

Package ‘causalOT’

October 12, 2022

Type Package

Title Optimal Transport Weights for Causal Inference

Version 0.1.2

Date 2022-08-30

Author Eric Dunipace [aut, cre] (<<https://orcid.org/0000-0001-8909-213X>>)

Maintainer Eric Dunipace <edunipace@mail.harvard.edu>

Description Uses optimal transport distances to find probabilistic matching estimators for causal inference.

These methods are described in Dunipace, Eric (2021) <[arXiv:2109.01991](https://arxiv.org/abs/2109.01991)>.

The package will build the weights, estimate treatment effects, and calculate confidence intervals via the methods described in the paper.

The package also supports several other methods as described in the help files.

License GPL (>= 3.0)

Imports approxOT, Matrix, matrixStats, methods, lbfgsb3c, loo, osqp, pbapply, reticulate, R6 (>= 2.4.1), Rcpp (>= 1.0.3), RSpectra, sandwich

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0),

Suggests CBPS, data.table (>= 1.12.8), rstan (>= 2.19.3), Rmosek, testthat (>= 2.1.0), knitr, rmarkdown

Biarch true

Depends R (>= 3.5.0)

Encoding UTF-8

RoxygenNote 7.2.1

LazyData true

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-09-04 14:30:12 UTC

R topics documented:

calc_weight	2
causalEffect-class	5
causalOT	6
causalWeights-class	7
confint.causalEffect	7
cost_fun	8
DataSim	10
dist.metrics	12
ESS	13
estimate_effect	14
Hainmueller	15
LaLonde	17
mean_bal	19
ot.methods	20
pph	21
PSIS	22
sampleWeights-class	24
sinkhorn	24
supported.methods	26
supported.solvers	26
Index	28

calc_weight	<i>Estimate causal weights</i>
-------------	--------------------------------

Description

Estimate causal weights

Usage

```
calc_weight(
  data,
  constraint = NULL,
  estimand = c("ATE", "ATT", "ATC", "cATE", "feasible"),
  method = supported.methods(),
  formula = NULL,
  transport.matrix = FALSE,
  grid.search = FALSE,
  ...
)
```

Arguments

data	Either a matrix, a data.frame, or a DataSim class. Arguments "balance.covariates" and "treatment.indicator" must be provided in the ... arguments if data is of class data.frame or matrix.
constraint	The constraints or penalties for the weights. See details.
estimand	The estimand of interest. One of "ATT", "ATC", or "ATE".
method	The method to estimate the causal weights. Must be one of the methods returned by supported.methods() .
formula	The formula for creating the design matrix used in various methods. See details.
transport.matrix	Should the method calculate the transportation matrix if not done as a part of the method (TRUE/FALSE)? Default is FALSE.
grid.search	Should hyperparameters be selected by a grid search? Only available for "SBW" and "COT"/"Wasserstein" methods.
...	Many additional arguments are possible depending on the chosen method. See details for more information. Arguments "balance.covariates" and "treatment.indicator" must be provided if data is of class data.frame or matrix.

Details

We detail some of the particulars of the function arguments below.

data:

The following classes are recognized by the data variable.

DataSim class:

The DataSim class is provided by this package for simulations. You can pass a DataSim object (once data has been simulated) to this function and it will be recognized and handled appropriately.

data.frame or matrix:

If the data argument is of class data.frame or matrix, then additional arguments are necessary to pass in the dots (...). These *must* include a vector argument balance.covariates and an integer or character in the treatment.indicator argument. The balance.covariates argument should be either an integer vector giving the column numbers of the covariates to balance or a character vector giving the names of the columns to balance. Similarly, the treatment.indicator argument should be a integer giving the column number of the treatment labels or a character giving the column name.

Constraints:

The constraint argument is used by the balancing methods like "SBW". This will specify a tolerance for basis function balance.

If method "COT"/"Wasserstein" is used, will specify the penalty parameter to put on the weights. For "ATT" and "ATC" estimands, must be of the form `list(penalty = ###)`, while for estimand "ATE", must be a list of length 2 specifying penalty first for the controls and then for treated: `list(list(penalty = ###), list(penalty = ###))`.

This argument is not needed if grid.search is TRUE.

Formula:

For methods "SBW" or "COT", should be a formula object or character without a response but with the covariate functions desired. e.g., "~." includes all covariates without transformation.

For methods "Logistic" and "Probit", a propensity score model either as a formula object or character: "z ~.".

Additional arguments in . . . :

In addition to the already mentioned arguments, there are several additional optional arguments for the method "COT".

- `p`. The power of the Wasserstein distance to use.
- `metric`. The metric to use for the ground cost function. See `dist.metrics()` for supported distance metrics.
- `penalty`. What type of penalty should be used on the weights? Must be one of "entropy" or "L2".
- `add.divergence`. TRUE or FALSE. If TRUE, `penalty` defaults to entropy. and will calculate the Sinkhorn divergence version of Causal Optimal Transport. If choosing Sinkhorn divergences, the Python package `geomLoss` must be installed.
- `balance.constraints`. The tolerance for the balancing basis function methods.
- `cost`. If the cost matrix is already calculated, you can supply this to potentially save time.

Additionally, methods like "SBW" and "COT" need the specification of a solver function if using balancing functions, i.e. if the `formula` argument is specified.

- `solver`. Should be one of "mosek" or "osqp".

Value

An object of class `causalWeights`

See Also

`estimate_effect()`

Examples

```
set.seed(23483)
n <- 2^7
p <- 6
overlap <- "low"
design <- "A"
estimate <- "ATE"
#### get simulation functions ####
data <- causalOT::Hainmueller$new(n = n, p = p,
  design = design, overlap = overlap)
data$gen_data()
weights <- calc_weight(data = data,
  p = p,
  estimand = estimate,
  method = "NNM")
## Not run:
```

```

# Needs Python package GeomLoss
COTweights <- calc_weight(data = data,
  p = 2,
  constraint = list(list(penalty = 1000),
    list(penalty = 10000)),
  estimand = estimate,
  method = "COT",
  penalty = "entropy",
  add.divergence = TRUE,
  verbose = TRUE
)
# with basis function balancing.
COTweightsBF <- calc_weight(data = data,
  p = 2,
  constraint = list(list(penalty = 1000),
    list(penalty = 10000)),
  estimand = estimate,
  method = "COT",
  penalty = "entropy",
  add.divergence = TRUE,
  formula = "~.",
  balance.constraints = 0.2,
  solver = "osqp",
  verbose = TRUE
)

## End(Not run)

```

causalEffect-class *causalEffect class*

Description

causalEffect class

Details

The `variance.components` slot is a list with slots

- `E_Y1`: The mean if the target population had all been treated.
- `E_Y0`: The mean if the target population had all received control
- `E_Y1_X`: The predicted conditional mean if the target population had all been treated.
- `E_Y0_X`: The predicted conditional mean if the target population had all received control.

Note that for "ATT" and "ATC" estimands, `E_Y1_X` or `E_Y0_X` will be NA, respectively.

Meanwhile, the `options` slot is a list with slots

- `hajek`: Were weights normalized to sum to 1 (TRUE/FALSE)
- `doubly.robust`: Was an augmented estimator used? (TRUE/FALSE)

- `matched`: Was barycentric projection estimator used? (TRUE/FALSE)
- `split.model`: Was the outcome model calculated separately in each treatment group? (TRUE/FALSE)
- `balance.covariates`: The covariates selected for balance or in the outcome model in slot `data`
- `treatment.indicator`: The column that is the treatment indicator in slot `data`
- `outcome`: The columns that is the outcome in slot `data`
- `addl.args`: Any additional arguments passed in the dots (...) of `estimate_effect()`.

Slots

`estimate` The estimated treatment effect.

`data` The original data as a `data.frame`.

`model` The function used as the outcome model.

`formula` The formula for the outcome model.

`weights` The weights as an object of class `causalWeights`

`estimand` A character denoting the estimand targeted by the weights. One of "ATT", "ATC", or "ATE".

`variance.components` Objects for the asymptotic variance calculation designed so expensive models don't have to be re-fit.

`options` A list with the arguments from the `estimate_effect` function. See details.

`call` The call from the `estimate_effect()` function.

causalOT

An R package to perform causal inference using optimal transport distances.

Description

R code to perform causal inference weighting using a variety of methods and optimizers. The code can estimate weights, estimate treatment effects, and also give variance estimates. These methods are described in Dunipace, Eric (2021) <https://arxiv.org/abs/2109.01991>.

Author(s)

Eric Dunipace

causalWeights-class *causalWeights class*

Description

causalWeights class

Slots

w0 A slot with the weights for the control group.

w1 The weights for the treated group.

gamma The transportation matrix. If estimand is "ATE", will be a list with the transportation plan for each treatment group to balance towards the overall treatment.

estimand A character denoting the estimand targeted by the weights. One of "ATT", "ATC", or "ATE".

method A character denoting the method used to estimate the weights.

args The other arguments used to construct the weights.

confint.causalEffect *Confidence Intervals for Causal Effects*

Description

Confidence Intervals for Causal Effects

Usage

```
## S3 method for class 'causalEffect'
confint(
  object,
  parm,
  level = 0.95,
  method = c("asymptotic", "bootstrap", "jackknife"),
  ...
)
```

Arguments

object	An object of class causalEffect
parm	Unused. Included to match forms of other confint functions
level	Confidence level. Should be between 0 and 1. Default is 0.95.

method	How to calculate the confidence interval. Choices are "bootstrap" for a bootstrap confidence interval, "asymptotic" for "asymptotic" confidence intervals, and "jackknife" for jackknife confidence intervals. Default is "asymptotic" since it is faster.
...	Additional arguments if method is "bootstrap". Can include <ul style="list-style-type: none"> • <code>n.boot</code>. How many bootstrap samples should be used. Default is 1000. • <code>boot.method</code>. One of "n-out-of-n" or "m-out-of-n". Optimal transport methods default to "m-out-of-n". • <code>verbose</code>. Should a progress bar be printed? (TRUE/FALSE) Defaults to FALSE.

Value

A list with slots "CI" giving the confidence bounds and "SD" giving estimates of the standard error of the causal effects. If method is "bootstrap" and `boot.method` is "m-out-of-n", then there will also be a slot named "unadjusted" giving the unadjusted confidence interval and standard error estimate for reference.

Examples

```
# set-up data
set.seed(1234)
data <- Hainmueller$new()
data$gen_data()

# calculate quantities
weight <- calc_weight(data, method = "Logistic")
tx_eff <- estimate_effect(data = data, weights = weight)

# get asymptotic C.I.
confint(tx_eff, model = "lm", method = "asymptotic",
        formula = list(treated = "y ~ .", control = "y ~ ."))
```

cost_fun

Calculate cost matrix for a given estimand

Description

Calculate cost matrix for a given estimand

Usage

```
cost_fun(x, z, power = 2, metric = dist.metrics(), estimand = "ATE", ...)
```


Arguments

x	An object of class <code>matrix</code>
z	A treatment indicator with values in 0 and 1. Should be of class <code>integer</code> or <code>vector</code>
power	The power used to calculate the the cost matrix: $\{(x - y)^{power}\}^{(1/power)}$
metric	One of the values in dist.metrics .
estimand	The estimand desired for the weighting estimator. See details
...	Arguments passed to the RKHS calculating function including <ul style="list-style-type: none"> • <code>kernel</code>, one of "RBF", "polynomial", "linear" • <code>rkhs.args</code> The arguments used to construct the kernel <p>... can also be used to handle extra arguments passed by mistake so that an error is not thrown.</p>

Details

If the estimand is "ATT" or "ATC", `cost_fun` will calculate the cost matrix where the rows are the control and the columns are the treated. If "ATE" will calculate to cost matrices with the first having the rows corresponding to the control individual and the second having rows correspond to the treated individuals. For both matrices, the columns will correspond to the full sample. The dimensions of the output will depend on the estimand. For reference, let $n_1 = \sum_i z_i$, $n_0 = \sum_i (1 - z_i)$, and $n = n_1 + n_0$.

Value

Output depends on the estimand.

- For ATT and ATC: a matrix of dimension

$$n_0 \times n_1$$

- For ATE: a list of two matrices of dimension $n_0 \times n$ and $n_1 \times n$. See details for more information.

Examples

```
n0 <- 100
n1 <- 55
d <- 5
x1 <- matrix(stats::rnorm(n1*d), n1, d)
x0 <- matrix(stats::rnorm(n0*d), n0, d)

x <- rbind(x0,x1)
z <- c(rep(0,n0), rep(1,n1))
power <- 2.0

# ATT
estimand <- "ATT"
```

```
metric <- "Lp"
cost_ATT <- cost_fun(x, z, power = power, metric = metric, estimand = estimand)
print(dim(cost_ATT))

# ATE
estimand <- "ATE"
cost_ATT <- cost_fun(x, z, power = power, metric = metric, estimand = estimand)
length(cost_ATT)
```

DataSim

R6 Data Generating Parent Class

Description

R6 Data Generating Parent Class

R6 Data Generating Parent Class

Details

Can be used to make your own data simulation class. Should have the same slots listed in this class at a minimum, but you can add your own, of course. An easy way to do this is to make your class inherit from this one. See the example.

Value

An [R6](#) object

Methods

Public methods:

- [DataSim\\$get_x\(\)](#)
- [DataSim\\$get_y\(\)](#)
- [DataSim\\$get_z\(\)](#)
- [DataSim\\$get_n\(\)](#)
- [DataSim\\$get_x1\(\)](#)
- [DataSim\\$get_x0\(\)](#)
- [DataSim\\$get_p\(\)](#)
- [DataSim\\$get_tau\(\)](#)
- [DataSim\\$gen_data\(\)](#)
- [DataSim\\$opt_weight\(\)](#)
- [DataSim\\$opt_weight_dist\(\)](#)
- [DataSim\\$clone\(\)](#)

Method [get_x\(\)](#): Gets the covariate data

Usage:

DataSim\$get_x()

Method get_y(): Gets the outcome vector

Usage:

DataSim\$get_y()

Method get_z(): Gets the treatment indicator

Usage:

DataSim\$get_z()

Method get_n(): Gets the number of observations

Usage:

DataSim\$get_n()

Method get_x1(): Gets the covariate data for the treated individuals

Usage:

DataSim\$get_x1()

Method get_x0(): Gets the covariate data for the control individuals

Usage:

DataSim\$get_x0()

Method get_p(): Gets the dimensionality covariate data

Usage:

DataSim\$get_p()

Method get_tau(): Gets the individual treatment effects

Usage:

DataSim\$get_tau()

Method gen_data(): Generates the data. Default is an empty function

Usage:

DataSim\$gen_data()

Method opt_weight(): Gets the optimal weights to get the correct expectation

Usage:

DataSim\$opt_weight(estimand = "ATE", augment = FALSE, solver = "mosek")

Arguments:

estimand One of "ATT", "ATC", "ATE"

augment Should we use an augmented estimator? TRUE or FALSE.

solver One of "mosek" or "gurobi"

Method opt_weight_dist(): Gets the distance of the weights from the optimal weights

Usage:

```
DataSim$opt_weight_dist(
  weight,
  estimand = "ATE",
  augment = FALSE,
  solver = "mosek"
)
```

Arguments:

weight The estimated weights
estimand One of "ATT", "ATC", "ATE"
augment Should we use an augmented estimator? TRUE or FALSE.
solver One of "mosek", "gurobi", or "quadprog"

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
DataSim$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
MyClass <- R6::R6Class("MyClass",
  inherit = DataSim,
  public = list(),
  private = list())
```

 dist.metrics

Supported distance metrics

Description

Supported distance metrics

Usage

```
dist.metrics()
```

Details

The "sdLp" method uses a metric with distances normalized by the standard deviation of each of the covariates $((x[,j]-y[,j])/sd(c(x[,j],y[,j])))^p$, where x and y are the data matrices in each group and j is a column in each matrix.

The "mahalanobis" metric is related except it normalizes by the full variance-covariance matrix. Be warned that neither "sdLp" or "mahalanobis" may make sense for binary covariates and care should be taken.

The "Lp" method uses the simple L_p norm, while "RKHS" calculates the kernel for a reproducible kernel Hilbert space (RKHS).

Value

a character vector with values "sdLp", "mahalanobis", "Lp", and "RKHS"

Examples

```
dist.metrics()
```

ESS

Effective Sample Size

Description

Effective Sample Size

Usage

```
ESS(x)

## S4 method for signature 'numeric'
ESS(x)

## S4 method for signature 'causalWeights'
ESS(x)
```

Arguments

x Either a vector of weights summing to 1 or an object of class [causalWeights](#)

Details

Calculates the effective sample size as described by Kish (1965). However, this calculation has some problems and the [PSIS\(\)](#) function should be used instead.

Value

Either a number denoting the effective sample size or if x is of class [causalWeights](#), then returns a list of both values in the treatment and control groups.

Methods (by class)

- `ESS(numeric)`: default ESS method for numeric vectors
- `ESS(causalWeights)`: ESS method for objects of class [causalWeights](#)

See Also

[PSIS\(\)](#)

Examples

```
x <- rep(1/100,100)
ESS(x)
```

estimate_effect	<i>Estimate treatment effects</i>
-----------------	-----------------------------------

Description

Estimate treatment effects

Usage

```
estimate_effect(
  data,
  formula = NULL,
  weights,
  hajek = TRUE,
  doubly.robust = TRUE,
  matched = FALSE,
  estimand = c("ATT", "ATC", "ATE", "feasible"),
  model = NULL,
  split.model = TRUE,
  sample_weight = NULL,
  ...
)
```

Arguments

data	A data.frame, a list, or a DataSim object
formula	the outcome model formula
weights	An object of class causalWeights
hajek	Should the weights be normalized to sum to 1 (TRUE/FALSE)
doubly.robust	Should an augmented estimator be used? (TRUE/FALSE)
matched	Should a matched or barycentric project estimator be used? (TRUE/FALSE)
estimand	Estimand to use. Should agree with estimand in the weights or can be left blank. One of "ATT", "ATC", or "ATE".
model	The outcome model as a character referring to a function or function
split.model	Should the outcome model be calculated separately in each treatment group? (TRUE/FALSE)
sample_weight	The sample weights. Either NULL or an object of class sampleWeights
...	Pass additional arguments to the outcome modeling functions like <code>lm</code> . Arguments "balance.covariates" and "treatment.indicator" must be provided if data is of class data.frame or matrix.

Value

an object of class `causalEffect`

Examples

```
# set-up data
data <- Hainmueller$new()
data$gen_data()

# calculate quantities
weight <- calc_weight(data, method = "Logistic")
tx_eff <- estimate_effect(data = data, weights = weight)

# get estimate
print(tx_eff$estimate)
```

Hainmueller

Hainmueller data example

Description

Hainmueller data example

Hainmueller data example

Details

Generates the data as described in Hainmueller (2012).

Value

An `R6` object of class `DataSim`

Super class

`causalOT::DataSim` -> Hainmueller

Methods**Public methods:**

- `Hainmueller$gen_data()`
- `Hainmueller$gen_x()`
- `Hainmueller$gen_y()`
- `Hainmueller$gen_z()`
- `Hainmueller$new()`
- `Hainmueller$get_design()`
- `Hainmueller$get_pscore()`

- [Hainmueller\\$clone\(\)](#)

Method `gen_data()`: Generates the data

Usage:

```
Hainmueller$gen_data()
```

Method `gen_x()`: Generates the covariate data

Usage:

```
Hainmueller$gen_x()
```

Method `gen_y()`: Generates the outcome data

Usage:

```
Hainmueller$gen_y()
```

Method `gen_z()`: Generates the treatment indicator

Usage:

```
Hainmueller$gen_z()
```

Method `new()`: Generates the the Hainmueller R6 class

Usage:

```
Hainmueller$new(
  n = 100,
  p = 6,
  param = list(),
  design = "A",
  overlap = "low",
  ...
)
```

Arguments:

`n` The number of observations

`p` The dimensions of the covariates. Fixed to 6.

`param` The data generating parameters fed as a list.

`design` One of "A" or "B". See details.

`overlap` One of "high", "low", or "medium". See details.

`...` Extra arguments. Currently unused.

Details:

Design:

Design "A" is the setting where the outcome is generated from a linear model, $Y(0) = Y(1) = X_1 + X_2 + X_3 - X_4 + X_5 + X_6 + \eta$ and design "B" is where the outcome is generated from the non-linear model $Y(0) = Y(1) = (X_1 + X_2 + X_5)^2 + \eta$.

Overlap:

The treatment indicator is generated from $Z = 1(X_1 + 2X_2 - 2X_3 - X_4 - 0.5X_5 + X_6 + \nu > 0)$, where ν depends on the overlap selected. If overlap is "high", then $\nu \sim N(0, 100)$. If overlap is "low", then $\nu \sim N(0, 30)$. Finally, if overlap is "medium", then ν is drawn from a χ^2 with 5 degrees of freedom that is scaled and centered to have mean 0.5 and variance 67.6.

Returns: An object of class [DataSim](#).

Examples:

```
data <- Hainmueller$new(n = 100, p = 6, design = "A", overlap = "low")
data$gen_data()
print(data$get_x()[1:2,])
```

Method `get_design()`: Returns the chosen design parameters

Usage:

```
Hainmueller$get_design()
```

Method `get_pscore()`: Returns the true propensity score

Usage:

```
Hainmueller$get_pscore()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Hainmueller$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## -----
## Method `Hainmueller$new`
## -----

data <- Hainmueller$new(n = 100, p = 6, design = "A", overlap = "low")
data$gen_data()
print(data$get_x()[1:2,])
```

LaLonde

LaLonde data example

Description

LaLonde data example

LaLonde data example

Details

Returns the LaLonde data as used by Dehja and Wahba. Note the data is fixed and `gen_data()` will just initialize the fixed data.

Value

An [R6](#) object of class [DataSim](#)

Super class

`causalOT::DataSim` -> LaLonde

Methods**Public methods:**

- `LaLonde$gen_data()`
- `LaLonde$get_tau()`
- `LaLonde$gen_x()`
- `LaLonde$gen_y()`
- `LaLonde$gen_z()`
- `LaLonde$new()`
- `LaLonde$get_design()`
- `LaLonde$clone()`

Method `gen_data()`: Sets up the data

Usage:

`LaLonde$gen_data()`

Method `get_tau()`: Returns the experimental treatment effect, \$1794

Usage:

`LaLonde$get_tau()`

Method `gen_x()`: Sets up the covariate data

Usage:

`LaLonde$gen_x()`

Method `gen_y()`: Sets up the outcome data

Usage:

`LaLonde$gen_y()`

Method `gen_z()`: Sets up the treatment indicator

Usage:

`LaLonde$gen_z()`

Method `new()`: Initializes the LaLonde object.

Usage:

`LaLonde$new(n = NULL, p = NULL, param = list(), design = "NSW", ...)`

Arguments:

`n` Not used. Maintained for symmetry with other DataSim objects.

`p` Not used. Maintained for symmetry with other DataSim objects.

`param` Not used. Maintained for symmetry with other DataSim objects.

`design` One of "NSW" or "Full". "NSW" uses the original experimental data from the job training program while option "Full" uses the treated individuals from LaLonde's study and compares them to individuals from the Current Population Survey as controls.

... Not used.

Examples:

```
nsw <- LaLonde$new(design = "NSW")
nsw$gen_data()
nsw$get_n()
```

```
obs.study <- LaLonde$new(design = "Full")
obs.study$gen_data()
obs.study$get_n()
```

Method `get_design()`: Returns the chosen design parameters

Usage:

```
LaLonde$get_design()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
LaLonde$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## -----
## Method `LaLonde$new`
## -----

nsw <- LaLonde$new(design = "NSW")
nsw$gen_data()
nsw$get_n()

obs.study <- LaLonde$new(design = "Full")
obs.study$gen_data()
obs.study$get_n()
```

mean_bal

Standardized absolute mean difference calculations

Description

This function will calculate the difference in means between treatment groups standardized by the pooled standard-deviation of the respective covariates.

Usage

```
mean_bal(data, weights = NULL, ...)
```

Arguments

data	Either a data.frame, matrix, or object of class DataSim
weights	An object of class causalWeights or a list with slots w0 and w1.
...	Additional arguments passed to the function to know which covariates are to be balanced ("balance.covariates"), and treatment indicator ("treatment.indicator"). These can be column names or numbers. These arguments are not needed if using the causalOT DataSim class.

Value

A vector of mean balances

Examples

```
n <- 100
p <- 6
x0 <- matrix(rnorm(2*n * p), 2*n, p)
x1 <- matrix(rnorm(n * p), n, p)
weights <- list(w0 = rep(1/(2*n), 2 * n), w1 = rep(1/n, n))
data <- cbind(rbind(x0,x1), z = c(rep(0,2*n), rep(1, n)))
colnames(data) <- c(paste0("x", 1:p), "z")
mb <- mean_bal(data, weights, balance.covariates = paste0("x", 1:p),
               treatment.indicator = "z")
print(mb)
```

ot.methods

Supported optimal transport methods

Description

Lists the supported OT methods. Note "COT" and "Wasserstein" are equivalent.

Usage

```
ot.methods()
```

Value

A character vector with values "NNM", "Wasserstein", "COT", and "SCM".

Examples

```
ot.methods()
```

pph

An external control trial of treatments for post-partum hemorrhage

Description

A dataset evaluating treatments for post-partum hemorrhage. The data contain treatment groups receiving misoprostol vs potential controls from other locations that received only oxytocin. The data is stored as a numeric matrix.

Usage

```
data(pph)
```

Format

A matrix with 802 rows and 17 variables

Details

The variables are as follows:

- `cum_blood_20m`. The outcome variable denoting cumulative blood loss in mL 20 minutes after the diagnosis of post-partum hemorrhage (650 – 2000).
- `tx`. The treatment indicator of whether an individual received misoprostol (1) or oxytocin (0).
- `age`. the mother's age in years (15 – 43).
- `no_educ`. whether a woman had no education (1) or some education (0).
- `num_livebirth`. the number of previous live births.
- `cur_married`. whether a mother is currently married (1 = yes, 0 = no).
- `gest_age`. the gestational age of the fetus in weeks (35 – 43).
- `prev_pphyes`. whether the woman has had a previous post-partum hemorrhage.
- `hb_test`. the woman's hemoglobin in mg/dL (7 – 15).
- `induced_laboryes`. whether labor was induced (1 = yes, 0 = no).
- `augmented_laboryes`. whether labor was augmented (1 = yes, 0 = no).
- `early_cordclampyes`. whether the umbilical cord was clamped early (1 = yes, 0 = no).
- `control_cordtractionyes`. whether cord traction was controlled (1 = yes, 0 = no).
- `uterine_massageyes`. whether a uterine massage was given (1 = yes, 0 = no).
- `placenta`. whether placenta was delivered before treatment given (1 = yes, 0 = no).
- `bloodlossattx`. amount of blood lost when treatment given (500 mL – 1800 mL)
- `sitecode`. Which site is the individual from? (1 = Cairo, Egypt, 2 = Turkey, 3 = Hocmon, Vietnam, 4 = Cuchi, Vietnam, and 5 Burkina Faso).

See Also

Data from the following Harvard Dataverse:

- Winikoff, Beverly, 2019, "Two randomized controlled trials of misoprostol for the treatment of postpartum hemorrhage", <https://doi.org/10.7910/DVN/ETHH4N>, Harvard Dataverse, V1.

The data was originally analyzed in

- Blum, J. et al. Treatment of post-partum haemorrhage with sublingual misoprostol versus oxytocin in women receiving prophylactic oxytocin: a double-blind, randomised, non-inferiority trial. *The Lancet* 375, 217–223 (2010).

 PSIS

Pareto-Smoothed Importance Sampling

Description

Pareto-Smoothed Importance Sampling

Usage

```

PSIS(x, r_eff = NULL, ...)

## S4 method for signature 'numeric'
PSIS(x, r_eff = NULL, ...)

## S4 method for signature 'causalWeights'
PSIS(x, r_eff = NULL, ...)

## S4 method for signature 'list'
PSIS(x, r_eff = NULL, ...)

PSIS_diag(x, ...)

## S4 method for signature 'numeric'
PSIS_diag(x, r_eff = NULL)

## S4 method for signature 'causalWeights'
PSIS_diag(x, r_eff = NULL)

## S4 method for signature 'causalPSIS'
PSIS_diag(x, ...)

## S4 method for signature 'list'
PSIS_diag(x, r_eff = NULL)

## S4 method for signature 'psis'
PSIS_diag(x, r_eff = NULL)

```

Arguments

<code>x</code>	For <code>PSIS()</code> , a vector of weights, an object of class <code>causalWeights</code> , or a list with slots "w0" and "w1". For <code>PSIS_diag</code> , the results of a run of <code>PSIS()</code> .
<code>r_eff</code>	A vector of relative effective sample size with one estimate per observation. If providing an object of class <code>causalWeights</code> , should be a list of vectors with one vector for each sample. See <code>psis()</code> from the <code>loo</code> package for more details. Updates to the <code>loo</code> package now make it so this parameter should be ignored.
<code>...</code>	Arguments passed to the <code>psis()</code> function.

Details

Acts as a wrapper to the `psis()` function from the `loo` package. It is built to handle the data types found in this package. This method is preferred to the `ESS()` function in `causalOT` since the latter is prone to error (infinite variances) but will not give good any indication that the estimates are problematic.

Value

For `PSIS()`, returns a list. See `psis()` from `loo` for a description of the outputs. Will give the log of the smoothed weights in slot `log_weights`, and in the slot `diagnostics`, it will give the `pareto_k` parameter (see the [pareto-k-diagnostic](#) page) and the `n_eff` estimates. `PSIS_diag()` returns the diagnostic slot from an object of class "psis".

Methods (by class)

- `PSIS(numeric)`: numeric weights
- `PSIS(causalWeights)`: object of class `causalWeights`
- `PSIS(list)`: list of weights
- `PSIS_diag(numeric)`: numeric weights
- `PSIS_diag(causalWeights)`: object of class `causalWeights` diagnostics
- `PSIS_diag(causalPSIS)`: diagnostics from the output of a previous call to `PSIS`
- `PSIS_diag(list)`: a list of objects
- `PSIS_diag(psis)`: output of `PSIS` function

See Also

[ESS\(\)](#)

Examples

```
x <- runif(100)
w <- x/sum(x)

res <- PSIS(x = w, r_eff = 1)
PSIS_diag(res)
```

sampleWeights-class *sampleWeights class*

Description

sampleWeights class

Slots

- a The sample weights for the fist group
- b The sample weights for the second group
- total The sample weights for the overall sample

sinkhorn *Sinkhorn Loss*

Description

This function serves as an R wrapper to the Python function SamplesLoss in the GeomLoss package <http://www.kernel-operations.io/geomloss/api/pytorch-api.html?highlight=samplesloss#geomloss.SamplesLoss>

Usage

```
sinkhorn(  
  x,  
  y,  
  a,  
  b,  
  power = 2,  
  blur = 0.05,  
  reach = NULL,  
  diameter = NULL,  
  scaling = 0.5,  
  truncate = 5,  
  metric = "Lp",  
  cluster_scale = NULL,  
  debias = TRUE,  
  verbose = FALSE,  
  backend = "auto",  
  ...  
)
```


Arguments

x	covariates for the first set of samples. Should be of class matrix.
y	covariates for the second set of samples. Should be of class matrix.
a	The empirical measure of the first set of samples.
b	The empirical measure of the second set of samples.
power	power of the optimal transport distance.
blur	The finest level of detail that should be handled by the loss function to prevent overfitting on the samples/ locations.
reach	specifies the typical scale associated to the constraint strength
diameter	A rough indication of the maximum distance between points, which is used to tune the epsilon-scaling descent and provide a default heuristic for clustering multiscale schemes. If None, a conservative estimate will be computed on-the-fly.
scaling	specifies the ratio between successive values of sigma in the epsilon-scaling descent. This parameter allows you to specify the trade-off between speed (scaling < .4) and accuracy (scaling > .9).
truncate	If backend is "multiscale", specifies the effective support of a Gaussian/Laplacian kernel as a multiple of its standard deviation
metric	Set the metric. One of "Lp", "sdLp", or "mahalanobis".
cluster_scale	If backend is "multiscale", specifies the coarse scale at which cluster centroids will be computed. If NULL, a conservative estimate will be computed from diameter and the ambient space's dimension, making sure that memory overflows won't take place.
debias	specifies if we should compute the unbiased Sinkhorn divergence instead of the classic, entropy-regularized "SoftAssign" loss.
verbose	if backend is "multiscale", specifies whether information on the clustering and epsilon-scaling descent should be displayed in the standard output.
backend	one of "auto", "tensorized", "online", or "multiscale"
...	not currently used. Used to absorb extra arguments passed by other functions without throwing an error.

Value

a list with slots "loss", "f", "g". "loss" is the Sinkhorn distance, "f" is the potential corresponding to data x, and "g" is the potential corresponding to data y.

Examples

```
## Not run:
# requires Python and GeomLoss package
x <- stats::rnorm(100, 100, 10)
a <- rep(1/100, 100)

y <- stats::rnorm(50, 50, 10)
```

```

b <- rep(1/50, 50)

sink <- sinkhorn(x = x, y = y, a = a, b = b, power = 2,
                metric = "Lp", debias = TRUE)

# sinkhorn distance, de-biased
print(sink$loss)

# potentials for first 5 obs in each group
print(sink$f[1:5])
print(sink$g[1:5])

## End(Not run)

```

`supported.methods` *Supported weighting methods*

Description

Supported weighting methods

Usage

```
supported.methods()
```

Value

A character vector with values "Logistic", "Probit", "SBW", "SCM", "CBPS", "NNM", "Wasserstein" or equivalently "COT", and "None".

Examples

```
supported.methods()
```

`supported.solvers` *Supported solvers*

Description

Supported solvers

Usage

```
supported.solvers()
```

Details

The solvers "mosek" and "gurobi" are commercial solvers that require software licenses. "quadprog" uses the `osqp` R package and "lbfgs" will either use `pytorch` in Python or the `lbfgs3c` package in R, which are both free.

Value

a character vector with values "lbfgs", "mosek", "gurobi", and "osqp"

Examples

```
supported.solvers()
```

Index

- * **datasets**
 - pph, [21](#)
- calc_weight, [2](#)
- causalEffect, [7](#), [15](#)
- causalEffect-class, [5](#)
- causalOT, [6](#)
- causalOT::DataSim, [15](#), [18](#)
- causalWeights, [4](#), [6](#), [13](#), [14](#), [23](#)
- causalWeights-class, [7](#)
- confint.causalEffect, [7](#)
- cost_fun, [8](#)

- DataSim, [10](#), [14](#), [15](#), [17](#)
- dist.metrics, [9](#), [12](#)
- dist.metrics(), [4](#)

- ESS, [13](#)
- ESS(), [23](#)
- ESS, causalWeights-method (ESS), [13](#)
- ESS, numeric-method (ESS), [13](#)
- estimate_effect, [6](#), [14](#)
- estimate_effect(), [4](#), [6](#)

- Hainmueller, [15](#)

- LaLonde, [17](#)

- mean_bal, [19](#)

- ot.methods, [20](#)

- pareto-k-diagnostic, [23](#)
- pph, [21](#)
- PSIS, [22](#)
- PSIS(), [13](#)
- psis(), [23](#)
- PSIS, causalWeights-method (PSIS), [22](#)
- PSIS, list-method (PSIS), [22](#)
- PSIS, numeric-method (PSIS), [22](#)
- PSIS_diag (PSIS), [22](#)
- PSIS_diag, causalPSIS-method (PSIS), [22](#)
- PSIS_diag, causalWeights-method (PSIS), [22](#)
- PSIS_diag, list-method (PSIS), [22](#)
- PSIS_diag, numeric-method (PSIS), [22](#)
- PSIS_diag, psis-method (PSIS), [22](#)

- R6, [10](#), [15](#), [17](#)

- sampleWeights, [14](#)
- sampleWeights-class, [24](#)
- sinkhorn, [24](#)
- supported.methods, [26](#)
- supported.methods(), [3](#)
- supported.solvers, [26](#)