Package ‘EpiForsk’

March 30, 2026

Title Code Sharing at the Department of Epidemiology Research at Statens Serum Institut

Version 0.2.2

Description This is a collection of assorted functions and examples collected from various projects. Currently we have functionalities for simplifying overlapping time intervals, Charlson comorbidity score constructors for Danish data, getting frequency for multiple variables, getting standardized output from logistic and log-linear regressions, sibling design linear regression functionalities a method for calculating the confidence intervals for functions of parameters from a GLM, Bayes equivalent for hypothesis testing with asymptotic Bayes factor, and several help functions for generalized random forest analysis using 'grf'.

URL https://github.com/Laksafoss/EpiForsk

BugReports https://github.com/Laksafoss/EpiForsk/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports broom, cowplot, data.table, dplyr, forcats, ggplot2, glue, grf, gridExtra, Hmisc, MatchIt, methods, nnet, patchwork, policytree, progressr, purrr, rlang, stringr, survey, survival, svyVGAM, tidyr, VGAM

Depends R (>= 4.2)

LazyData true

Suggests cli, CVXR, furrr, future, ggsci, knitr, parallel, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Anders Husby [aut] (ORCID: <https://orcid.org/0000-0002-7634-8455>), Anna Laksafoss [aut] (ORCID: <https://orcid.org/0000-0002-9898-2924>), Emilia Myrup Thiesson [aut] (ORCID:

<https://orcid.org/0000-0001-6258-4177>),
Kim Daniel Jakobsen [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0086-9980>),
Mikael Andersson [aut] (ORCID: <https://orcid.org/0000-0002-0114-2057>),
Klaus Rostgaard [aut] (ORCID: <https://orcid.org/0000-0001-6220-9414>)

Maintainer Kim Daniel Jakobsen <kija@ssi.dk>

Repository CRAN

Date/Publication 2026-03-30 09:30:37 UTC

Contents

`EpiForsk-package` *EpiForsk*

Description

This is a collection of assorted functions and examples collected from various projects. Currently we have functionalities for simplifying overlapping time intervals, Charlson comorbidity score constructors for Danish data, getting frequency for multiple variables, getting standardized output from logistic and log-linear regressions, sibling design linear regression functionalities a method for calculating the confidence intervals for functions of parameters from a GLM, Bayes equivalent for hypothesis testing with asymptotic Bayes factor, and several help functions for generalized random forest analysis using the grf package.

Author(s)

Maintainer: Kim Daniel Jakobsen `<kija@ssi.dk>` (ORCID)

Authors:

- Anders Husby `<andh@ssi.dk>` (ORCID)
- Anna Laksafoss `<adls@ssi.dk>` (ORCID)
- Emilia Myrup Thiesson `<emth@ssi.dk>` (ORCID)
- Mikael Andersson `<aso@ssi.dk>` (ORCID)
- Klaus Rostgaard `<klp@ssi.dk>` (ORCID)

See Also

Useful links:

- `https://github.com/Laksafoss/EpiForsk`
- Report bugs at `https://github.com/Laksafoss/EpiForsk/issues`

`.datatable.aware` *make package data table aware*

Description

This package uses data.table as a fast alternative to dplyr in cases where performance is essential.

Usage

```
.datatable.aware
```

Format

An object of class `logical` of length 1.

`adls_timevarying_region_data` *Simulated Time-Varying Residence Data*

Description

A dataset of simulated time-varying residence and gender data.

Usage

```
adls_timevarying_region_data
```

Format

`andh_forest_data`**:**

A data frame with 546 rows and 7 columns describing 100 people:

id an id number

dob date of birth

region region of Denmark

move_in date of moving to region

move_out date of moving away from region

gender gender of the person

claim whether or not the person made a claim here

`andh_forest_data` *Example Data for Husby's Forest Plot Vignette*

Description

A data example for the construction of a multi faceted forest plot.

Usage

```
andh_forest_data
```

Format

`andh_forest_data`**:**

A data frame with 18 rows and 12 columns:

type text formattaing, bold/plain

indent number of indents in final formatting

text description text

A_est point estimate in first figure column

A_l lower limit of confidence interval in first figure column
A_u upper limit of confidence interval in first figure column
B_est point estimate in second figure column
B_l lower limit of confidence interval in second figure column
B_u upper limit of confidence interval in second figure column
C_est point estimate in third figure column
C_l lower limit of confidence interval in third figure column
C_u upper limit of confidence interval in third figure column

---

`braid_rows` *Bind lists of list of multiple data frames by row*

Description

Row binds the matching innermost data frames in a list of lists. This is essentially a list inversion `purrr::list_transpose()` with row-binding `dplyr::bind_rows()`

Usage

```
braid_rows(list)
```

Arguments

`list`	A list of lists of `data.frames` where matching innermost elements must be bound together row-wise.

Value

A list of `data.frames` with the combined information from the inputted list.

Examples

```
# A simple example
lst <- lapply(1:5, function(x) {
  list(
    "A" = data.frame("first" = x, "second" = rnorm(x)),
    "B" = data.frame("info" = 1, "other" = 3)
  )
})
braid_rows(lst)

# An example with an additional layer and jagged innermost info
lapply(c(28, 186, 35), function(len) {
  lapply(c("a", "b"), function(x) {
    res <- list(
      "descriptive" = data.frame(
         risk = len,
         event = x,
        var = 1,
        other = 2
      ),
      "results" = data.frame(
        risk = len,
        event = x,
        important = 4:7,
        new = 3:6
      )
    )
    if (len < 30) {
      res <- c(res, list("additional" = data.frame(helps = "extra data")))
    }
    return(res)
  }) |> braid_rows()
}) |> braid_rows()
```

`CATESurface` *Calculate CATE on a surface in the covariate space*

Description

Calculates CATE estimates from a causal forest object on a specified surface within the covariate space.

Usage

```
CATESurface(
  forest,
  continuous_covariates,
  discrete_covariates,
  estimate_variance = TRUE,
  grid = 100,
  fixed_covariate_fct = median,
  other_discrete = NULL,
  max_predict_size = 1e+05,
  num_threads = 2
)
```

Arguments

`forest`	An object of class `causal_forest`, as returned by causal_forest(). Alternatively, and object of class `regression_forest`, as returned by regression_forest().
`continuous_covariates`	character, continuous covariates to use for the surface. Must match names in `forest$X.orig`.

`discrete_covariates`
: character, discrete covariates to use for the surface. Note that discrete covariates are currently assumed to be one-hot encoded with columns named `{fct_nm}_{lvl_nm}`. Names supplied to discrete_covariates should match `fct_nm`.

`estimate_variance`
: boolean, If `TRUE`, the variance of CATE estimates is computed.

`grid`
: list, points in which to predict CATE along continuous covariates. Index i in the list should contain a numeric vectors with either a single integer, specifying the number of equally spaced points within the range of the i'th continuous covariate in which to calculate the CATE, or a numeric vector with manually specified points in which to calculate the CATE along the i'th continuous covariate. If all elements of grid specify a number of points, this can be supplied using a numeric vector. If the list is named, the names must match the continuous covariates. grid will be reordered to match the order of continuous_covariates.

`fixed_covariate_fct`
: Function applied to covariates not in the sub-surface which returns the fixed value of the covariate used to calculate the CATE. Must be specified in one of the following ways:

  - A named function, e.g. `mean`.
  - An anonymous function, e.g. `\(x) x + 1` or `function(x) x + 1`.
  - A formula, e.g. `~ .x + 1`. You must use `.x` to refer to the first argument. Only recommended if you require backward compatibility with older versions of R.
  - A string, integer, or list, e.g. `"idx"`, `1`, or `list("idx", 1)` which are shorthand for `\(x) purrr::pluck(x, "idx")`, `\(x) purrr::pluck(x, 1)`, and `\(x) purrr::pluck(x, "idx", 1)` respectively. Optionally supply `.default` to set a default value if the indexed element is `NULL` or does not exist.

`other_discrete`
: A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr) with columns `covs` and `lvl`. Used to specify the level of each discrete covariate to use when calculating the CATE. assumes the use of one-hot encoding. `covs` must contain the name of discrete covariates, and `lvl` the level to use. Set to `NULL` if none of the fixed covariates are discrete using one-hot-encoding.

`max_predict_size`
: integer, maximum number of examples to predict at a time. If the surface has more points than max_predict_size, the prediction is split up into an appropriate number of chunks.

`num_threads`
: Number of threads used in training. If set to `NULL`, the software automatically selects an appropriate amount.

Value

Tibble with the predicted CATE's on the specified surface in the covariate space. The tibble has columns for each covariate used to train the input forest, as well as columns output from predict.causal_forest().

Author(s)

KIJA

Examples

```
n <- 1000
p <- 3
X <- matrix(rnorm(n * p), n, p) |> as.data.frame()
X_d <- data.frame(
  X_d1 = factor(sample(1:3, n, replace = TRUE)),
  X_d2 = factor(sample(1:3, n, replace = TRUE))
)
X_d <- DiscreteCovariatesToOneHot(X_d)
X <- cbind(X, X_d)
W <- rbinom(n, 1, 0.5)
event_prob <- 1 / (1 + exp(2 * (pmax(2 * X[, 1], 0) * W - X[, 2])))
Y <- rbinom(n, 1, event_prob)
cf <- grf::causal_forest(X, Y, W)
cate_surface <- CATESurface(
  cf,
  continuous_covariates = paste0("V", 1:2),
  discrete_covariates = "X_d1",
  grid = list(
    V1 = 10,
    V2 = -5:5
  ),
  other_discrete = data.frame(
    covs = "X_d2",
    lvl = "4"
  )
)
```

`CausalForestDynamicSubgroups`
Calculate CATE in dynamically determined subgroups

Description

Determines subgroups ranked by CATE estimates from a causal_forest object, then calculates comparable CATE estimates in each subgroup and tests for differences.

Usage

```
CausalForestDynamicSubgroups(forest, n_rankings = 3, n_folds = 5, ...)
```

Arguments

`forest`	An object of class `causal_forest`, as returned by causal_forest().
`n_rankings`	Integer, scalar with number of groups to rank CATE's into.
`n_folds`	Integer, scalar with number of folds to split data into.
`...`	Additional arguments passed to causal_forest() and regression_forest().

Details

To evaluate heterogeneity in treatment effect one can split data into groups by estimated CATE (for an alternative, see also RATEOmnibusTest). To compare estimates one must use a model which is not trained on the subjects we wish to compare. To achieve this, data is partitioned into n_folds folds and a causal forest is trained for each fold where the fold is left out. If the data has no existing clustering, one causal_forest() is trained with the folds as clustering structure. This enables predictions on each fold where trees using data from the fold are left out for the prediction. In the case of preexisting clustering in the data, folds are sampled within each cluster and combined across clusters afterwards.

Value

A list with elements

- forest_subgroups: A tibble with CATE estimates, ranking, and AIPW-scores for each subject.
- forest_rank_ate: A tibble with the ATE estimate and standard error of each subgroup.
- forest_rank_diff_test: A tibble with estimates of the difference in ATE between subgroups and p-values for a formal test of no difference.
- heatmap_data: A tibble with data used to draw a heatmap of covariate distribution in each subgroup.
- forest_rank_ate_plot: ggplot with the ATE estimates in each subgroup.
- heatmap: ggplot with heatmap of covariate distribution in each subgroup.

Author(s)

KIJA

Examples

```
n <- 800
p <- 3
X <- matrix(rnorm(n * p), n, p) |> as.data.frame()
W <- rbinom(n, 1, 0.5)
event_prob <- 1 / (1 + exp(2 * (pmax(2 * X[, 1], 0) * W - X[, 2])))
Y <- rbinom(n, 1, event_prob)
cf <- grf::causal_forest(X, Y, W)
cf_ds <- CausalForestDynamicSubgroups(cf, 2, 4)
```

`ceiling_dec` *Round numbers up to a given number of decimal places*

Description

Round numbers up to a given number of decimal places

Usage

```
ceiling_dec(x, digits = 1)
```

Arguments

`x`	a numeric vector
`digits`	integer indicating the number of decimal places

Value

The rounded up numeric vector

`CForBenefit` *c-for-benefit*

Description

Calculates the c-for-benefit, as proposed by D. van Klaveren et al. (2018), by matching patients based on patient characteristics.

Usage

```
CForBenefit(
  forest,
  match = c("covariates", "CATE"),
  match_method = "nearest",
  match_distance = "mahalanobis",
  tau_hat_method = c("risk_diff", "tau_avg"),
  CI = c("simple", "bootstrap", "none"),
  level = 0.95,
  n_bootstraps = 999L,
  time_limit = Inf,
  time_limit_CI = Inf,
  verbose = TRUE,
  Y = NULL,
  W = NULL,
  X = NULL,
  p_0 = NULL,
  p_1 = NULL,
  tau_hat = NULL,
  ...
)
```

Arguments

`forest`	An object of class `causal_forest`, as returned by causal_forest().
`match`	character, `"covariates"` to match on covariates or `"CATE"` to match on estimated CATE.
`match_method`	see matchit.
`match_distance`	see matchit.
`tau_hat_method`	character, `"risk_diff"` to calculate the expected treatment effect in matched groups as the risk under treatment for the treated subject minus the risk under control for the untreated subject. `"tau_avg"` to calculate it as the average treatment effect of matched subject.
`CI`	character, `"none"` for no confidence interval, `"simple"` to use a normal approximation, and `"bootstrap"` to use the bootstrap.
`level`	numeric, confidence level of the confidence interval.
`n_bootstraps`	numeric, number of bootstraps to use for the bootstrap confidence interval computation.
`time_limit`	numeric, maximum allowed time to compute C-for-benefit. If limit is reached, execution stops.
`time_limit_CI`	numeric, maximum time allowed to compute the bootstrap confidence interval. If limit is reached, the user is asked if execution should continue or be stopped.
`verbose`	boolean, TRUE to display progress bar, FALSE to not display progress bar.
`Y`	a vector of outcomes. If provided, replaces `forest$Y.orig`.
`W`	a vector of treatment assignment; 1 for active treatment; 0 for control If provided, replaces `forest$W.orig`.
`X`	a matrix of patient characteristics. If provided, replaces `forest$X.orig`.
`p_0`	a vector of outcome probabilities under control.
`p_1`	a vector of outcome probabilities under active treatment.
`tau_hat`	a vector of individualized treatment effect predictions. If provided, replaces forest$predictions.
`...`	additional arguments for matchit.

Details

The c-for-benefit statistic is inspired by the c-statistic used with prediction models to measure discrimination. The c-statistic takes all pairs of observations discordant on the outcome, and calculates the proportion of these where the subject with the higher predicted probability was the one who observed the outcome. In order to extend this to treatment effects, van Klaveren et al. suggest matching

a treated subject to a control subject on the predicted treatments effect (or alternatively the covariates) and defining the observed effect as the difference between the outcomes of the treated subject and the control subject. The c-for-benefit statistic is then defined as the proportion of matched pairs with unequal observed effect in which the subject pair receiving greater treatment effect also has the highest expected treatment effect.
When calculating the expected treatment effect, van Klaveren et al. use the average CATE from the matched subjects in a pair (tau_hat_method = "mean"). However, this doesn't match the observed effect used, unless the baseline risks are equal. The observed effect is the difference between the observed outcome for the subject receiving treatment and the observed outcome for the subject receiving control. Their outcomes are governed by the exposed risk and the baseline risk respectively. The baseline risks are ideally equal when covariate matching, although instability of the forest estimates can cause significantly different baseline risks due to non-exact matching. When matching on CATE, we should not expect baseline risks to be equal. Instead, we can more closely match the observed treatment effect by using the difference between the exposed risk for the subject receiving treatment and the baseline risk of the subject receiving control (tau_hat_method = "treatment").

Value

a list with the following components:

- type: a list with the input provided to the function which determines how C-for-benefit is computed.
- matched_patients: a tibble containing the matched patients.
- c_for_benefit: the resulting C-for-benefit value.
- lower_CI: the lower bound of the confidence interval (if CI = TRUE).
- upper_CI: the upper bound of the confidence interval (if CI = TRUE).

Author(s)

KIJA

Examples

```
n <- 800
p <- 3
X <- matrix(rnorm(n * p), n, p)
W <- rbinom(n, 1, 0.5)
event_prob <- 1 / (1 + exp(2 * (pmax(2 * X[, 1], 0) * W - X[, 2])))
Y <- rbinom(n, 1, event_prob)
cf <- grf::causal_forest(X, Y, W)
CB_out <- CForBenefit(
forest = cf, CI = "bootstrap", n_bootstraps = 20L, verbose = TRUE,
match_method = "nearest", match_distance = "mahalanobis"
)
```

charlson_score *Charlson Score Constructor*

Description

Charlson comorbidity score for Danish ICD-10 and ICD-8 data. This is a SAS-macro ASO translated to R in March of 2022

Usage

```
charlson_score(
  data,
  Person_ID,
  diagnosis_variable,
  time_variable = NULL,
  end_date = NULL,
  days_before_end_date = NULL,
  amount_output = "total"
)
```

Arguments

`data`	A data.frame with at least an id variable and a variable with all diagnosis codes. The data should be in the long format (only one variable with diagnoses, but several lines per person is OK).
`Person_ID`	<`data-masking`> An unquoted expression naming the id variable in `data`. This variable must always be specified.
`diagnosis_variable`	<`data-masking`> An unquoted expression naming the diagnosis variable in `data`. This variable must always be specified.
`time_variable`	<`data-masking`> An unquoted expression naming the diagnosis time variable in `data` if needed. The `time_variable` must be in a date format. When `time_variable` is specified, `end_date` must also be specified.
`end_date`	<`data-masking`> An unquoted expression naming the end of time-period to search for relevant diagnoses or a single date specifying the end date. If `end_date` names a variable, this variable must be in a date format.
`days_before_end_date`	A numeric specifying the number of days look-back from `end_date` to search for relevant diagnoses.
`amount_output`	A character specifying whether all created index variables should be returned. When `amount_output` is "total" (the default) only the resulting Charlson scores are returned, otherwise all disease- specific index variables are returned.

Details

The `charlson_score()` function calculates the Charlson Charlson Comorbidity Index for each person. Three different variations on the score has been implemented:

- cc: Article from Quan et al. (Coding Algorithms for Defining Comorbidities in ICD-9 and ICD-10 Administrative Data, Med Care 2005:43: 1130-1139), the same HTR and others have used - ICD10 only
- ch: Article from Christensen et al. (Comparison of Charlson comorbidity index with SAPS and APACHE sources for prediction of mortality following intensive care, Clinical Epidemiology 2011:3 203-211), include ICD8 and ICD10 but the included diagnoses are not the same as in Quan
- cd: Article from Sundarajan et al. (New ICD-10 version of Charlson Comorbidity Index predicted in-hospital mortality, Journal of clinical Epidemiology 57 (2004) 1288-1294, include ICD10 = Charlson-Deyo including cancer

Value

If `Person_ID` and `diagnosis_variable` are the only specifications, the function will calculate the different versions of the Charlson score on all data available for each person, regardless of timing etc. This is OK if only relevant records are included.

NOTE

The diagnoses to use in this function at the current state should be either ICD-8, but preferably ICD-10. The ICD-10 codes should start with two letters, where the first one is "D". Furthermore, the code should only have letters and digits (i.e. the form "DA000" not "DA00.0")

Author(s)

ASO & ADLS

Examples

```
# An example dataset

test_data <- data.frame(
  IDs = c(
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    17, 18, 19, 20, 21, 22, 22, 23, 23, 24, 24, 24, 24, 24
  ),
  Diags = c(
    "DZ36", "DZ38", "DZ40", "DZ42", "DC20", "DI252",
    "DP290", "DI71", "DH340", "DG30", "DJ40", "DM353",
    "DK26", "DK700", "DK711", "DE106", "DE112", "DG82",
    "DZ940", "DC80", "DB20", "DK74", "DK704", "DE101",
    "DE102", "DB20", "DK74", "DK704", "DE101", "DE102"
 ),
 time = as.Date(c(
   "2001-01-30", "2004-05-20", "2007-01-02", "2013-12-01",
   "2017-04-30", "2001-01-30", "2004-05-20", "2007-01-02",
   "2013-12-01", "2017-04-30", "2001-01-30", "2004-05-20",
   "2007-01-02", "2013-12-01", "2017-04-30", "2001-01-30",
   "2004-05-20", "2007-01-02", "2013-12-01", "2017-04-30",
   "2001-01-30", "2004-05-20", "2007-01-02", "2013-12-01",
   "2017-04-30", "2001-01-30", "2004-05-20", "2007-01-02",
   "2013-12-01", "2017-04-30"
 )),
 match_date = as.Date(c(
   "2001-10-15", "2005-10-15", "2011-10-15", "2021-10-15",
   "2021-10-15", "2001-10-15", "2005-10-15", "2011-10-15",
   "2021-10-15", "2021-10-15", "2001-10-15", "2005-10-15",
   "2011-10-15", "2021-10-15", "2021-10-15", "2001-10-15",
   "2005-10-15", "2011-10-15", "2021-10-15", "2021-10-15",
   "2001-10-15", "2005-10-15", "2011-10-15", "2021-10-15",
   "2021-10-15", "2001-10-15", "2005-10-15", "2011-10-15",
   "2021-10-15", "2021-10-15"
 ))
)

# Minimal example
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags
)

# Minimal example with all index diagnosis variables
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags,
  amount_output = "all"
)

# Imposing uniform date restrictions to diagnoses
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags,
  time_variable = time,
  end_date = as.Date("2012-01-01")
)

# Imposing differing date restriction to diagnoses
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags,
  time_variable = time,
  end_date = match_date
)
```

```
# Imposing both a start and end to the lookup period for
# relevant diagnoses
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags,
  time_variable = time,
  end_date = match_date,
  days_before_end_date = 365.25
)
```

---

ci_fct *solve optimization problem for CI bounds*

Description

solve optimization problem for each coordinate of f, to obtain the uniform limit.

Usage

```
ci_fct(i, f, xtx_red, beta_hat, which_parm, level, n_grid, k)
```

Arguments

`i`	An index for the point at which to solve for confidence limits.
`f`	A function taking the parameter vector as its single argument, and returning a numeric vector.
`xtx_red`	Reduced form of matrix *X^TX*.
`beta_hat`	Vector of parameter estimates.
`which_parm`	Vector indicating which parameters to include.
`level`	The confidence level required.
`n_grid`	Either `NULL` or an integer vector of length 1 or the number of `TRUE`/indices in which_parm. Specifies the number of grid points in each dimension of a grid with endpoints defined by len. If `NULL` or `0L`, will instead sample k points uniformly on a sphere.
`k`	If n_grid is `NULL` or `0L`, the number of points to sample uniformly from a sphere.

Value

One row tibble with estimate and confidence limits.

Examples

```
1+1
```

`ci_fct_error_handler` *Handle errors returned by ci_fct*

Description

Handle errors returned by ci_fct

Usage

```
ci_fct_error_handler(e, which_parm, env)
```

Arguments

`e`	error returned by ci_fct
`which_parm`	Either a logical vector the same length as the coefficient vector, with `TRUE` indicating a coefficient is used by `f`, or an integer vector with the indices of the coefficients used by `f`.
`env`	environment to assign n_grid and k

Value

returns NULL if no stop command is executed.

Examples

```
1+1
```

`CovariateBalance` *Plots for checking covariate balance in causal forest*

Description

Generate plots showing balance in the covariates before and after propensity score weighting with a causal forest object.

Usage

```
CovariateBalance(
  cf,
  plots = c("all", "Love", "density", "ecdf"),
  balance_table = TRUE,
  covariates = NULL,
  names = NULL,
  factor = NULL,
  treatment_name = "W",
  love_breaks = NULL,
  love_xlim = NULL,
  love_scale_color = NULL,
  cd_nrow = NULL,
  cd_ncol = NULL,
  cd_x_scale_width = NULL,
  cd_bar_width = NULL,
  cd_scale_fill = NULL,
  ec_nrow = NULL,
  ec_ncol = NULL,
  ec_x_scale_width = NULL,
  ec_scale_color = NULL
)
```

Arguments

`cf`	An object of class causal_forest (and inheriting from class grf).
`plots`	Character, `"all"` returns both Love plots and density plots, `"Love"` returns only Love plots, `"density"` returns only density plots.
`balance_table`	Boolean, TRUE to return a table with balance statistics.
`covariates`	A vector to select covariates to show in balance plots. If `cf$X.orig` is an unnamed matrix, use a numeric vector to select variables. Otherwise use a character vector. Names provided in the `names` argument takes priority over existing names in `cf$X.orig`. If discrete covariates have been one-hot encoded using DiscreteCovariatesToOneHot the name of these discrete covariates can be provided in `covariates` to select it and to collect all levels into a bar plot to show the distribution.
`names`	A named character vector. The vector itself should contain covariate names from the causal_forest object, while the names attribute should contain the names to use when plotting. If discrete covariates have been one-hot encoded using DiscreteCovariatesToOneHot, providing just the name of a discrete covariate will modify the name of all levels for plotting. If the vector is unnamed, the provided vector will act as the new covariate names, given in the order of `cf$X_orig`. If `NULL` (the default), the original names are used.
`factor`	A named list with covariates to be converted to factor. Note that one-hot encoded covariates are automatically converted, so need not be specified in the factor argument. Each component of the list must contain the factor levels, using a named vector to supply custom labels.
`treatment_name`	Character, name of treatment.
`love_breaks`	Numeric, breaks used in the plot of absolute standardized mean differences.
`love_xlim`	Numeric, x-limits used in the plot of absolute standardized mean differences.
`love_scale_color`	Function, `scale_color_.` function to use in the plot of absolute standardized mean differences.
`cd_nrow, cd_ncol`	Numeric, the dimensions of the grid to create in covariate distribution plots. If both are `NULL` it will use the same logic as facet_wrap to set the dimensions.

`cd_x_scale_width`	Numeric, the distance between major x-axis tics in the covariate distribution plots. If `NULL`, a width is chosen to display approximately six major tics. If length 1, the same width is used for all covariate plots. If the same length as the number of covariates included, each number is used as the width for different covariates, in the order of the covariates after selection with the tidy-select expression in `covariates`.
`cd_bar_width`	Numeric, the width of the bars in the covariate distribution plots (barplots for categorical variables, histograms for continuous variables). If `NULL`, a width is chosen to display approximately 50 bars in histograms, while 0.9 times the resolution of the data is used in bar plots. If length 1, the same width is used for all covariate plots. This is not recommended if there are both categorical and continuous covariates. If the same length as the number of covariates included, each number is used as the bar width for different covariates, in the order of the covariates after selection with the tidy-select expression in `covariates`.
`cd_scale_fill`	Function, `scale_fill_.` function to use in covariate distribution plots.
`ec_nrow`, `ec_ncol`	Numeric, the dimensions of the grid to create in empirical CDF plots. If both are `NULL` it will use the same logic as facet_wrap to set the dimensions.
`ec_x_scale_width`	Numeric, the distance between major x-axis tics in the empirical CDF plots. If `NULL`, a width is chosen to display approximately six major tics. If length 1, the same width is used for all plots. If the same length as the number of covariates included, each number is used as the width for different covariates, in the order of the covariates after selection with the tidy-select expression in `covariates`.
`ec_scale_color`	Function, `scale_color_.` function to use in empirical CDF plots.

Details

If an unnamed character vector is provided in `names`, it must have length `ncol(cf$X.orig)`. Names of covarates not selected by `covariates` can be set to `NA`. If a named character vector is provided in `names`, all renamed covariates will be kept regardless if they are selected in `covariates`. Thus to select only renamed covariates, `character(0)` can be used in `covariates`. The plot theme can be adjusted using ggplot2 active theme modifiers, see theme_get.

Value

A list with up to five elements:

- love_data: data used to plot the absolute standardized mean differences.
- love: plot object for absolute standardized mean differences.
- cd_data: data used to plot covariate distributions.
- cd_unadjusted: plot of unadjusted covariate distributions in the exposure groups.
- cd_adjusted: plot of adjusted covariate distributions in the exposure groups.

Author(s)

KIJA

Examples

```
n <- 1000
p <- 5
X <- matrix(rnorm(n * p), n, p) |>
as.data.frame() |>
dplyr::bind_cols(
  DiscreteCovariatesToOneHot(
    dplyr::tibble(
      D1 = factor(
        sample(1:3, n, replace = TRUE, prob = c(0.2, 0.3, 0.5)),
        labels = c("first", "second", "third")
      ),
      D2 = factor(
        sample(1:2, n, replace = TRUE, prob = c(0.2, 0.8)),
        labels = c("a", "b")
      )
    )
  )
) |>
dplyr::select(
  V1,
  V2,
  dplyr::starts_with("D1"),
  V3,
  V4,
  dplyr::starts_with("D2"),
  V5
)
expo_prob <- 1 / (1 + exp(0.4 * X[, 1] + 0.2 * X[, 2] - 0.6 * X[, 3] +
                          0.4 * X[, 6] + 0.6 * X[, 8] - 0.2 * X[, 9]))
W <- rbinom(n, 1, expo_prob)
event_prob <- 1 / (1 + exp(2 * (pmax(2 * X[, 1], 0) * W - X[, 2] +
                           X[, 6] + 3 * X[, 9])))
Y <- rbinom(n, 1, event_prob)
cf <- grf::causal_forest(X, Y, W)
cb1 <- CovariateBalance(cf)
cb2 <- CovariateBalance(
  cf,
  covariates = character(0),
  names = c(
  "medium imbalance" = "V1",
  "low imbalance" = "V2",
  "high imbalance" = "V3",
  "no imbalance" = "V4",
  "discrete 1" = "D1",
  "discrete 2" = "D2"
  )
)
cb3 <- CovariateBalance(
  cf,
  covariates = character(0),
  names = c(
    "medium imbalance" = "V1",
    "low imbalance" = "V2",
    "high imbalance" = "V3",
    "no imbalance" = "V4"
  ),
  treatment_name = "Treatment",
  love_breaks = seq(0, 0.5, 0.1),
  love_xlim = c(0, 0.5),
  cd_nrow = 2,
  cd_x_scale_width = 1,
  cd_bar_width = 0.3
)
```

`decimalplaces` *Determine number of decimal places*

Description

Determine number of decimal places

Usage

```
decimalplaces(x)
```

Arguments

`x`	Numeric, a single decimal number

Value

The number of decimal places in x

`DiscreteCovariateNames` *Extract discrete covariate names*

Description

Detect elements in covariates which match a string from the discrete_covariates argument.

Usage

```
DiscreteCovariateNames(covariates, discrete_covariates = NULL)
```

Arguments

`covariates`	character, names of covariates
`discrete_covariates`	character, names of discrete covariates. Currently it is assumed that discrete covariates are one-hot encoded with naming in covariates following `{fct_nm}_{lvl_nm}`.

Value

A character vector with elements from covariates matching the names supplied in discrete_covariates.

Author(s)

KIJA

Examples

```
one_hot_df <- mtcars |>
  dplyr::mutate(across(c(2, 8:11), factor)) |>
  as.data.frame() |>
  DiscreteCovariatesToOneHot(cyl)
EpiForsk:::DiscreteCovariateNames(colnames(one_hot_df), c("cyl"))
```

`DiscreteCovariatesToOneHot` *One-hot encode factors*

Description

Convert factors in a data frame to one-hot encoding.

Usage

```
DiscreteCovariatesToOneHot(df, factors = dplyr::everything())
```

Arguments

`df`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`factors`	<`tidy-select`> One or more unquoted expressions naming factors in df to one-hot encode.

Value

Data frame with one-hot encoded factors. One-hot encoded columns have names `{fct_nm}_{lvl_nm}`.

Author(s)

KIJA

Examples

```
mtcars |>
dplyr::mutate(dplyr::across(c(2, 8:11), factor)) |>
 as.data.frame() |>
 DiscreteCovariatesToOneHot(cyl)
mtcars |>
dplyr::mutate(dplyr::across(c(2, 8:11), factor)) |>
 as.data.frame() |>
 DiscreteCovariatesToOneHot(c(2, 8:11))
```

`fct_confint` *Confidence set for functions of model parameters*

Description

Computes confidence sets of functions of model parameters by computing a confidence set of the model parameters and returning the codomain of the provided function given the confidence set of model parameters as domain.

Usage

```
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  ...
)

## S3 method for class 'lm'
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  return_beta = FALSE,
  n_grid = NULL,
  k = NULL,
  len = 0.1,
  ...
)

## S3 method for class 'glm'
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  return_beta = FALSE,
  n_grid = NULL,
  k = NULL,
  len = 0.1,
  ...
)

## S3 method for class 'lms'
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  return_beta = FALSE,
  n_grid = NULL,
  k = NULL,
  len = 0.1,
  ...
)

## Default S3 method:
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  ...
)
```

Arguments

`object`	A fitted model object.
`f`	A function taking the parameter vector as its single argument, and returning a numeric vector.
`which_parm`	Either a logical vector the same length as the coefficient vector, with `TRUE` indicating a coefficient is used by `f`, or an integer vector with the indices of the coefficients used by `f`.
`level`	The confidence level required.
`...`	Additional argument(s) passed to methods.
`return_beta`	Logical, if `TRUE` returns both the confidence limits and the parameter values used from the boundary of the parameter confidence set.
`n_grid`	Either `NULL` or an integer vector of length 1 or the number of `TRUE`/indices in which_parm. Specifies the number of grid points in each dimension of a grid with endpoints defined by len. If `NULL` or `0L`, will instead sample k points uniformly on a sphere.

`k`	If n_grid is `NULL` or `0L`, the number of points to sample uniformly from a sphere.
`len`	numeric, the radius of the sphere or box used to define directions in which to look for boundary points of the parameter confidence set.

Details

Assume the response Y and predictors X are given by a generalized linear model, that is, they fulfill the assumptions

$$E(Y|X) = \mu(X^T\beta)$$
$$V(Y|X) = \psi\nu(\mu(X^T\beta))$$
$$Y|X \sim \varepsilon(\theta, \nu_\psi).$$

Here μ is the mean value function, ν is the variance function, and ψ is the dispersion parameter in the exponential dispersion model $\varepsilon(\theta, \nu_\psi)$, where θ is the canonical parameter and ν_ψ is the structure measure. Then it follows from the central limit theorem that

$$\hat{\beta} \sim N(\beta, (X^TWX)^{-1})$$

will be a good approximation in large samples, where X^TWX is the Fisher information of the exponential dispersion model.

From this, the combinant

$$(\hat{\beta} - \beta)^T X^TWX(\hat{\beta} - \beta)$$

is an approximate pivot, with a χ^2_p distribution. Then

$$C_\beta = \{\beta|(\hat{\beta} - \beta)^T X^TWX(\hat{\beta} - \beta) < \chi^2_p(1 - \alpha)\}$$

is an approximate $(1 - \alpha)$-confidence set for the parameter vector β. Similarly, confidence sets for sub-vectors of β can be obtained by the fact that marginal distributions of normal distributions are again normally distributed, where the mean vector and covariance matrix are appropriate subvectors and submatrices.

Finally, a confidence set for the transformed parameters $f(\beta)$ is obtained as

$$\{f(\beta)|\beta \in C_\beta\}$$

Note this is a conservative confidence set, since parameters outside the confidence set of β can be mapped to the confidence set of the transformed parameter.

To determine C_β, `fct_confint()` uses a convex optimization program when f is follows DCP rules. Otherwise, it finds the boundary by taking a number of points around $\hat{\beta}$ and projecting them onto the boundary. In this case, the confidence set of the transformed parameter will only be valid if the boundary of C_β is mapped to the boundary of the confidence set for the transformed parameter.

The points projected to the boundary are either laid out in a grid around $\hat{\beta}$, with the number of points in each direction determined by `n_grid`, or uniformly at random on a hypersphere, with the number of points determined by `k`. The radius of the grid/sphere is determined by `len`.

@section Progress bar:

To print a progress bar with information about the fitting process, wrap the call to fct_confint in with_progress, i.e. `progressr::with_progress({result <- fct_confint(object, f)})`

@section Specifying a plan to resolve futures:

`fct_confint()` uses futures to enable parallel processing. Use the `future::plan()` function to specify how futures are resolved.

Value

A tibble with columns estimate, conf.low, and conf.high or if return_beta is `TRUE`, a list with the tibble and the beta values on the boundary used to calculate the confidence limits.

Author(s)

KIJA

Examples

```
data <- 1:5 |>
  purrr::map(
    \(x) {
      name = paste0("cov", x);
      dplyr::tibble("{name}" := rnorm(100, 1))
    }
  ) |>
  purrr::list_cbind() |>
  dplyr::mutate(
  y = rowSums(dplyr::across(dplyr::everything())) + rnorm(100)
  )
lm <- lm(
 as.formula(
  paste0("y ~ 0 + ", paste0(names(data)[names(data) != "y"], collapse = " + "))
 ),
 data
)
fct_confint(lm, sum)
fct_confint(lm, sum, which_parm = 1:3, level = 0.5)
```

`FlattenDatesDT` *Flatten Date Intervals*

Description

A data.table compatible function for simplifying time interval data

Usage

```
FlattenDatesDT(
  data,
  id,
  in_date,
  out_date,
  status = NULL,
  overlap_handling = c("none", "first", "most_recent"),
  lag = 0
)
```

Arguments

`data`	A list, A data frame, or a data frame extension (e.g. a tibble) coercible to data.table.
`id`	One or more unquoted expression naming the id variables in data.
`in_date`	One unquoted expressions naming the start date variable in data.
`out_date`	One unquoted expression naming the end date variable in data.
`status`	One or more unquoted expressions naming a status variable in data, such as region or hospitalization reason.
`overlap_handling`	A character naming the method for handling overlaps within an individuals time when `status` has been specified. • "none": No special handling of the overlapping time intervals within person is done. • "first": The `status` mentioned first, that is, has the smallest `in_date`, dominates. • "most_recent" (default): The most recent `status`, that is, the one with the largest `in_date`, dominates. When the most recent `status` is fully contained within an older (and different) `status` then the `out_date` associated with the most recent `in_date` is kept, but the remaining time from the older `status` is removed. See examples below. We currently don't have a method that lets the most recent status dominate and then potentially return to an older longer running status. If this is needed, please contact ADLS.
`lag`	A numeric, giving the number of days allowed between time intervals that should be collapsed into one.

Details

This functions identifies overlapping time intervals within individual and collapses them into distinct and disjoint intervals. When `status` is specified these intervals are both individual and status specific.

If `lag` is specified then intervals must be more then `lag` time units apart to be considered distinct.

Value

A data.table with the `id`, `status` if specified and simplified `in_date` and `out_date`. The returned data is sorted by `id` and `in_date`.

Author(s)

ADLS, EMTH, ASO & KIJA

Examples

```
### The flatten function works with both dates and numeric

dat <- data.frame(
   ID    = c(1, 1, 1, 2, 2, 3, 3, 4),
   START = c(1, 2, 5, 3, 6, 2, 3, 6),
   END   = c(3, 3, 7, 4, 9, 3, 5, 8))
dat |> FlattenDatesDT(ID, START, END)

dat <- data.frame(
   ID    = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
   START = as.Date(c("2012-02-15", "2005-12-13", "2006-01-24",
                     "2002-03-14", "1997-02-27",
                     "2008-08-13", "1998-09-23",
                     "2005-01-12", "2007-05-10")),
   END   = as.Date(c("2012-06-03", "2007-02-05", "2006-08-22",
                     "2005-02-26", "1999-04-16",
                     "2008-08-22", "2015-01-29",
                     "2007-05-07", "2008-12-12")))
dat |> FlattenDatesDT(ID, START, END)



###  Allow for a 5 days lag between

dat |> FlattenDatesDT(ID, START, END, lag = 5)



### Adding status information

dat <- data.frame(
   ID     = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
   START  = as.Date(c("2012-02-15", "2005-12-13", "2006-01-24",
                      "2002-03-14", "1997-02-27",
                      "2008-08-13", "1998-09-23",
                      "2005-01-12", "2007-05-10")),
   END    = as.Date(c("2012-06-03", "2007-02-05", "2006-08-22",
                      "2005-02-26", "1999-04-16",
                      "2008-08-22", "2015-01-29",
                     "2007-05-07", "2008-12-12")),
   REGION = c("H", "H", "N", "S", "S", "M", "N", "S", "S"))

# Note the difference between the the different overlap_handling methods
dat |> FlattenDatesDT(ID, START, END, REGION, "none")
dat |> FlattenDatesDT(ID, START, END, REGION, "first")
dat |> FlattenDatesDT(ID, START, END, REGION, "most_recent")
```

`flatten_date_intervals` *Flatten Date Intervals*

Description

A tidyverse compatible function for simplifying time interval data

Usage

```
flatten_date_intervals(
  data,
  id,
  in_date,
  out_date,
  status = NULL,
  overlap_handling = "most_recent",
  lag = 0
)
```

Arguments

`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`id`	<`tidy-select`> One or more unquoted expression naming the id variables in data.
`in_date`	<`data-masking`> One unquoted expressions naming the start date variable in data.
`out_date`	<`data-masking`> One unquoted expression naming the end date variable in data.
`status`	<`tidy-select`> One or more unquoted expressions naming a status variable in data, such as region or hospitalization reason.
`overlap_handling`	A character naming the method for handling overlaps within an individuals time when `status` has been specified. • "none": No special handling of the overlapping time intervals within person is done. • "first": The `status` mentioned first, that is, has the smallest `in_date`, dominates. • "most_recent" (default): The most recent `status`, that is, the one with the largest `in_date`, dominates. When the most recent `status` is fully contained within an older (and different) `status` then the `out_date` associated with the most recent `in_date` is kept, but the remaining time from the older `status` is removed. See examples below. We currently don't have a method that lets the most recent status dominate and then potentially return to an older longer running status. If this is needed, please contact ADLS.

`lag`	A numeric, giving the number of days allowed between time intervals that should be collapsed into one.

Details

This functions identifies overlapping time intervals within individual and collapses them into distinct and disjoint intervals. When `status` is specified these intervals are both individual and status specific.

If `lag` is specified then intervals must be more then `lag` time units apart to be considered distinct.

Value

A data frame with the `id`, `status` if specified and simplified `in_date` and `out_date`. The returned data is sorted by `id` and `in_date`.

Author(s)

ADLS, EMTH & ASO

Examples

```
### The flatten function works with both dates and numeric

dat <- data.frame(
   ID    = c(1, 1, 1, 2, 2, 3, 3, 4),
   START = c(1, 2, 5, 3, 6, 2, 3, 6),
   END   = c(3, 3, 7, 4, 9, 3, 5, 8))
dat |> flatten_date_intervals(ID, START, END)

dat <- data.frame(
   ID    = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
   START = as.Date(c("2012-02-15", "2005-12-13", "2006-01-24",
                     "2002-03-14", "1997-02-27",
                     "2008-08-13", "1998-09-23",
                     "2005-01-12", "2007-05-10")),
   END   = as.Date(c("2012-06-03", "2007-02-05", "2006-08-22",
                     "2005-02-26", "1999-04-16",
                     "2008-08-22", "2015-01-29",
                     "2007-05-07", "2008-12-12")))
dat |> flatten_date_intervals(ID, START, END)



###  Allow for a 5 days lag between

dat |> flatten_date_intervals(ID, START, END, lag = 5)



### Adding status information

dat <- data.frame(
   ID     = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
   START  = as.Date(c("2012-02-15", "2005-12-13", "2006-01-24",
                      "2002-03-14", "1997-02-27",
                      "2008-08-13", "1998-09-23",
                      "2005-01-12", "2007-05-10")),
   END    = as.Date(c("2012-06-03", "2007-02-05", "2006-08-22",
                      "2005-02-26", "1999-04-16",
                      "2008-08-22", "2015-01-29",
                     "2007-05-07", "2008-12-12")),
   REGION = c("H", "H", "N", "S", "S", "M", "N", "S", "S"))

# Note the difference between the the different overlap_handling methods
dat |> flatten_date_intervals(ID, START, END, REGION, "none")
dat |> flatten_date_intervals(ID, START, END, REGION, "first")
dat |> flatten_date_intervals(ID, START, END, REGION, "most_recent")
```

`floor_dec` *Round numbers down to a given number of decimal places*

Description

Round numbers down to a given number of decimal places

Usage

```
floor_dec(x, digits = 1)
```

Arguments

`x`	a numeric vector
`digits`	integer indicating the number of decimal places

Value

The rounded down numeric vector

`freq_function` *Frequency Tables with Percentage and Odds Ratios*

Description

A method for making 1- and 2-way frequency tables with percentages and odds ratios.

Usage

```
freq_function(
  normaldata,
  var1,
  var2 = NULL,
  by_vars = NULL,
  include_NA = FALSE,
  values_to_remove = NULL,
  weightvar = NULL,
  textvar = NULL,
  number_decimals = 2,
  output = c("all", "numeric", "col", "colw", "row", "roww", "total", "totalw"),
  chisquare = FALSE
)
```

Arguments

`normaldata`	A data frame or data frame extension (e.g. a tibble).
`var1`	A character string naming the first variable to get frequencies.
`var2`	An optional character naming the second variable to get frequencies. If `NULL` (standard) a 1-way frequency table of only `var1` is created, and if `var2` is specified a 2-way table is returned.
`by_vars`	An optional character vector naming variables in `normal_data` to stratify the calculations and output by. That is, ALL calculations will be made within the combinations of variables in the vector, hence it's possible to get N and % for many groups in one go.
`include_NA`	A logical. If `FALSE` (standard) missing variables (`NA`'s) will be removed from `var1` and `var2`. Any missing values in `by_vars` will not be removed. If `TRUE` all missing values will be included in calculations and the output.
`values_to_remove`	An optional character vector. When specified all values from `var1` and `var2` found in `values_to_remove` will be removed from the calculations and output.
`weightvar`	An optional character naming a column in `normaldata` with numeric weights for each observation. If `NULL` (standard) all observations have weight 1.
`textvar`	An optional character. When specified `textvar` is added to the resulting table as a comment. When `NULL` (standard) no such text addition is made.
`number_decimals`	A numeric indicating the number of decimals to show on percentages and weighted frequencies in the combined frequency and percent variables.
`output`	A character indicating the output type wanted: • `"all"` - will give ALL output from tables. In many cases unnecessary and hard to get an overview of. This is set as the standard. • `"numeric"` - will give frequencies and percents as numeric variables only, thus the number_decimals option is not in effect. This option might be useful when making figures/graphs.

- `"col"` - will only give unweighted number of observations and weighted column percent (if weights are used, otherwise unweighted)
- `"colw"` - will only give weighted number of observations and weighted column percent (if weights are used, otherwise unweighted)
- `"row"`- will only give unweighted number of observations and weighted row percent (if weights are used, otherwise unweighted). Only works in two-way tables (`var2` is specified)
- `"roww"` - will only give weighted number of oberservations and weighted column percent (if weights are used, otherwise unweighted). Only works in two-way tables (`var2` is specified)
- `"total"` - will only give unweighted number of observations and weighted percent of the total (if weights are used, otherwise unweighted). Only works in two-way tables (`var2` is specified)
- `"totalw"` - will only give weighted number of observations and weighted percent of the total (if weights are used, otherwise unweighted). Only works in two-way tables (`var2` is specified)
- Any other text will give the default ("all")

`chisquare`	A logical. `FALSE` (standard) will not calculate p-value for the chi-square test for two-way tables (`var2` is specified). If `TRUE`, the table will include the chi-square p-value as well as the chi-square statistic and the corresponding degrees of freedom. It will be included in the output whichever output option have been specified. No chi-square test is performed or included in one-way tables (`var2` is unspecified)

Value

A frequency table as a data frame object.

Author(s)

ASO

See Also

`freq_function_repeated()` to to get frequencies for multiple variables in one go.

Examples

```
data("starwars", package = "dplyr")

test_table1 <- freq_function(
  starwars,
  var1 = "homeworld"
)

test_table2 <- freq_function(
  starwars,
  var1 = "sex",
  var2 = "eye_color",
  output = "total"
)

test_table3 <- freq_function(
  starwars,
  var1 = "hair_color",
  var2 = "skin_color",
  by_vars = "gender",
  output = "col",
  number_decimals = 5
)
```

---

freq_function_repeated

Wrapper for `freq_function()` *to get frequencies for many variables in one go.*

Description

A method for making multiple 1- and 2-way frequency tables with percentages and odds ratios.

Usage

```
freq_function_repeated(
  normaldata,
  var1,
  var2 = NULL,
  by_vars = NULL,
  include_NA = FALSE,
  values_to_remove = NULL,
  weightvar = NULL,
  textvar = NULL,
  number_decimals = 2,
  output = c("all", "numeric", "col", "colw", "row", "roww", "total", "totalw"),
  chisquare = FALSE
)
```

Arguments

`normaldata`	A data frame or data frame extension (e.g. a tibble).
`var1`	A character vector with the names of the first variable to get frequencies from for each frequency table.
`var2`	An optional character naming the second variable to get frequencies. If `NULL` (standard) 1-way frequency tables of only variables in `var1` are created, and if `var2` is specified 2-way tables are returned.

`by_vars`	An optional character vector naming variables in `normal_data` to stratify the calculations and output by. That is, ALL calculations will be made within the combinations of variables in the vector, hence it's possible to get N and % for many groups in one go.
`include_NA`	A logical. If `FALSE` (standard) missing variables (`NA`'s) will be removed from `var1` and `var2`. Any missing values in `by_vars` will not be removed. If `TRUE` all missing values will be included in calculations and the output.
`values_to_remove`	An optional character vector. When specified all values from `var1` and `var2` found in `values_to_remove` will be removed from the calculations and output.
`weightvar`	An optional character naming a column in `normaldata` with numeric weights for each observation. If `NULL` (standard) all observations have weight 1.
`textvar`	An optional character. When specified `textvar` is added to the resulting table as a comment. When `NULL` (standard) no such text addition is made.
`number_decimals`	A numeric indicating the number of decimals to show on percentages and weighted frequencies in the combined frequency and percent variables.
`output`	A character indicating the output type wanted: • `"all"` - will give ALL output from tables. In many cases unnecessary and hard to get an overview of. This is set as the standard. • `"numeric"` - will give frequencies and percents as numeric variables only, thus the number_decimals option is not in effect. This option might be useful when making figures/graphs. • "col" - will only give unweighted number of observations and weighted column percent (if weights are used, otherwise unweighted) • `"colw"` - will only give weighted number of observations and weighted column percent (if weights are used, otherwise unweighted) • `"row"`- will only give unweighted number of observations and weighted row percent (if weights are used, otherwise unweighted). Only works in two-way tables (`var2` is specified) • `"roww"` - will only give weighted number of oberservations and weighted column percent (if weights are used, otherwise unweighted). Only works in two-way tables (`var2` is specified) • `"total"` - will only give unweighted number of observations and weighted percent of the total (if weights are used, otherwise unweighted). Only works in two-way tables (`var2` is specified) • `"totalw"` - will only give weighted number of observations and weighted percent of the total (if weights are used, otherwise unweighted). Only works in two-way tables (`var2` is specified) • Any other text will give the default ("all")
`chisquare`	A logical. `FALSE` (standard) will not calculate p-value for the chi-square test for two-way tables (`var2` is specified). If `TRUE`, the table will include the chi-square p-value as well as the chi-square statistic and the corresponding degrees of freedom. It will be included in the output whichever output option have been specified. No chi-square test is performed or included in one-way tables (`var2` is unspecified)

Value

Multiple frequency tables stored in a data frame object.

Author(s)

ASO

See Also

`freq_function()` for the function that creates frequency tables for single variables.

Examples

```
# Examples
data("starwars", package = "dplyr")
test_table1 <- freq_function_repeated(
  starwars,
  var1 = c("sex","homeworld","eye_color"),
  include_NA = TRUE
)
test_table2 <- freq_function_repeated(
  starwars,
  var1 = c("homeworld","eye_color","skin_color"),
  var2 = "sex",
  output = "col",
  number_decimals = 3
)
test_table3 <- freq_function_repeated(
  starwars,
  var1 = c("homeworld","eye_color","skin_color"),
  var2 = "sex",
  by_vars = c("gender"),
  output = "row"
)
```

`ListPermute` *Permute the levels of a nested list*

Description

`ListPermute()` permutes the nesting structure of a nested list. e.g. $x[["a"]][["b"]][["c"]]$ is equivalent to $ListPermute(x, c(3, 1, 2))[["c"]][["a"]][["b"]]$.

Usage

```
ListPermute(x, order, depth = NULL, enforce_rectangular = TRUE)
```

Arguments

x	A nested list. By default, the list is assumed to be rectangular. If `enforce_rectangular` = FALSE, the length of each level in the list will be based on the first element from the level above.
order	A numeric vector, containing a permutation of the levels of `x`. If `depth` is user specified, the length of `order` must equal depth.
depth	A natural number or NULL (default). If NULL, the depth of the list is inferred by following the first element of the list at each level until the next element is not a list. If a natural number, a number smaller than the actual depth of `x` will result in permutation of the lists at the level of `depth`, and a number higher than the actual depth of `x` will result in extra levels added to the output at the locations specified by `order`.
enforce_rectangular	logical, should the function exit with an error in case `x` is not rectangular?

Value

A list, with the components of `x` permuted according to `order`.

Author(s)

KIJA

Examples

```
list_builder <- function(len, nm = NULL, index = NULL, len_orig = len) {
  out <- vector("list", len[1])
  if (!is.null(nm)) names(out) <- nm[[1]]
  if (length(len) > 1) {
    return(
      lapply(
        if (is.null(nm)) seq_along(out) else structure(seq_along(out), names = nm[[1]]),
        \(i) {
          list_builder(
          len = len[-1],
          nm = if (is.null(nm)) NULL else nm[-1],
          index = c(index, i),
          len_orig = len_orig
          )
        }
      )
    )
  } else {
    for (i in seq_along(out)) {
      index2 <- c(index, i)
      out[[i]] <- as.numeric(
      (index2 - c(rep(1, length(len_orig) - 1), 0)) %*%
      c(rev(cumprod(rev(len_orig[-1]))), 1)
      )
    }
    return(out)
  }
}

l1 <- list_builder(c(3, 2))
ListPermute(l1, c(2,1))

l2 <- list_builder(c(3, 2), list(letters[24:26], letters[14:13]))
ListPermute(l2, c(2,1))

l3 <- list_builder(
c(4, 3, 2),
list(letters[1:4], letters[24:26], letters[14:13])
)
ListPermute(l3, c(2,1,3))

l4 <- list_builder(
c(4, 3, 2, 3),
list(letters[1:4], letters[24:26], letters[14:13], letters[16:18])
)
ListPermute(l4, c(4, 1, 2, 3))

l5 <- list_builder(
c(4, 3, 2, 3), list(letters[1:4], letters[24:26], letters[14:13], letters[16:18])
)
l5a <- l5
for (i in 3:4) {
  for (j in 1:3) {
    for (k in 1:2) {
      l5a[[i]][[j]][[k]] <- l5[[i]][[j]][[k]][[1]]
    }
  }
}
l5b <- l5
for (i in 1:2) {
  for (j in 1:3) {
    for (k in 1:2) {
      l5b[[i]][[j]][[k]] <- l5[[i]][[j]][[k]][[1]]
    }
  }
}
ListPermute(l5, c(3, 1, 2), depth = 3L)
ListPermute(l5a, c(4, 1, 2, 3), depth = 4L, enforce_rectangular = FALSE)
ListPermute(l5b, c(4, 1, 2, 3), depth = 4L, enforce_rectangular = FALSE)

l6 <- list()
ListPermute(l6, vector("numeric"))

l7 <- list(c(1, 2))
ListPermute(l7, 1)
```

lms *Wrapper around lm for sibling design*

Description

Fits a linear model using demeaned data. Useful for sibling design.

Usage

```
lms(formula, data, grp_id, obs_id = NULL, ...)

## S3 method for class 'lms'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`formula`	A formula, used to create a model matrix with demeaned columns.
`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`grp_id`	<`data-masking`> One unquoted expression naming the id variable in data defining the groups to demean, e.g. sibling groups.
`obs_id`	<`data-masking`> Optional, One unquoted expression naming an id variable to keep track of the input data order.
`...`	Additional arguments to be passed to lm. In print, additional arguments are ignored without warning.
`x`	An S3 object with class lms.
`digits`	The number of significant digits to be passed to format(coef(x), .) when print()ing.

Details

`lms` estimates parameters in the linear model

$$y_{ij_i} = \alpha_i + x_{ij_i}^T \beta + \varepsilon_{ij_i}$$

where α_i is a group (e.g. sibling group) specific intercept and x_{ij_i} are covariate values for observation j_i in group i. $\varepsilon_{ij_i} \sim N(0, \sigma^2)$ is a normally distributed error term. It is assumed that interest is in estimating the vector β while α_i are nuisance parameters. Estimation of β uses the mean deviation method, where

$$y_{ij_i}^{'} = y_{ij_i} - y_i$$

is regressed on

$$x_{ij_i}^{'} = x_{ij_i} - x_i.$$

Here y_i and x_i refers to the mean of y and x in group i.
`lms` can keep track of observations by providing a unique identifier column to `obs_id`. `lms` will return `obs_id` so it matches the order of observations in model.
`lms` only supports syntactic covariate names. Using a non-syntactic name risks returning an error, e.g if names end in + or -.

Value

A list with class `c("lms", "lm")`. Contains the output from `lm` applied to demeaned data according to `formula`, as well as the original data and the provided formula.

Author(s)

KIJA

Examples

```
sib_id <- sample(200, 1000, replace = TRUE)
sib_out <- rnorm(200)
x1 <- rnorm(1000)
x2 <- rnorm(1000) + sib_out[sib_id] + x1
y <- rnorm(1000, 1, 0.5) + 2 * sib_out[sib_id] - x1 + 2 * x2
data <- data.frame(
  x1 = x1,
  x2 = x2,
  y = y,
  sib_id = sib_id,
  obs_id = 1:1000
)
mod_lm <- lm(y ~ x1 + x2, data) # OLS model
mod_lm_grp <- lm(y ~ x1 + x2 + factor(sib_id), data) # OLS with grp
mod_lms <- lms(y ~ x1 + x2, data, sib_id, obs_id) # conditional model
summary(mod_lm)
coef(mod_lm_grp)[1:3]
summary(mod_lms)
print(mod_lms)
```

`logasympBF` *Asymptotic Bayes factors*

Description

The Bayesian equivalent of a significance test for H1: an unrestricted parameter value versus H0: of a specific parameter value based only on data D can be obtained from Bayes factor (BF). Then `BF = Probability(H1|D) / Probability(H0|D)` and is a Bayesian equivalent of a likelihood ratio. It is based on the same asymptotics as the ubiqutous chi-square tests. This BF only depends on the difference in deviance between the models corresponding to H0 and H1 (chisquare) and the dimension d of H1. This BF is monotone in chisquare (and hence the p-value p) for fixed d. It is thus a tool to turn p-values into evidence, also retrospectively. The expression for BF depends on a parameter lambda which expresses the ratio between the information in the prior and the data (likelihood). By default `lambda = min(d / chisquare, lambdamax = 0.255)`. Thus, as chisquare goes to infinity we effectively maximize BF and hence the evidence favoring H1, and opposite for small chisquare has a well-defined watershed where we have equal preferences for H1 and H0. The value 0.255 corresponds to a watershed of 2, that is we prefer H1 when `chisquare > d * 2` and prefer H0 when `chisquare < d * 2`, similar to having a BF that is a continuous version of the Akaike Information Criterion for model selection. For derivations and details, see Rostgaard (2023).

Usage

```
logasympBF(chisq = NA, p = NA, d = 1, lambda = NA, lambdamax = 0.255)

asympBF(chisq = NA, p = NA, d = 1, lambda = NA, lambdamax = 0.255)

invlogasympBF(logasympBF = NA, d = 1, lambda = NA, lambdamax = 0.255)

invasympBF(bf, d = 1, lambda = NA, lambdamax = 0.255)

watershed(chisq)

invwatershed(lambda)
```

Arguments

`chisq`	a non-negative number denoting the difference in deviance between the statistical models corresponding to H0 and H1
`p`	the p value corresponding to the test statistic chisq on d degrees of freedom
`d`	the dimension of H1, `d => 1`
`lambda`	a strictly positive number corresponding to the ratio between the information in the prior and the data
`lambdamax`	an upper limit on lambda
`logasympBF`	log(bf)
`bf`	Bayes factor, a strictly positive number

Details

For fixed dimension d of the alternative hypothesis H1 `asympBF(.) = exp(logasympBF(.))` maps a test statistic (chisquare) or a p-value p into a Bayes factor (the ratio between the probabilities of the models corresponding to each hypothesis). `asympBF(.)` is monotonely increasing in chisquare, attaining the value 1 when `chisquare = d * watershed`. The watershed is thus a device to specify what the user considers a practical null result by choosing `lambdamax = watershed(watershed)`.

For sufficiently large values of chisquare, lambda is estimated as d/chisquare. This behavior can be overruled by specifying lambda. Using `invasympBF(.) = exp(invlogasympBF(.))` we can map a Bayes factor, bf to a value of chisquare.

Likewise, we can obtain the watershed corresponding to a given lambdamax as `invwatershed(lambdamax)`.

Generally in these functions we recode or ignore illegal values of parameters, rather than returning an error code. `chisquare` is always recoded as `abs(chisquare)`. `d` is set to `1` as default, and missing or entered values of `d < 1` are recoded as `d = 1`. Entered values of `lambdamax <= 0` or missing are ignored. Entered values of `lambda <= 0` or missing are ignored in `invwatershed(.)`. we use `abs(lambda)` as argument, `lambda = 0` results in an error.

Author(s)

KLP & KIJA

References

Klaus Rostgaard (2023): Simple nested Bayesian hypothesis testing for meta-analysis, Cox, Poisson and logistic regression models. Scientific Reports. https://doi.org/10.1038/s41598-023-31838-8

Examples

```
# example code

# 1. the example(s) from Rostgaard (2023)
asympBF(p = 0.19, d = 8) # 0.148411
asympBF(p = 0.19, d = 8, lambdamax = 100) # 0.7922743
asympBF(p = 0.19, d = 8, lambda = 100 / 4442) # 5.648856e-05
# a maximal value of a parameter considered practically null
deltalogHR <- -0.2 * log(0.80)
sigma <- (log(1.19) - log(0.89)) / 3.92
chisq <- (deltalogHR / sigma) ** 2
chisq # 0.3626996
watershed(chisq)
# leads nowhere useful chisq=0.36<2

# 2. tests for interaction/heterogeneity - real world examples
asympBF(p = 0.26, d = 24) # 0.00034645
asympBF(p = 0.06, d = 11) # 0.3101306
asympBF(p = 0.59, d = 7) # 0.034872

# 3. other examples
asympBF(p = 0.05) # 2.082664
asympBF(p = 0.05, d = 8) # 0.8217683
chisq <- invasympBF(19)
chisq # 9.102697
pchisq(chisq, df = 1, lower.tail = FALSE) # 0.002552328
chisq <- invasympBF(19, d = 8)
chisq # 23.39056
pchisq(chisq, df = 8, lower.tail = FALSE) # 0.002897385
```

---

many_merge *Merging Many Data Frames with Name Handling*

Description

Function to join/merge multiple data.frames with one or more common variable names.

Usage

```
many_merge(by, first_data, ...)
```

Arguments

by	A join specification created with join_by(), or a character vector of variables to join by. The by must be present in all data frames first_data and ....
first_data	A data frame (presented on the left in the final table).
...	Data frames to merge onto first_data.

Value

The many_merge() function returns a data frame.

Author(s)

ASO

Examples

```
# Create some dummy data
testdata_id <- c(1:10)
var1 <- rep(letters[1:5], times = 2)
var2 <- letters[1:10]
var3 <- rep(letters[11:12], times = 5)
var4 <- letters[13:22]
var5 <- letters[11:20]

# Rename alle the variables to "var"
data1 <- data.frame(testdata_id, var = var1)
data2 <- data.frame(testdata_id, var = var2)
data3 <- data.frame(testdata_id, var = var3)
data4 <- data.frame(testdata_id, var = var4)
data5 <- data.frame(testdata_id, var = var5)

# Many merge
final_data <- many_merge(
  by = c("testdata_id"),
  data1,
  data2,
  data3,
  data4,
  data5
)
```

---

multi_join *Join many data frames with name handling*

Description

Function to join multiple data.frames with one or more common variable names.

Usage

```
multi_join(..., .by)
```

Arguments

`...`	Data frames to join. Each argument in `...` must either be a data.frame or a list of data.frames.
`.by`	A character vector of variables to join by. The `.by` must be present in all data frames in `...`.

Value

The `multi_join()` function returns a data frame.

Author(s)

KIJA

Examples

```
# Create some dummy data
testdata_id <- c(1:10)
a1 <- 1:10; b1 <- rep(letters[1:5], times = 2); c1 <- runif(10)
a2 <- 11:20; b2 <- letters[1:10]
a3 <- 21:30; b3 <- rep(letters[11:12], times = 5)
a4 <- 31:40; b4 <- letters[13:22]
a5 <- 41:50; b5 <- letters[11:20]

# Define data.frames with common key and shared column names
data1 <- data.frame(testdata_id, a = a1, b = b1, c = c1)
data2 <- data.frame(testdata_id, b = b2, a = a2)
data3 <- data.frame(testdata_id, a = a3, b = b3)
data4 <- data.frame(testdata_id, a = a4, b = b4)
data5 <- data.frame(testdata_id, a = a5, b = b5)

# multi join
final_data <- multi_join(
  data1,
  data2,
  data3,
  data4,
  data5,
  .by = "testdata_id"
)
```

`odds_ratio_function` *Easier to perform logistic and log-linear regressions giving a standardized output table*

Description

odds_ratio_function analyses specified data given user specifications, including outcome, exposures and possible weights. It can handle survey-data, but not complex sampling schemes (if specified as survey-data, the model will create a simple survey-object from the data, using weights as specified - if not specified, the weights are 1 for each observation) The standard regression is logistic regression (yielding Odds Ratios=OR) but it is possible to perform a log-linear regression (yielding Risk Ratios=RR) instead, if specified and requirements are met.

Usage

```
odds_ratio_function(
  normaldata,
  outcomevar,
  expvars,
  number_decimals = 2,
  alpha = 0.05,
  regtype = c("logistic", "log-linear"),
  matchgroup = NULL,
  matchtiemethod = c("exact", "approximate", "efron", "breslow"),
  values_to_remove = NULL,
  weightvar = NULL,
  surveydata = FALSE,
  textvar = NULL,
  model_object = FALSE
)
```

Arguments

`normaldata`	A data frame or data frame extension (e.g. a tibble).
`outcomevar`	A character string naming of factor variable in normaldata to use as the outcome.
`expvars`	A character vector with the names of the exposure variables (either numeric or factors). Any transformations or interactions to be included must also be specified, e.g. `c("Var1", "I(Var1^2)", "Var2", "Var3*Var4")`.
`number_decimals`	An integer giving the number of decimals to show in the standardized output (default is two decimals).
`alpha`	A scalar, between 0 and 1 specifying the desired significance level of the confidence intervals (default is 0.05 which will yield the usual 95% confidence interval).

`regtype`	A character string specifying the analysis method. Can either be "logistic" for logistic regression (the default) or "log-linear" for log-linear regression. Log-linear regression can only be used with binomial, unconditional analysis.
`matchgroup`	Character string specifying a variable in normaldata to condition the analysis on. Can only be used in binomial logistic regression models (default is NULL).
`matchtiemethod`	Character string specifying the method for ties when using a matched/conditional analysis. The default options is "exact", however this option does not take weights into account for the analysis, so if weights (other than 1) are used, another option should be selected. Other options are "approximate", "efron", and "breslow" - for further explanations, see documentation for clogit.
`values_to_remove`	A Character vector specifying values to remove from ALL variables used in the regression before the analysis (default is NULL). This is useful if some value(s) are used consistently to encode missing/irrelevant in the data (e.g. c("888", "987") - normal missing (NA) don't need to be specified as it will be removed automatically. Do NOT remove the reference values as this will lead to unexpected results!
`weightvar`	A character string specifying a numeric variable in normaldata with pre-calculated weights for observations in the analysis. The default value NULL corresponds to weight 1 for all observations.
`surveydata`	A Boolean specifying whether the data comes from a survey (default is FALSE).
`textvar`	A character string with text (like a note) to be added to the output. The default value NULL corresponds to no added note.
`model_object`	A Boolean. If TRUE, returns the raw output object from the analysis instead of the standard output. This might be useful to see information not included in the standardized output (default is FALSE).

Value

A standardized analysis object with results from a model.

Author(s)

ASO

See Also

`odds_ratio_function_repeated()` to perform several analysis in one go.

Examples

```
### Binomial outcome
data(logan, package = "survival")

resp <- levels(logan$occupation)
n <- nrow(logan)
indx <- rep(1:n, length(resp))
logan2 <- data.frame(
  logan[indx,],
  id = indx,
  tocc = factor(rep(resp, each=n))
)
logan2$case <- (logan2$occupation == logan2$tocc)
logan2$case <- as.factor(logan2$case)
logan2$case <- relevel(logan2$case, ref = "FALSE")

# Standard binomial logistic regression but using interaction for exposures:
func_est1 <- odds_ratio_function(
  logan2,
  outcomevar = "case",
  expvars = c("tocc", "education", "tocc:education")
)


# Conditional binomial logistic regression with some extra text added:
func_est2 <- odds_ratio_function(
  logan2,
  outcomevar = "case",
  expvars = c("tocc", "tocc:education"),
  matchgroup = "id",
  textvar = "Testing function"
)


# Standard binomial logistic regression as survey data with no prepared
# weights:
func_est3 <- odds_ratio_function(
  logan2,
  outcomevar = "case",
  expvars = c("tocc", "education"),
  surveydata = TRUE
)

# Example changing significance level and the number of decimals in fixed
# output and adding some text:
func_est4 <- odds_ratio_function(
  logan2,
  outcomevar = "case",
  expvars = c("tocc", "education"),
  number_decimals = 5,
  alpha = 0.01,
  textvar = "Testing function"
)

# Getting raw output from the regression function:
func_est5 <- odds_ratio_function(
  logan2,
  outcomevar = "case",
  expvars = c("tocc", "education"),
  model_object = TRUE
)
```

```
### Polytomous/multinomial outcome
data(api, package = "survey")

# As normal data, but using weights:
func_est6 <- odds_ratio_function(
  apiclus2,
  outcomevar = "stype",
  expvars = c("ell", "meals", "mobility", "sch.wide"),
  weightvar = "pw"
)

# As survey data with weights:
func_est7 <- odds_ratio_function(
  apiclus2,
  outcomevar = "stype",
  expvars = c("ell", "meals", "mobility"),
  weightvar = "pw", surveydata = TRUE
)

# Binomial logistic regression with same data (by removing all observations
# with a specific value of outcome):
func_est8 <- odds_ratio_function(
  apiclus2,
  outcomevar = "stype",
  expvars = c("ell", "meals", "mobility"),
  weightvar = "pw",
  values_to_remove = c("E")
)
```

odds_ratio_function_repeated

Wrapper for the `odds_ratio_function()`*to perform several similar analyses in one go*

Description

The function is intended to make it easy to get OR's for several similar models in one go, where either the same analysis is performed except for one variable or the same analysis is performed but by each variable (each level of the variable is analysed separately).

Usage

```
odds_ratio_function_repeated(
  normaldata,
  outcomevar,
  expvars,
  adjustment_fixed = NULL,
  by_var = NULL,
  number_decimals = 2,
  alpha = 0.05,
  regtype = c("logistic", "log-linear"),
  matchgroup = NULL,
  matchtiemethod = c("exact", "approximate", "efron", "breslow"),
  values_to_remove = NULL,
  weightvar = NULL,
  surveydata = FALSE,
  textvar = NULL,
  model_object = FALSE
)
```

Arguments

`normaldata`	A data frame or data frame extension (e.g. a tibble).
`outcomevar`	A character vector naming factor variables in normaldata to use as outcomes in separate models.
`expvars`	A character vector naming exposure variables (either numeric or factors) to use in separate models.
`adjustment_fixed`	A character vector naming adjustment variables to include in all models. NULL is the default resulting in no fixed adjustment.
`by_var`	A character vector specifying a factor on which to run the analyses completely separate for all levels. It only works with one variable (default is NULL). NOTE: NA and "" levels will not be used but all other levels will have separate models.
`number_decimals`	An integer giving the number of decimals to show in the standardized output (default is two decimals).
`alpha`	A scalar, between 0 and 1 specifying the desired significance level of the confidence intervals (default is 0.05 which will yield the usual 95% confidence interval).
`regtype`	A character string specifying the analysis method. Can either be "logistic" for logistic regression (the default) or "log-linear" for log-linear regression. Log-linear regression can only be used with binomial, unconditional analysis.
`matchgroup`	Character string specifying a variable in normaldata to condition the analysis on. Can only be used in binomial logistic regression models (default is NULL).
`matchtiemethod`	Character string specifying the method for ties when using a matched/conditional analysis. The default options is "exact", however this option does not take weights into account for the analysis, so if weights (other than 1) are used, another option should be selected. Other options are "approximate", "efron", and "breslow" - for further explanations, see documentation for clogit.
`values_to_remove`	Character vector specifying values to remove from ALL variables used in the regression before the analysis (default is NULL). This is useful if some value(s) are used consistently to encode missing/irrelevant in the data (e.g. c("888",

	"987") - normal missing (NA) don't need to be specified as it will be removed automatically. Do NOT remove the reference values as this will lead to unexpected results!
`weightvar`	A character string specifying a numeric variable in normaldata with pre-calculated weights for observations in the analysis. The default value NULL corresponds to weight 1 for all observations.
`surveydata`	A Boolean specifying whether the data comes from a survey (default is FALSE).
`textvar`	A character string with text (like a note) to be added to the output. The default value NULL corresponds to no added note.
`model_object`	A Boolean. If TRUE, returns the raw output object from the analysis instead of the standard output. This might be useful to see information not included in the standardized output (default is FALSE).

Details

It's possible to have same variable in `expvars` and `adjustment_fixed`.

When a model results in an error, the function will not stop - it continues with other models until done BUT in the output the error text can be seen.

Value

A standardized analysis object with results from multiple models.

Author(s)

ASO

See Also

`odds_ratio_function()` to perform a single logistic or log-linear regression giving a standardized output table.

Examples

```
# Data to use
data("infert", package = "datasets")
infert2 <- infert |>
  dplyr::mutate(
    Age_grp = relevel(as.factor(dplyr::case_when(
      age < 25 ~ "<25",
      25 <= age & age < 35 ~ "25-<35",
      age >= 35 ~ "35+"
    )), ref="25-<35"),
    Parity_grp = relevel(as.factor(dplyr::case_when(
      parity == 1 ~ "1",
      parity >= 2 & parity <= 3 ~ "2-3",
      parity > 3 ~ "4+"
    )), ref="2-3"),
    induced = relevel(as.factor(induced), ref="0"),
    case = relevel(as.factor(case), ref="0"),
    spontaneous = relevel(as.factor(spontaneous), ref="0")
  )

# Two outcomes (Parity_grp, case) with their own set of models, three
# variables included in separate models (spontaneous,induced and education)
# and one variable that is included in all models (Age_grp)
test <- odds_ratio_function_repeated(
  normaldata = infert2,
  outcomevar = c("Parity_grp","case"),
  expvars = c("spontaneous","induced","education"),
  adjustment_fixed = c("Age_grp")
)

# One outcome (case), two variables included in separate models
# (spontaneous and induced), one variable included in all models (Age_grp)
# and all analyses made for each level of another variable (Parity_grp)
test2 <- odds_ratio_function_repeated(
  normaldata = infert2,
  outcomevar = c("case"),
  expvars = c("spontaneous","induced"),
  adjustment_fixed = c("Age_grp"),
  by_var = "Parity_grp"
)
```

`RATEOmnibusTest` *RATE based omnibus test of heterogeneity*

Description

Provides the P-value for a formal test of heterogeneity based on the RATE statistic by Yadlowsky et al.

Usage

```
RATEOmnibusTest(forest, ...)

## S3 method for class 'causal_forest'
RATEOmnibusTest(
  forest,
  level = 0.95,
  target = c("AUTOC", "QINI"),
  q = seq(0.1, 1, 0.1),
  R = 500,
  num.threads = 1,
  seed = NULL,
  honesty = TRUE,
  stabilize.splits = TRUE,
  ...
)

## S3 method for class 'causal_survival_forest'
RATEOmnibusTest(
  forest,
  level = 0.95,
  target = c("AUTOC", "QINI"),
  q = seq(0.1, 1, 0.1),
  R = 500,
  num.threads = 1,
  seed = NULL,
  failure.times = NULL,
  honesty = TRUE,
  stabilize.splits = TRUE,
  sample.fraction = 0.5,
  mtry = min(ceiling(sqrt(ncol(forest$X.orig)) + 20), ncol(forest$X.orig)),
  min.node.size = 5,
  honesty.fraction = 0.5,
  honesty.prune.leaves = TRUE,
  alpha = 0.05,
  imbalance.penalty = 0,
  ...
)
```

Arguments

`forest`	An object of class `causal_forest`, as returned by causal_forest(), with binary treatment.
`...`	additional arguments passed to causal_forest. By default, the arguments used by forest will be used to train new forests on the random half-samples. Arguments provided through ... will override these. Note that sample.weights and clusters are passed to both causal_forest and rank_average_treatment_effect.fit.
`level`	numeric, level of RATE confidence interval.
`target`	character, see rank_average_treatment_effect.
`q`	numeric, see rank_average_treatment_effect.
`R`	integer, see rank_average_treatment_effect
`num.threads`	passed to causal_forest. Number of threads used in training. Default value is 1.
`seed`	numeric, either length 1, in which case the same seed is used for both new forests, or length 2, to train each forest with a different seed. Default is `NULL`, in which case two seeds are randomly sampled.
`honesty`	Boolean, `TRUE` if forest was trained using honesty. Otherwise `FALSE`. Argument controls if honesty is used to train the new forests on the random half-samples, so misspecification will lead to invalid results. Default is `TRUE`, the default in causal_forest.

`stabilize.splits`	Boolean, `TRUE` if forest was trained taking treatment into account when determining the imbalance of a split. Otherwise `FALSE`. Argument controls if treatment is taken into account when determining the imbalance of a split during training of the new forests on the random half-samples, so misspecification will lead to invalid results. Default is `TRUE`, the default in causal_forest.
`failure.times`	For valid results, specify the value used to train `forest`.
`sample.fraction`	For valid results, specify the value used to train `forest`.
`mtry`	For valid results, specify the value used to train `forest`.
`min.node.size`	For valid results, specify the value used to train `forest`.
`honesty.fraction`	For valid results, specify the value used to train `forest`.
`honesty.prune.leaves`	For valid results, specify the value used to train `forest`.
`alpha`	For valid results, specify the value used to train `forest`.
`imbalance.penalty`	For valid results, specify the value used to train `forest`.

Details

RATE evaluates the ability of a provided prioritization rule to prioritize treatment to subjects with a large benefit. In order to test for heterogeneity, we want estimated CATE's to define the prioritization rule. However, to obtain valid inference the prioritization scores must be constructed independently of the evaluating forest training data. To accomplice this, we split the data and train separate forests on each part. Then we estimate double robust scores on the observations used to train each forest, and obtain prioritization scores by predicting CATE's with each forest on the samples not used for training.

Value

A list of class `rank_average_treatment_effect` with elements

- estimate: the RATE estimate.
- std.err: bootstrapped standard error of RATE.
- target: the type of estimate.
- TOC: a data.frame with the Targeting Operator Characteristic curve estimated on grid q, along with bootstrapped SEs.
- confint: a data.frame with the lower and upper bounds of the RATE confidence interval.
- pval: the p-value for the test that RATE is non-positive.

Author(s)

KIJA

References

Yadlowsky S, Fleming S, Shah N, Brunskill E, Wager S. Evaluating Treatment Prioritization Rules via Rank-Weighted Average Treatment Effects. 2021. http://arxiv.org/abs/2111.07966.

Examples

```
n <- 2000
p <- 5
X <- matrix(rnorm(n * p), n, p)
X_surv <- matrix(runif(n * p), n, p)
W <- rbinom(n, 1, 0.5)
event_prob <- 1 / (1 + exp(2 * (pmax(2 * X[, 1], 0) * W - X[, 2])))
horizon <- 1
failure.time <- pmin(rexp(n) * X_surv[, 1] + W, horizon)
censor.time <- 2 * runif(n)
D <- as.integer(failure.time <= censor.time)
Y <- rbinom(n, 1, event_prob)
Y_surv <- pmin(failure.time, censor.time)
clusters <- sample(1:4, n, replace = TRUE)
cf <- grf::causal_forest(X, Y, W, clusters = clusters)
csf <- grf::causal_survival_forest(X, round(Y_surv, 2), W, D, horizon = horizon)
rate_cf <- RATEOmnibusTest(cf, target = "QINI")
rate_csf <- RATEOmnibusTest(csf, target = "QINI")
rate_cf
rate_csf
```

`RATETest` *wrapper for rank_average_treatment_effect*

Description

Provides confidence interval and p-value together with the standard output from rank_average_treatment_effect.

Usage

```
RATETest(
  forest,
  priorities,
  level = 0.95,
  cov_type = c("continuous", "discrete"),
  target = c("AUTOC", "QINI"),
  q = seq(0.1, 1, by = 0.1),
  R = 500,
  subset = NULL,
  debiasing.weights = NULL,
  compliance.score = NULL,
  num.trees.for.weights = 500
)
```

Arguments

`forest`	An object of class `causal_forest`, as returned by causal_forest().
`priorities`	character, name of covariate to test for heterogeneity.
`level`	numeric, level of RATE confidence interval.
`cov_type`	character, either "continuous" or "discrete". If "discrete", and q is not manually set, TOC will be evaluated at the quantiles corresponding to transitions from one level to the next.
`target`	character, see rank_average_treatment_effect.
`q`	numeric, see rank_average_treatment_effect.
`R`	integer, see rank_average_treatment_effect.
`subset`	numeric, see rank_average_treatment_effect.
`debiasing.weights`	numeric, see rank_average_treatment_effect.
`compliance.score`	numeric, see rank_average_treatment_effect.
`num.trees.for.weights`	integer, see rank_average_treatment_effect.

Value

A list of class 'rank_average_treatment_effect' with elements

- estimate: the RATE estimate.
- std.err: bootstrapped standard error of RATE.
- target: the type of estimate.
- TOC: a data.frame with the Targeting Operator Characteristic curve estimated on grid q, along with bootstrapped SEs.
- confint: a data.frame with the lower and upper bounds of the RATE confidence interval.
- pval: the p-value for the test that RATE is non-positive.

Author(s)

KIJA

Examples

```
n <- 800
p <- 3
X <- matrix(rnorm(n * p), n, p)
W <- rbinom(n, 1, 0.5)
event_prob <- 1 / (1 + exp(2 * (pmax(2 * X[, 1], 0) * W - X[, 2])))
Y <- rbinom(n, 1, event_prob)
cf <- grf::causal_forest(X, Y, W)
rate <- RATETest(cf, 1)
rate$pval
```

summary.svy_vglm *Summary function for svy_vglm objects*

Description

Internal summary function for svy_vglm objects

Usage

```
## S3 method for class 'svy_vglm'
summary(object, ...)
```

Arguments

`object`	An svy_vglm object
`...`	additional arguments. Not used.

Value

A "summary.svy_vglm" object is returned.

try_catch_warnings *Try Catch with Warning Handling*

Description

Try Catch with Warning Handling

Usage

```
try_catch_warnings(expr, character = FALSE)
```

Arguments

`expr`	An expression to be evaluated.
`character`	A logical indicating if the returned error and warning should be characters (`character` = `TRUE`) or language (`character` = `FALSE`).

Value

The `try_catch_warnings()` funciton returns a list with three elements

- `values` is the evaluated `expr` or `NULL` if the evaluations throws an error.
- `warning` is any warning given while evaluating `expr`. When `character` = `FALSE`, the default, `warning` is a simpleWarning, otherwise it is a character.
- `error` is any error given while trying to evaluate `expr`. When `character` = `FALSE`, the default, `error` is a simpleError, otherwise it is a character.

Examples

```
# No errors or warnings
try_catch_warnings(log(2))

# Warnings
try_catch_warnings(log(-1))

# Errors
try_catch_warnings(stop("Error Message"))
try_catch_warnings(stop("Error Message"), character = TRUE)
```

---

vcovHC *Heteroscedasticity-Consistent Covariance Matrix*

Description

Calculate Heteroscedasticity-Consistent Covariance Matrix from a linear model using the HC3 method from sandwich.

Usage

```
vcovHC(x)
```

Arguments

x	lm object

Value

A matrix containing the covariance matrix estimate.

Examples

```
1+1
```

Index