 0.05 by default.

return_vcov  logical. Whether to return the variance-covariance matrix for later usage, TRUE by default.

try_cholesky  logical. Whether to try using a Cholesky decomposition to solve least squares instead of a QR decomposition, FALSE by default. Using a Cholesky decomposition may result in speed gains, but should only be used if users are sure their model is full-rank (i.e., there is no perfect multi-collinearity)

Details

This function performs linear regression and provides a variety of standard errors. It takes a formula and data much in the same was as lm does, and all auxiliary variables, such as clusters and weights, can be passed either as quoted names of columns, as bare column names, or as a self-contained vector. Examples of usage can be seen below and in the Getting Started vignette.

The mathematical notes in this vignette specify the exact estimators used by this function. The default variance estimators have been chosen largely in accordance with the procedures in this manual. The default for the case without clusters is the HC2 estimator and the default with clusters is the analogous CR2 estimator. Users can easily replicate Stata standard errors in the clustered or non-clustered case by setting `se_type = "stata"`.

The function estimates the coefficients and standard errors in C++, using the RcppEigen package. By default, we estimate the coefficients using Column-Pivoting QR decomposition from the Eigen C++ library, although users could get faster solutions by setting `try_cholesky = TRUE` to use a Cholesky decomposition instead. This will likely result in quicker solutions, but the algorithm does not reliably detect when there are linear dependencies in the model and may fail silently if they exist.

If `fixed_effects` are specified, both the outcome and design matrix are centered using the method of alternating projections (Halperin 1962; Gaure 2013). Specifying fixed effects in this way will result in large speed gains with standard error estimators that do not need to invert the matrix of fixed effects. This means using "classical", "HC0", "HC1", "CR0", or "stata" standard errors will be faster than other standard error estimators. Be wary when specifying fixed effects that may result in perfect fits for some observations or if there are intersecting groups across multiple fixed effect variables (e.g. if you specify both "year" and "country" fixed effects with an unbalanced panel where one year you only have data for one country).

As with `lm()`, multivariate regression (multiple outcomes) will only admit observations into the estimation that have no missingness on any outcome.

Value

An object of class "lm_robust".

The post-estimation commands functions summary and tidy return results in a data.frame. To get useful data out of the return, you can use these data frames, you can use the resulting list directly,
or you can use the generic accessor functions `coef`, `vcov`, `confint`, and `predict`. Marginal effects and uncertainty about them can be gotten by passing this object to `margins` from the `margins`, or to `emmeans` in the `emmeans` package.

Users who want to print the results in TeX of HTML can use the `extract` function and the `texreg` package.

If users specify a multivariate linear regression model (multiple outcomes), then some of the below components will be of higher dimension to accommodate the additional models.

An object of class "lm_robust" is a list containing at least the following components:

- `coefficients` the estimated coefficients
- `std.error` the estimated standard errors
- `statistic` the t-statistic
- `df` the estimated degrees of freedom
- `p.value` the p-values from a two-sided t-test using `coefficients`, `std.error`, and `df`
- `conf.low` the lower bound of the $1 - \alpha$ percent confidence interval
- `conf.high` the upper bound of the $1 - \alpha$ percent confidence interval
- `term` a character vector of coefficient names
- `alpha` the significance level specified by the user
- `se_type` the standard error type specified by the user
- `res_var` the residual variance
- `N` the number of observations used
- `k` the number of columns in the design matrix (includes linearly dependent columns!)
- `rank` the rank of the fitted model
- `vcov` the fitted variance covariance matrix
- `r.squared` The $R^2$,
  \[ R^2 = 1 - \frac{\text{Sum}(e[i]^2)}{\text{Sum}((y[i] - y^*)^2)}, \]
  where $y^*$ is the mean of $y[i]$ if there is an intercept and zero otherwise, and $e[i]$ is the ith residual.
- `adj.r.squared` The $R^2$ but penalized for having more parameters, `rank`
- `fstatistic` a vector with the value of the F-statistic with the numerator and denominator degrees of freedom
- `weighted` whether or not weights were applied
- `call` the original function call
- `fitted.values` the matrix of predicted means

We also return `terms` and `contrasts`, used by `predict`. If `fixed_effects` are specified, then we return `proj_fstatistic`, `proj_r.squared`, and `proj_adj.r.squared`, which are model fit statistics that are computed on the projected model (after demeaning the fixed effects).
References


Examples

```r
set.seed(15)
library(fabricatr)

dat <- fabricate(
  N = 40,
  y = rpois(N, lambda = 4),
  x = rnorm(N),
  z = rbinom(N, 1, prob = 0.4)
)

# Default variance estimator is HC2 robust standard errors
lmro <- lm_robust(y ~ x + z, data = dat)

# Can tidy() the data in to a data.frame
tidy(lmro)
# Can use summary() to get more statistics
summary(lmro)
# Can also get coefficients three ways
lmro$coefficients
tidy(lmro)$estimate
# Can also get confidence intervals from object or with new 1 - `alpha`
lmro$conf.low
confint(lmro, level = 0.8)

# Can recover classical standard errors
lmclassic <- lm_robust(y ~ x + z, data = dat, se_type = "classical")
```
tidy(lmclassic)

# Can easily match Stata's robust standard errors
lmstata <- lm_robust(y ~ x + z, data = dat, se_type = "stata")
tidy(lmstata)

# Easy to specify clusters for cluster-robust inference
dat$clusterID <- sample(1:10, size = 40, replace = TRUE)

lmclust <- lm_robust(y ~ x + z, data = dat, clusters = clusterID)
tidy(lmclust)

# Can also match Stata's clustered standard errors
lm_robust(
  y ~ x + z,
  data = dat,
  clusters = clusterID,
  se_type = "stata"
)

# Works just as LM does with functions in the formula
dat$blockID <- rep(c("A", "B", "C", "D"), each = 10)

lm_robust(y ~ x + z + factor(blockID), data = dat)

# Weights are also easily specified
dat$w <- runif(40)

lm_robust(
  y ~ x + z,
  data = dat,
  weights = w,
  clusters = clusterID
)

# Subsetting works just as in 'lm()'
lm_robust(y ~ x, data = dat, subset = z == 1)

# One can also choose to set the significance level for different CIs
lm_robust(y ~ x + z, data = dat, alpha = 0.1)

# We can also specify fixed effects
# Speed gains with fixed effects are greatest with "stata" or "HC1" std.errors
tidy(lm_robust(y ~ z, data = dat, fixed_effects = ~ blockID, se_type = "HC1"))

## Not run:
# Can also use 'margins' or 'emmeans' package if you have them installed
# to get marginal effects
library(margins)
lmrout <- lm_robust(y ~ x + z, data = dat)
summary(margins(lmrout))

# Can output results using 'texreg'
library(texreg)
texreg(lmrout)

# Using emmeans to obtain covariate-adjusted means
library(emmeans)
fiber.rlm <- lm_robust(strength ~ diameter + machine, data = fiber)
emmeans(fiber.rlm, "machine")

## End(Not run)

### lm_robust_fit

Internal method that creates linear fits

**Description**

Internal method that creates linear fits

**Usage**

```r
lm_robust_fit(
  y,
  X,
  yoriginal = NULL,
  Xoriginal = NULL,
  weights,
  cluster,
  fixed_effects = NULL,
  ci = TRUE,
  se_type,
  has_int,
  alpha = 0.05,
  return_vcov = TRUE,
  return_fit = TRUE,
  try_cholesky = FALSE,
  iv_stage = list(0)
)
```

**Arguments**

- `y` numeric outcome vector
- `X` numeric design matrix
- `yoriginal` numeric outcome vector, unprojected if there are fixed effects
- `Xoriginal` numeric design matrix, unprojected if there are fixed effects. Any column named "(Intercept)" will be dropped
- `weights` numeric weights vector
cluster  numeric cluster vector
fixed_effects  character matrix of fixed effect groups
ci  boolean that when T returns confidence intervals and p-values
se_type  character denoting which kind of SEs to return
has_int  logical, whether the model has an intercept, used for $R^2$
alpha  numeric denoting the test size for confidence intervals
return_vcov  logical, whether to return the vcov matrix for later usage
return_fit  logical, whether to return fitted values
try_cholesky  logical, whether to try using a cholesky decomposition to solve LS instead of a QR decomposition
iv_stage  list of length two, the first element denotes the stage of 2SLS IV estimation, where 0 is used for OLS. The second element is only used for the second stage of 2SLS and has the first stage design matrix. For OLS, the default, list(0), for the first stage of 2SLS list(1), for second stage of 2SLS list(2, first_stage_design_mat).

na.omit_detailed.data.frame

Extra logging on na.omit handler

Description

Extra logging on na.omit handler

Usage

na.omit_detailed.data.frame(object)

Arguments

object  a data.frame

Value

a normal omit object, with the extra attribute why.omit, which contains the leftmost column containing an NA for each row that was dropped, by column name, if any were dropped.

See Also

na.omit
permutations_to_condition_pr_mat

Builds condition probability matrices for Horvitz-Thompson estimation from permutation matrix

Description

Builds condition probability matrices for Horvitz-Thompson estimation from permutation matrix

Usage

permutations_to_condition_pr_mat(permutations)

Arguments

permutations A matrix where the rows are units and the columns are different treatment permutations; treated units must be represented with a 1 and control units with a 0

Details

This function takes a matrix of permutations, for example from the obtain_permutation_matrix function in randomizr or through simulation and returns a 2n*2n matrix that can be used to fully specify the design for horvitz_thompson estimation. You can read more about these matrices in the documentation for the declaration_to_condition_pr_mat function.

This is done by passing this matrix to the condition_pr_mat argument of horvitz_thompson.

Value

a numeric 2n*2n matrix of marginal and joint condition treatment probabilities to be passed to the condition_pr_mat argument of horvitz_thompson.

See Also

declare_ra, declaration_to_condition_pr_mat

Examples

# Complete randomization
perms <- replicate(1000, sample(rep(0:1, each = 50)))
comp_pr_mat <- permutations_to_condition_pr_mat(perms)

# Arbitrary randomization
possible_treats <- cbind(
  c(1, 1, 0, 1, 0, 0, 1, 1, 0),
  c(0, 1, 1, 0, 1, 0, 1, 0, 1),
  c(1, 0, 1, 1, 1, 1, 0, 0, 0)
arb_pr_mat <- permutations_to_condition_pr_mat(possible_treats)
# Simulating a column to be realized treatment
z <- possible_treats[, sample(ncol(possible_treats), size = 1)]
y <- rnorm(nrow(possible_treats))
horvitz_thompson(y - z, condition_pr_mat = arb_pr_mat)

predict.lm_robust

**Predict method for lm_robust object**

**Description**

Predict method for lm_robust object

**Usage**

```r
## S3 method for class 'lm_robust'
predict(
  object,
  newdata,
  se.fit = FALSE,
  interval = c("none", "confidence", "prediction"),
  alpha = 0.05,
  na.action = na.pass,
  pred.var = NULL,
  weights,
  ...
)
```

**Arguments**

- `object` an object of class 'lm_robust'
- `newdata` a data frame in which to look for variables with which to predict
- `se.fit` logical. Whether standard errors are required, default = FALSE
- `interval` type of interval calculation. Can be abbreviated, default = none
- `alpha` numeric denoting the test size for confidence intervals
- `na.action` function determining what should be done with missing values in newdata. The default is to predict NA.
- `pred.var` the variance(s) for future observations to be assumed for prediction intervals.
- `weights` variance weights for prediction. This can be a numeric vector or a bare (unquoted) name of the weights variable in the supplied newdata.
- `...` other arguments, unused
Details

Produces predicted values, obtained by evaluating the regression function in the frame newdata for fits from `lm_robust` and `lm_lin`. If the logical se.fit is TRUE, standard errors of the predictions are calculated. Setting intervals specifies computation of confidence or prediction (tolerance) intervals at the specified level, sometimes referred to as narrow vs. wide intervals.

The equation used for the standard error of a prediction given a row of data x is:

$$\sqrt{x^T \Sigma x'},$$

where $\Sigma$ is the estimated variance-covariance matrix from `lm_robust`.

The prediction intervals are for a single observation at each case in newdata with error variance(s) `pred.var`. The default is to assume that future observations have the same error variance as those used for fitting, which is gotten from the fit `lm_robust` object. If weights is supplied, the inverse of this is used as a scale factor. If the fit was weighted, the default is to assume constant prediction variance, with a warning.

Examples

```r
# Set seed
set.seed(42)

# Simulate data
n <- 10
dat <- data.frame(y = rnorm(n), x = rnorm(n))

# Fit lm
lm_out <- lm_robust(y ~ x, data = dat)
# Get predicted fits
fits <- predict(lm_out, newdata = dat)
# With standard errors and confidence intervals
fits <- predict(lm_out, newdata = dat, se.fit = TRUE, interval = "confidence")

# Use new data as well
new_dat <- data.frame(x = runif(n, 5, 8))
predict(lm_out, newdata = new_dat)

# You can also supply custom variance weights for prediction intervals
new_dat$w <- runif(n)
predict(lm_out, newdata = new_dat, weights = w, interval = "prediction")
```

---

**starprep**

Prepare model fits for stargazer

**Description**

Prepare model fits for stargazer
Usage

starprep(
  ..., 
  stat = c("std.error", "statistic", "p.value", "ci", "df"), 
  se_type = NULL, 
  clusters = NULL, 
  alpha = 0.05
)

Arguments

... a list of lm_robust or lm objects
stat either "std.error" (the default), "statistic" (the t-statistic), "p.value", "ci", or "df"
se_type (optional) if any of the objects are lm objects, what standard errors should be used. Must only be one type and will be used for all lm objects passed to starprep. See commarobust for more.
clusters (optional) if any of the objects are lm objects, what clusters should be used, if clusters should be used. Must only be one vector and will be used for all lm objects passed to starprep. See commarobust for more.
alpha (optional) if any of the objects are lm objects, what significance level should be used for the p-values or confidence intervals

Details

Used to help extract statistics from lists of model fits for stargazer. Prefers lm_robust objects, but because stargazer does not work with lm_robust objects, starprep can also take lm objects and calls commarobust to get the preferred, robust statistics.

Value

a list of vectors of extracted statistics for stargazers

Examples

library(stargazer)

lm1 <- lm(mpg ~ hp, data = mtcars)
lm2 <- lm(mpg ~ hp + wt, data = mtcars)

# Use default "HC2" standard errors
stargazer(lm1, lm2, 
  se = starprep(lm1, lm2), 
  p = starprep(lm1, lm2, stat = "p.value"), 
  omit.stat = "f")

# NB: We remove the F-stat because stargazer only can use original F-stat
# which uses classical SEs

# Use default "CR2" standard errors with clusters
stargazer(lm1, lm2,
    se = starprep(lm1, lm2, clusters = mtcars$carb),
    p = starprep(lm1, lm2, clusters = mtcars$carb, stat = "p.value"),
    omit.stat = "f")

# Can also specify significance levels and different standard errors
stargazer(lm1, lm2,
    ci.custom = starprep(lm1, lm2, se_type = "HC3", alpha = 0.1, stat = "ci"),
    omit.stat = "f")
Index

* datasets
  * alo_star_men, 2

* estimatr glancers
  * estimatr_glancers, 11

* estimatr tidiers
  * estimatr_tidiers, 13

alo_star_men, 2
commarobust, 3
declaration_to_condition_pr_mat, 4, 16, 18, 19, 37
declare_ra, 4, 5, 17–19, 37
difference_in_means, 6
difference_in_means(), 13, 14

estimatr, 10
estimatr-package (estimatr), 10
estimatr_glancers, 11
estimatr_tidiers, 13
extract, 28, 32
extract.iv_robust
  (extract.robust_default), 14
extract.lm_robust
  (extract.robust_default), 14
extract.robust_default, 14

gen_pr_matrix_cluster, 16
generics::glance(), 13
generics::tidy(), 14
glance.difference_in_means
  (estimatr_glancers), 11
glance.horvitz_thompson
  (estimatr_glancers), 11
glance.iv_robust(estimatr_glancers), 11
glance.lh_robust(estimatr_glancers), 11
glance.lm_robust(estimatr_glancers), 11

horvitz_thompson, 5, 16, 16, 37
horvitz_thompson(), 13, 14

iv_robust, 21
iv_robust(), 13, 14

lh_robust, 25
linearHypothesis, 25
lm, 6, 17, 27, 30, 31
lm_lin, 8, 26
lm_lin(), 13
lm_robust, 4, 7, 15, 22, 25–27, 29, 30, 39
lm_robust(), 13, 14
lm_robust_fit, 35

margins, 28, 32

na.omit, 36
na.omit_detailed.data.frame, 36

obtain_permutation_matrix, 37

permutations_to_condition_pr_mat, 5, 19, 37

predict.lm_robust, 38

starprep, 39

tidy, 7, 19, 23, 28, 31
tidy.difference_in_means
  (estimatr_tidiers), 13
tidy.horvitz_thompson
  (estimatr_tidiers), 13
tidy.iv_robust (estimatr_tidiers), 13
tidy.lh (estimatr_tidiers), 13
tidy.lh_robust (estimatr_tidiers), 13
tidy.lm_robust (estimatr_tidiers), 13