Package ‘OutcomeWeights’

January 20, 2025

Type Package

Title Outcome Weights of Treatment Effect Estimators

Version 0.1.1

Description Many treatment effect estimators can be written as weighted outcomes.
These weights have established use cases like checking covariate balancing via packages like 'cobalt'.
This package takes the original estimator objects and outputs these outcome weights.
It builds on the general framework of Knaus (2024) <doi:10.48550/arXiv.2411.11559>.
This version is compatible with the 'grf' package and provides an internal implementation of Double Machine Learning.

License GPL-3

Encoding UTF-8

URL `https://github.com/MCKnaus/OutcomeWeights`

BugReports `https://github.com/MCKnaus/OutcomeWeights/issues`

Imports ggplot2, grf, methods

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Michael C. Knaus [aut, cre] (<`https://orcid.org/0000-0002-7328-1363`>),
Henri Pfleiderer [ctb]

Maintainer Michael C. Knaus <`michael.knaus@uni-tuebingen.de`>

Repository CRAN

Date/Publication 2024-12-20 10:10:02 UTC

Contents

`dml_with_smoother` *Double ML estimators with outcome smoothers*

Description

Existing Double ML implementations are too general to easily extract smoother matrices required to be compatible with the get_forest_weights() method. This motivates yet another Double ML implementation.

Usage

```
dml_with_smoother(
  Y,
  D,
  X,
  Z = NULL,
  estimators = c("PLR", "PLR_IV", "AIPW_ATE", "Wald_AIPW"),
  smoother = "honest_forest",
  n_cf_folds = 5,
  n_reps = 1,
  ...
)
```

Arguments

Y	Numeric vector containing the outcome variable.
D	Optional binary treatment variable.
X	Covariate matrix with N rows and p columns.
Z	Optional binary instrumental variable.

`estimators` String (vector) indicating which estimators should be run. Current menu: c("PLR","PLR_IV","AIPW_AT

`smoother` Indicate which smoother to be used for nuisance parameter estimation. Currently only available option `"honest_forest"` from the **grf** package.

`n_cf_folds` Number of cross-fitting folds. Default is 5.

`n_reps` Number of repetitions of cross-fitting. Default is 1.

`...` Options to be passed to smoothers.

Value

A list with three entries:

- `results`: a list storing the results, influence functions, and score functions of each estimator
- `NuPa.hat`: a list storing the estimated nuisance parameters and the outcome smoother matrices

References

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., & Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. The Econometrics Journal, 21(1), C1-C68.

Knaus, M. C. (2024). Treatment effect estimators as weighted outcomes, https://arxiv.org/abs/2411.11559.

Examples

```
# Sample from DGP borrowed from grf documentation
n = 200
p = 5
X = matrix(rbinom(n * p, 1, 0.5), n, p)
Z = rbinom(n, 1, 0.5)
Q = rbinom(n, 1, 0.5)
W = Q * Z
tau =  X[, 1] / 2
Y = rowSums(X[, 1:3]) + tau * W + Q + rnorm(n)

# Run outcome regression and extract smoother matrix
# Run DML and look at results
dml = dml_with_smoother(Y,W,X,Z)
results_dml = summary(dml)
plot(dml)

# Get weights
omega_dml = get_outcome_weights(dml)

# Observe that they perfectly replicate the original estimates
all.equal(as.numeric(omega_dml$omega %*% Y),
          as.numeric(as.numeric(results_dml[,1])))

# The weights can then be passed to the cobalt package for example.
```

`get_outcome_weights` *Outcome weights method*

Description

This is a generic method for getting outcome weights. It calculates the outcome weights for objects created by other packages. See get_outcome_weight.<compatible_fct> in the package documentation for compatible functions.

Usage

```
get_outcome_weights(object, ...)
```

Arguments

`object`	An object, obtained from other packages.
`...`	Additional arguments specific to object class implementations. See the documentation which object requires which additional arguments.

Value

A list of at least these components:

- omega: matrix (number of point estimates x number of estimation units) of outcome weights
- treat: the treatment indicator to make it compatible with the cobalt package

References

Knaus, M. C. (2024). Treatment effect estimators as weighted outcomes, `https://arxiv.org/abs/2411.11559`.

`get_outcome_weights.causal_forest` *Outcome weights for the `causal_forest` function*

Description

Post-estimation command to extract outcome weights for causal forest implemented via the `causal_forest` function from the **grf** package.

Usage

```
## S3 method for class 'causal_forest'
get_outcome_weights(
  object,
  ...,
  S,
  newdata = NULL,
  S.tau = NULL,
  target = "CATE",
  checks = TRUE
)
```

Arguments

`object` An object of class `causal_forest`, i.e. the result of running `causal_forest`.

`...` Pass potentially generic get_outcome_weights options.

`S` A smoother matrix reproducing the outcome predictions used in building the `instrumental_forest`. Obtained by calling `get_forest_weights()` for the `regression_forest` object producing the outcome predictions.

`newdata` Corresponds to `newdata` option in `predict.causal_forest`. If `NULL`, out-of-bag outcome weights, otherwise for those for the provided test data returned.

`S.tau` Required if `target != "CATE"`, then S.tau is the CATE smoother obtained from running `get_outcome_weights()` with `target == "CATE"`.

`target` Target parameter for which outcome weights should be extracted. Currently `c("CATE","ATE")` implemented.

`checks` Default `TRUE` checks whether weights numerically replicate original estimates. Only set `FALSE` if you know what you are doing and need to save computation time.

Value

get_outcome_weights object with omega containing weights and `treat` the treatment

References

Athey, S., Tibshirani, J., & Wager, S. (2019). Generalized random forest. The Annals of Statistics, 47(2), 1148-1178.

Knaus, M. C. (2024). Treatment effect estimators as weighted outcomes, `https://arxiv.org/abs/2411.11559`.

Examples

```
# Sample from DGP borrowed from grf documentation
n = 500
p = 10
X = matrix(rnorm(n * p), n, p)
W = rbinom(n, 1, 0.5)
```

```
Y = pmax(X[, 1], 0) * W + X[, 2] + pmin(X[, 3], 0) + rnorm(n)

# Run outcome regression and extract smoother matrix
forest.Y = grf::regression_forest(X, Y)
Y.hat = predict(forest.Y)$predictions
outcome_smoother = grf::get_forest_weights(forest.Y)

# Run causal forest with external Y.hats
c.forest = grf::causal_forest(X, Y, W, Y.hat = Y.hat)

# Predict on out-of-bag training samples.
cate.oob = predict(c.forest)$predictions

# Predict using the forest.
X.test = matrix(0, 101, p)
X.test[, 1] = seq(-2, 2, length.out = 101)
cate.test = predict(c.forest, X.test)$predictions

# Calculate outcome weights
omega_oob = get_outcome_weights(c.forest,S = outcome_smoother)
omega_test = get_outcome_weights(c.forest,S = outcome_smoother,newdata = X.test)

# Observe that they perfectly replicate the original CATEs
all.equal(as.numeric(omega_oob$omega %*% Y),
          as.numeric(cate.oob))
all.equal(as.numeric(omega_test$omega %*% Y),
          as.numeric(cate.test))

# Also the ATE estimates are perfectly replicated
omega_ate = get_outcome_weights(c.forest,target = ”ATE”,
                                S = outcome_smoother,
                                S.tau = omega_oob$omega)
all.equal(as.numeric(omega_ate$omega %*% Y),
          as.numeric(grf::average_treatment_effect(c.forest, target.sample = ”all”)[1]))

# The omega weights can be plugged into balancing packages like cobalt
```

get_outcome_weights.dml_with_smoother

Outcome weights for the `dml_with_smoother` *function*

Description

Post-estimation command to extract outcome weights for double ML run with an outcome smoother.

Usage

```
## S3 method for class 'dml_with_smoother'
get_outcome_weights(object, ..., all_reps = FALSE)
```

Arguments

`object` An object of class `dml_with_smoother`, i.e. the result of running `dml_with_smoother`.

`...` Pass potentially generic get_outcome_weights options.

`all_reps` If `TRUE`, outcomes weights of each repetitions passed. Default `FALSE`.

Value

- If `all_reps == FALSE`: get_outcome_weights object
- If `all_reps == TRUE`: additionally list `omega_all_reps`: A list containing the outcome weights of each repetition.

References

Knaus, M. C. (2024). Treatment effect estimators as weighted outcomes, `https://arxiv.org/abs/2411.11559`.

---

get_outcome_weights.instrumental_forest

Outcome weights for the `instrumental_forest` *function*

Description

Post-estimation command to extract outcome weights for instrumental forest implemented via the `instrumental_forest` function from the **grf** package.

Usage

```
## S3 method for class 'instrumental_forest'
get_outcome_weights(object, ..., S, newdata = NULL, checks = TRUE)
```

Arguments

`object` An object of class `instrumental_forest`, i.e. the result of running `instrumental_forest`.

`...` Pass potentially generic get_outcome_weights options.

`S` A smoother matrix reproducing the outcome predictions used in building the `instrumental_forest`. Obtained by calling `get_forest_weights()` for the `regression_forest` object producing the outcome predictions.

`newdata` Corresponds to `newdata` option in `predict.instrumental_forest`. If `NULL`, out-of-bag outcome weights, otherwise for those for the provided test data returned.

`checks` Default `TRUE` checks whether weights numerically replicate original estimates. Only set `FALSE` if you know what you are doing and want to save computation time.

Value

get_outcome_weights object with omega containing weights and treat the treatment

References

Athey, S., Tibshirani, J., & Wager, S. (2019). Generalized random forest. The Annals of Statistics, 47(2), 1148-1178.

Knaus, M. C. (2024). Treatment effect estimators as weighted outcomes, https://arxiv.org/abs/2411.11559.

Examples

```
# Sample from DGP borrowed from grf documentation
n = 2000
p = 5
X = matrix(rbinom(n * p, 1, 0.5), n, p)
Z = rbinom(n, 1, 0.5)
Q = rbinom(n, 1, 0.5)
W = Q * Z
tau =  X[, 1] / 2
Y = rowSums(X[, 1:3]) + tau * W + Q + rnorm(n)

# Run outcome regression and extract smoother matrix
forest.Y = grf::regression_forest(X, Y)
Y.hat = predict(forest.Y)$predictions
outcome_smoother = grf::get_forest_weights(forest.Y)

# Run instrumental forest with external Y.hats
iv.forest = grf::instrumental_forest(X, Y, W, Z, Y.hat = Y.hat)

# Predict on out-of-bag training samples.
iv.pred = predict(iv.forest)$predictions

omega_if = get_outcome_weights(iv.forest, S = outcome_smoother)

# Observe that they perfectly replicate the original CLATEs
all.equal(as.numeric(omega_if$omega %*% Y),
          as.numeric(iv.pred))
```

NuPa_honest_forest *Nuisance parameter estimation via honest random forest*

Description

This function estimates different nuisance parameters using the honest random forest implementation of the 'grf' package

Usage

```
NuPa_honest_forest(
  NuPa = c("Y.hat", "Y.hat.d", "Y.hat.z", "D.hat", "D.hat.z", "Z.hat"),
  X,
  Y = NULL,
  D = NULL,
  Z = NULL,
  n_cf_folds = 5,
  n_reps = 1,
  cluster = NULL,
  progress = FALSE,
  ...
)
```

Arguments

`NuPa` String vector specifying the nuisance parameters to be estimated. Currently supported: `c("Y.hat","Y.hat.d","Y.hat.z","D.hat","D.hat.z","Z.hat")`

`X` Covariate matrix with N rows and p columns.

`Y` Optional numeric vector containing the outcome variable.

`D` Optional binary treatment variable.

`Z` Optional binary instrumental variable.

`n_cf_folds` Number of cross-fitting folds. Default is 5.

`n_reps` Number of repetitions of cross-fitting. Default is 1.

`cluster` Optional vector of cluster variable if cross-fitting should account for clusters.

`progress` If TRUE, progress of nuisance parameter estimation reported.

`...` Options passed to the `regression_forest`.

Value

List of two lists.

- `predictions` contains the requested nuisance parameters
- `smoothers` contains the smoother matrices of requested outcome nuisance parameters
- `cf_mat` Array of dimension n_reps x N x n_cf_folds storing indicators representing the folds used in estimation.

References

Wager, S., & Athey, S. (2018). Estimation and inference of heterogeneous treatment effects using random forests. Journal of the American Statistical Association, 113(523), 1228-1242.

pive_weight_maker *Outcome weights maker for pseudo-IV estimators.*

Description

This is a generic function taking pseudo-instrument, pseudo-treatment and the transformation matrix as inputs and returning outcome weights

Usage

```
pive_weight_maker(Z.tilde, D.tilde, T_mat)
```

Arguments

Z.tilde	Numeric vector of pseudo-instrument outcomes.
D.tilde	Numeric vector of pseudo-treatment.
T_mat	Transformation matrix

Value

A vector of outcome weights.

References

Knaus, M. C. (2024). Treatment effect estimators as weighted outcomes, soon on 'arXiv'.

plot.dml_with_smoother plot *method for class* dml_with_smoother

Description

plot method for class dml_with_smoother

Usage

```
## S3 method for class 'dml_with_smoother'
plot(x, ..., alpha = 0.05, contrast = FALSE)
```

Arguments

x	Object of class dml_with_smoother.
...	Pass generic plot options.
alpha	Significance level for confidence intervals (default 0.05).
contrast	Shows the differences between the coefficients.

Value

ggplot with point estimates and confidence intervals.

`prep_cf_mat` *Creates matrix of binary cross-fitting fold indicators (N x # cross-folds)*

Description

Creates matrix of binary cross-fitting fold indicators (N x # cross-folds)

Usage

```
prep_cf_mat(n, cf, w_mat = NULL, cl = NULL)
```

Arguments

`n` Number of observations.

`cf` Number of cross-fitting folds.

`w_mat` Optional logical matrix of treatment indicators (N x T+1). If specified, cross-fitting folds will preserve the treatment ratios from full sample.

`cl` Optional vector of cluster variable if cross-fitting should account for clusters.

Value

Logical matrix of cross-fitting folds (N x # folds).

`standardized_mean_differences` *Calls C++ implementation to calculate standardized mean differences.*

Description

Calculates standardized mean differences between treated and controls and towards target means for an outcome weights matrix with potentially many rows like for CATEs.

Usage

```
standardized_mean_differences(X, treat, omega, target = NULL)
```

Arguments

`X` Covariate matrix with N rows and p columns.

`treat` Binary treatment variable.

`omega` Outcome weights matrix with dimension number of weight vectors for which balancing should be checked x number of training units.

`target` Optional matrix with dimension number of weight vectors for which balancing should be checked x p indicating the target values the covariates should be balanced towards. If NULL, average of X used as target of ATE.

Value

3D-array of dimension p x 6 x number of weight vectors for which balancing should be checked where the second dimension provides the following quantities:

- "Mean 0": The weighted control mean
- "Mean 1": The weighted treated mean
- "SMD balancing": Standardized mean differences for covariate balancing (Mean 1 - Mean 0) / sd(X)
- "SMD targeting 0": Standardized mean difference to assess targeting of control (Mean 0 - target) / sd(X)
- "SMD targeting 1": Standardized mean difference to assess targeting of treated (Mean 1 - target) / sd(X)

References

Rosenbaum, P. R., & Rubin, D. B. (1984). Reducing bias in observational studies using subclassification on the propensity score. Journal of the American Statistical Association, 79 (387), 516–524.

`summary.dml_with_smoother`

`summary` *method for class* `dml_with_smoother`

Description

`summary` method for class `dml_with_smoother`

Usage

```
## S3 method for class 'dml_with_smoother'
summary(object, contrast = FALSE, quiet = FALSE, ...)
```

Arguments

`object` Object of class `dml_with_smoother`.

`contrast` Tests the differences between the coefficients.

`quiet` If TRUE, results are passed but not printed.

`...` further arguments passed to `printCoefmat`

Value

Invisible matrix with estimator(s) in the rows and c("Estimate","SE","t","p") in the columns.

---

summary.get_outcome_weights
summary *method for class* outcome_weights

Description

Calculates several summary measures of potentially many outcome weights.

Usage

```
## S3 method for class 'get_outcome_weights'
summary(object, quiet = FALSE, digits = 4, epsilon = 1e-04, ...)
```

Arguments

object	get_outcome_weights object.
quiet	If TRUE, results are passed but not printed.
digits	Number of digits to be displayed. Default 4.
epsilon	Threshold below which in absolute values non-zero but small values should be displayed as < ...
...	further arguments passed to printCoefmat

Value

3D-array of dimension

- c("Control","Treated") x
- number of point estimates x
- c("Minimum weight","Maximum weight","% Negative","Sum largest 10%","Sum of weights","Sum of absolute weights")

summary.standardized_mean_differences

summary *method for class* standardized_mean_differences

Description

Calls a C++ function to quickly summarize potentially many standardized mean differences.

Usage

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

Arguments

object	Object of class standardized_mean_differences.
...	further arguments passed to summary method.

Value

3D-array of dimension

- c("Maximum absolute SMD","Mean absolute SMD","Median absolute SMD", / % of absolute SMD > 20", "# / % of absolute SMD > 10","# / % of absolute SMD > 5") x
- c("Balancing","Targeting") x
- number of weight vectors for which balancing should be checked

References

Rosenbaum, P. R., & Rubin, D. B. (1984). Reducing bias in observational studies using subclassification on the propensity score. Journal of the American Statistical Association, 79 (387), 516–524.

Index