Package 'causalWins'

May 8, 2026

Title Compute the Causal Win Ratio Using Nearest Neighbor Matching

Version 0.1.0

Description Based on "Rethinking the Win Ratio: A Causal Framework for Hierarchical Outcome Analysis" (M. Even and J. Josse, 2025), this package provides implementations of three approaches - nearest neighbor matching, distributional regression forests, and efficient influence functions - to estimate the causal win ratio, win proportion, and net benefit.

Encoding UTF-8

License AGPL (>= 3)

RoxygenNote 7.3.3

Imports drf, FactoMineR, grf, MatchIt

Suggests knitr, rmarkdown, WINS, MASS

VignetteBuilder knitr

NeedsCompilation no

Author Francisco Andrade [aut],
Mathieu Even [aut, cre],
Julie Josse [aut]

Maintainer Mathieu Even <mathieu.even@inria.fr>

Repository CRAN

Date/Publication 2026-04-21 18:22:13 UTC

Contents

drf_win_ratio *Win/Loss Ratios with Distributional Random Forests*

Description

drf_win_ratio() Computes the win/loss proportion, win ratio, and net benefit by estimating nuisance parameters with machine learning methods. This estimation is performed via distributional regression, implemented using distributional random forests.

Usage

```
drf_win_ratio(
  base,
  treatment_name,
  outcome_names,
  win_function = "lexicographic",
  n_trees_drf = 1000,
  n_of_folds = 3,
  seed = NULL
)
```

Arguments

base	A data frame with n rows (individuals) and p columns. It must include a treatment column and one or more outcome columns. All remaining columns are considered covariates for matching. The data frame should not contain missing values. All covariates must be either numeric or factors.
treatment_name	A character vector of length one for the treatment column name, must be in names(base). The treatment column should contain only zeros and ones, with ones indicating the treatment group.
outcome_names	A character vector specifying the names of outcome columns. All names must exist in names(base). Each outcome column should be numeric or binary and binary outcomes must be encoded as 0 and 1.
win_function	By default, this function compares outcomes lexicographically according to the column order in the data frame, treating higher values as better. Once a distinct outcome is found, the function yields 1 (a win) if that outcome is larger for the treated individual and 0 otherwise; in case of a tie, it yields 0. To change the priority of outcomes, simply reorder the columns of the data frame. Alternatively, win_function can be a custom function that takes two arguments, each corresponding to a row of the data frame: win_function(base[i,], base[j,]) where the first argument represents a treated individual and the second a control individual. The function must return a value between 0 and 1, with 1 indicating a win for the treated individual.
n_trees_drf	Number of trees grown in the forest. It passed as the argument num.trees to drf. The default value is 1000.
n_of_folds	Number of folds used in cross-fitting. The default value is 3.

seed — Numeric value used in `set.seed()` to guarantee reproducible results when sampling the folds. The default value is `NULL`.

Value

A list of four elements, each of which is a list structured as follows:

The first element, `win_proportion`, is a list with two components: `value` and `std`, where `std` represents the standard deviation.

The second element, `lose_proportion`, is a list with two components: `value` and `std`, where `std` represents the standard deviation.

The third element, `win_ratio`, is a list containing three components: `value`, `lower_CI`, and `upper_CI`, with `lower_CI` and `upper_CI` representing the confidence interval bounds.

The fourth element, `net_benefit`, is a list containing three components: `value`, `lower_CI`, and `upper_CI`, with `lower_CI` and `upper_CI` representing the confidence interval bounds.

The fifth and sixth elements, win_vector and lose_vector, are numeric vectors containing the nuisance function values evaluated for each individual.

Examples

```
## Mixed Covariates ##

set.seed(123)
n <- 100
base <- data.frame(
  X1 = rnorm(n, mean = 5, sd = 2), # numeric covariate 1
  X2 = rnorm(n, mean = 10, sd = 3), # numeric covariate 2
  X3 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  Y1 = rnorm(n, mean = 50, sd = 10), # numeric outcome 1
  Y2 = rnorm(n, mean = 100, sd = 20), # numeric outcome 2
  arm = sample(c(0, 1), n, replace = TRUE) # binary treatment arm
)
treatment_name <- "arm"
outcome_names <- c("Y1", "Y2")
res <- drf_win_ratio(
  base = base,
  treatment_name = treatment_name,
  outcome_names = outcome_names
)
## Categorical Covariates ##

set.seed(123)
n <- 100
base <- data.frame(
  X1 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  X2 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  X3 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  Y1 = rnorm(n, mean = 50, sd = 10), # numeric outcome 1
  Y2 = rnorm(n, mean = 100, sd = 20), # numeric outcome 2
  arm = sample(c(0, 1), n, replace = TRUE) # binary treatment arm
)
```

```
treatment_name <- "arm"
outcome_names <- c("Y1", "Y2")
res <- drf_win_ratio(
  base = base,
  treatment_name = treatment_name,
  outcome_names = outcome_names
)
```

`eif_win_ratio` *Win/Loss Ratios with Efficient Influence Functions*

Description

`eif_win_ratio()` calculates the win/loss proportion, win ratio, and net benefit by adding a correction term to `drf_win_ratio()` using Efficient Influence Functions.

Usage

```
eif_win_ratio(
  base,
  treatment_name,
  outcome_names,
  win_function = "lexicographic",
  distance = NULL,
  n_trees_drf = 1000,
  n_of_folds = 3,
  seed = NULL,
  propensity_model = "probability_forest",
  n_trees_propensity = 1000,
  propensity_scores = NULL,
  clip_propensity = 0
)
```

Arguments

`base`	A data frame with n rows (individuals) and p columns. It must include a treatment column and one or more outcome columns. All remaining columns are considered covariates for matching. The data frame should not contain missing values. All covariates must be either numeric or factors.
`treatment_name`	A character vector of length one for the treatment column name, must be in `names(base)`. The treatment column should contain only zeros and ones, with ones indicating the treatment group.
`outcome_names`	A character vector specifying the names of outcome columns. All names must exist in `names(base)`. Each outcome column should be numeric or binary and binary outcomes must be encoded as 0 and 1.

`win_function` By default, this function compares outcomes lexicographically according to the column order in the data frame, treating higher values as better. Once a distinct outcome is found, the function yields 1 (a win) if that outcome is larger for the treated individual and 0 otherwise; in case of a tie, it yields 0. To change the priority of outcomes, simply reorder the columns of the data frame. Alternatively, `win_function` can be a custom function that takes two arguments, each corresponding to a row of the data frame: `win_function(base[i, ], base[j, ])` where the first argument represents a treated individual and the second a control individual. The function must return a value between 0 and 1, with 1 indicating a win for the treated individual.

`distance` A function that is used to compute the nearest neighbor matching on covariates. The distance function will be passed as an argument to `matchit` (see `distance`).

Default Distance Settings

The default value for the distance parameter is `NULL`. If no distance is provided, the function assigns defaults based on the type of covariates:

- **Mixed covariates:** Applies Factor Analysis of Mixed Data transformation (see `FAMD`) to the covariates and sets distance to `"euclidean"`.
- **Numeric covariates:** Sets distance to `"euclidean"`.
- **Categorical covariates:** Applies Multiple Correspondence Analysis (see `MCA`) transformation to the covariates and sets distance to `"euclidean"`.

`n_trees_drf` Number of trees grown in the distributional random forests. It passed as the argument `num.trees` to `drf`. The default value is 1000.

`n_of_folds` Number of folds used in cross-fitting. The default value is 3.

`seed` Numeric value used in `set.seed()` to guarantee reproducible results when sampling the folds. The default value is `NULL`.

`propensity_model`
Character string specifying the type of model to use. Must be either "glm" or "probability_forest". Defaults to "probability_forest". The model is only used if no vector of propensity scores (`propensity_scores`) is provided.

`n_trees_propensity`
Number of trees grown when computing the propensity scores using `propensity_model`.

`propensity_scores`
Numeric vector of propensity score probabilities

`clip_propensity`
Numeric value used to clip propensity scores, ensuring they lie within the interval `[clip_propensity, 1 - clip_propensity]`. The default value is zero.

Value

A list of six elements, each of which is a list structured as follows:

The first element, `win_proportion`, is a list with two components: `value` and `std`, where `std` represents the standard deviation.

The second element, `lose_proportion`, is a list with two components: `value` and `std`, where `std` represents the standard deviation.

The third element, `win_ratio`, is a list containing three components: `value`, `lower_CI`, and `upper_CI`, with `lower_CI` and `upper_CI` representing the confidence interval bounds.

The fourth element, `net_benefit`, is a list containing three components: `value`, `lower_CI`, and `upper_CI`, with `lower_CI` and `upper_CI` representing the confidence interval bounds.

The fifth and sixth elements, `win_vector` and `lose_vector`, are numeric vectors containing the plug-in estimator evaluated for each individual, along with the corresponding EIF correction term.

Examples

```
## Input Propensity Scores ##

set.seed(123)
n <- 100
base <- data.frame(
  X1 = rnorm(n, mean = 5, sd = 2), # numeric covariate 1
  X2 = rnorm(n, mean = 10, sd = 3), # numeric covariate 2
  X3 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  Y1 = rnorm(n, mean = 50, sd = 10), # numeric outcome 1
  Y2 = rnorm(n, mean = 100, sd = 20), # numeric outcome 2
  arm = sample(c(0, 1), n, replace = TRUE) # binary treatment arm
)
treatment_name <- "arm"
outcome_names <- c("Y1", "Y2")

ps_model <- stats::glm(arm ~ X1 + X3, # only consider X1 and X3 and no cross-fitting
  family = binomial(),
  data = base
)

propensity_scores <- stats::predict(ps_model, type = "response")
res <- eif_win_ratio(
  base = base,
  treatment_name = treatment_name,
  outcome_names = outcome_names,
  propensity_scores = propensity_scores
)
```

`nn_win_ratio` *Win/Loss Ratios with Nearest Neighbor Matching*

Description

`nn_win_ratio()` calculates the win/loss proportion, win ratio, and net benefit by performing nearest-neighbor matching between individuals in the treatment group and those in the control group based on a specified ground distance. After matching, the outcomes of paired individuals are compared to determine winners and losers.

Usage

```
nn_win_ratio(
  base,
  treatment_name,
  outcome_names,
  win_function = "lexicographic",
  distance = NULL
)
```

Arguments

base	A data frame with n rows (individuals) and p columns. It must include a treatment column and one or more outcome columns. All remaining columns are considered covariates for matching. The data frame should not contain missing values. All covariates must be either numeric or factors.
treatment_name	A character vector of length one for the treatment column name, must be in `names(base)`. The treatment column should contain only zeros and ones, with ones indicating the treatment group.
outcome_names	A character vector specifying the names of outcome columns. All names must exist in `names(base)`. Each outcome column should be numeric or binary and binary outcomes must be encoded as 0 and 1.
win_function	By default, this function compares outcomes lexicographically according to the column order in the data frame, treating higher values as better. Once a distinct outcome is found, the function yields 1 (a win) if that outcome is larger for the treated individual and 0 otherwise; in case of a tie, it yields 0. To change the priority of outcomes, simply reorder the columns of the data frame. Alternatively, `win_function` can be a custom function that takes two arguments, each corresponding to a row of the data frame: `win_function(base[i, ], base[j, ])` where the first argument represents a treated individual and the second a control individual. The function must return a value between 0 and 1, with 1 indicating a win for the treated individual.
distance	A function that is used to compute the nearest neighbor matching on covariates. The distance function will be passed as an argument to `matchit` (see `distance`). **Default Distance Settings** The default value for the distance parameter is `NULL`. If no distance is provided, the function assigns defaults based on the type of covariates: • **Numeric covariates:** Sets distance to `"euclidean"`. • **Categorical covariates:** Applies Multiple Correspondence Analysis (see `MCA`) transformation to the covariates and sets distance to `"euclidean"`. • **Mixed covariates:** Applies Factor Analysis of Mixed Data transformation (see `FAMD`) to the covariates and sets distance to `"euclidean"`.

Value

A list of six elements, each of which is a list structured as follows:

The first element, `win_proportion`, is a list with two components: `value` and `std`, where `std` represents the standard deviation.

The second element, `lose_proportion`, is a list with two components: `value` and `std`, where `std` represents the standard deviation.

The third element, `win_ratio`, is a list containing three components: `value`, `lower_CI`, and `upper_CI`, with `lower_CI` and `upper_CI` representing the confidence interval bounds.

The fourth element, `net_benefit`, is a list containing three components: `value`, `lower_CI`, and `upper_CI`, with `lower_CI` and `upper_CI` representing the confidence interval bounds.

The fifth and sixth elements, `win_vector` and `lose_vector`, are numeric vectors representing the number of wins and losses for each matched element, respectively. Their entries follow the same order as the rows in `base`.

Examples

```
## Mixed Covariates ##

set.seed(123)
n <- 100
base <- data.frame(
  X1 = rnorm(n, mean = 5, sd = 2), # numeric covariate 1
  X2 = rnorm(n, mean = 10, sd = 3), # numeric covariate 2
  X3 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  Y1 = rnorm(n, mean = 50, sd = 10), # numeric outcome 1
  Y2 = rnorm(n, mean = 100, sd = 20), # numeric outcome 2
  arm = sample(c(0, 1), n, replace = TRUE) # binary treatment arm
)
treatment_name <- "arm"
outcome_names <- c("Y1", "Y2")
res <- nn_win_ratio(
  base = base,
  treatment_name = treatment_name,
  outcome_names = outcome_names
)

## Categorical Covariates ##

set.seed(123)
n <- 100
base <- data.frame(
  X1 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  X2 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  X3 = factor(sample(c("A", "B", "C"), n, replace = TRUE)), # factor covariate
  Y1 = rnorm(n, mean = 50, sd = 10), # numeric outcome 1
  Y2 = rnorm(n, mean = 100, sd = 20), # numeric outcome 2
  arm = sample(c(0, 1), n, replace = TRUE) # binary treatment arm
)
treatment_name <- "arm"
outcome_names <- c("Y1", "Y2")
res <- nn_win_ratio(
  base = base,
  treatment_name = treatment_name,
```

```
  outcome_names = outcome_names
)

## Input a Distance Matrix ##

set.seed(123)
n <- 100
base <- data.frame(
  X1 = rnorm(n, mean = 5, sd = 2), # numeric covariate 1
  X2 = rnorm(n, mean = 5, sd = 2), # numeric covariate 2
  X3 = rnorm(n, mean = 5, sd = 2), # numeric covariate 3
  Y1 = rnorm(n, mean = 50, sd = 10), # numeric outcome 1
  Y2 = rnorm(n, mean = 100, sd = 20), # numeric outcome 2
  arm = sample(c(0, 1), n, replace = TRUE) # binary treatment arm
)
treatment_name <- "arm"
outcome_names <- c("Y1", "Y2")

covariate_names <- setdiff(colnames(base), c(outcome_names, treatment_name))
covariate_data <- base[, covariate_names, drop = FALSE]
distance_matrix <- as.matrix(dist(covariate_data, method = "euclidean"))

res_dist <- nn_win_ratio(
  base = base,
  treatment_name = treatment_name,
  outcome_names = outcome_names,
  distance = distance_matrix
)

res <- nn_win_ratio(
  base = base,
  treatment_name = treatment_name,
  outcome_names = outcome_names
)

res <- unlist(res)
res_dist <- unlist(res_dist)

print(all.equal(res, res_dist))

## Input a Win function ##

set.seed(123)
n <- 100
base <- data.frame(
  X1 = rnorm(n, mean = 5, sd = 2), # numeric covariate 1
  X2 = rnorm(n, mean = 5, sd = 2), # numeric covariate 2
  X3 = rnorm(n, mean = 5, sd = 2), # numeric covariate 3
  Y1 = rbinom(n, size = 1, prob = 0.5), # binary outcome 1 (0/1) with 50% chance
  Y2 = rbinom(n, size = 10, prob = 0.7), # counts out of 10 trials with 70% success rate
  arm = sample(c(0, 1), n, replace = TRUE) # binary treatment arm
)
treatment_name <- "arm"
```

```
outcome_names <- c("Y1", "Y2")

# Contrary to the default win function, which yields 0 for ties,
# the following win function yields 0.5.

win_function <- function(treated_row, control_row) {
  y <- as.matrix(treated_row[, outcome_names], drop = FALSE)
  z <- as.matrix(control_row[, outcome_names], drop = FALSE)
  if (all(y == z)) {
    return(0.5) # Tie, a zero in the default win function
  }
  result <- (y > z)[which.max(y != z)] * 1
  return(result)
}

res <- nn_win_ratio(
  base = base,
  treatment_name = treatment_name,
  outcome_names = outcome_names,
  win_function = win_function
)
```

Index