Package ‘WeightIt’

April 3, 2021

Type Package

Title Weighting for Covariate Balance in Observational Studies

Version 0.12.0

Description Generates weights to form equivalent groups in observational studies with point or longitudinal treatments by easing and extending the functionality of the R packages 'twang' for generalized boosted modeling (McCaffrey, Ridgeway & Morral, 2004) <doi:10.1037/1082-989X.9.4.403>, 'CBPS' for covariate balancing propensity score weighting (Imai & Ratkovic, 2014) <doi:10.1111/rssb.12027>, 'ebal' for entropy balancing (Hainmueller, 2012) <doi:10.1093/pan/mpr025>, 'optweight' for optimization-based weights (Zubizarreta, 2015) <doi:10.1080/01621459.2015.1023805>, 'ATE' for empirical balancing calibration weighting (Chan, Yam, & Zhang, 2016) <doi:10.1111/rssb.12129>, 'SuperLearner' for stacked machine learning-based propensity scores (Pirracchio, Petersen, & van der Laan, 2015) <doi:10.1093/aje/kwu253>, among others. Also allows for assessment of weights and checking of covariate balance by interfacing directly with 'cobalt'.

Depends R (>= 3.3.0)

Imports cobalt (>= 4.3.0), ggplot2 (>= 3.3.0), crayon, backports (>= 1.2.0)

Suggests twang (>= 1.5), CBPS (>= 0.18), ATE (>= 0.2.0), optweight (>= 0.2.4), SuperLearner (>= 2.0-25), mlogit (>= 1.1.0), mclogit, MNP (>= 3.1-0), brglm2 (>= 0.5.2), osqp (>= 0.6.0.3), survey, boot, MASS, gbm (>= 2.1.3), dbarts (>= 0.9-19), knitr, rmarkdown

License GPL (>= 2)

Encoding UTF-8

URL `https://ngreifer.github.io/WeightIt/`,
`https://github.com/ngreifer/WeightIt`

BugReports `https://github.com/ngreifer/WeightIt/issues`

VignetteBuilder knitr

NeedsCompilation no

Author Noah Greifer [aut, cre] (<https://orcid.org/0000-0003-3067-7154>)

Maintainer Noah Greifer <`noah.greifer@gmail.com`>

Repository CRAN

Date/Publication 2021-04-03 13:50:23 UTC

R topics documented:

`as.weightit` *Create a* `weightit` *object manually*

Description

This function allows users to get the benefits of a `weightit` object when using weights not estimated with `weightit()` or `weightitMSM()`. These benefits include diagnostics, plots, and direct compatibility with **cobalt** for assessing balance.

Usage

```
as.weightit(...)

## Default S3 method:
as.weightit(weights,
            treat,
            covs = NULL,
            estimand = NULL,
            s.weights = NULL,
            ps = NULL,
            ...)

as.weightitMSM(...)

## Default S3 method:
as.weightitMSM(weights,
               treat.list,
               covs.list = NULL,
               estimand = NULL,
               s.weights = NULL,
               ps.list = NULL,
               ...)
```

Arguments

`weights`	required; a `numeric` vector of weights, one for each unit.
`treat`	required; a vector of treatment statuses, one for each unit.
`covs`	an optional `data.frame` of covariates. For using **WeightIt** functions, this is not necessary, but for use with **cobalt** it is.
`estimand`	an optional `character` of length 1 giving the estimand. The text is not checked.
`s.weights`	an optional `numeric` vector of sampling weights, one for each unit.
`ps`	an optional `numeric` vector of propensity scores, one for each unit.
`treat.list`	a list of treatment statuses at each time point..
`covs.list`	an optional list of `data.frame`s of covariates of covariates at each time point. For using **WeightIt** functions, this is not necessary, but for use with **cobalt** it is.
`ps.list`	an optional list of `numeric` vectors of propensity scores at each time point.
`...`	additional arguments. These must be named. They will be included in the output object.

Value

An object of class `weightit` (for `as.weightit()`) or `weightitMSM` (for `as.weightitMSM()`).

Author(s)

Noah Greifer

Examples

```
treat <- rbinom(500, 1, .3)
weights <- rchisq(500, df = 2)
W <- as.weightit(weights= weights, treat = treat,
                 estimand = "ATE")
summary(W)
```

ESS *Compute effective sample size of weighted sample*

Description

Computes the effective sample size (ESS) of a weighted sample, which represents the size of an unweighted sample with approximately the same amount of precision as the weighted sample under consideration.

Usage

```
ESS(w)
```

Arguments

`w`	a vector of weights

Details

The ESS is calculated as `sum(w)^2/sum(w^2)`.

References

McCaffrey, D. F., Ridgeway, G., & Morral, A. R. (2004). Propensity Score Estimation With Boosted Regression for Evaluating Causal Effects in Observational Studies. Psychological Methods, 9(4), 403–425. doi: 10.1037/1082989X.9.4.403

Shook-Sa, B. E., & Hudgens, M. G. (2020). Power and Sample Size for Marginal Structural Models. ArXiv:2003.05979 [Stat]. `https://arxiv.org/abs/2003.05979`

See Also

`summary.weightit()`

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ps", estimand = "ATE"))
summary(W1)
ESS(W1$weights[W1$treat == 0])
ESS(W1$weights[W1$treat == 1])
```

`get_w_from_ps` *Compute weights from propensity scores*

Description

Given a vector or matrix of propensity scores, outputs a vector of weights that target the provided estimand.

Usage

```
get_w_from_ps(ps,
              treat,
              estimand = "ATE",
              focal = NULL,
              treated = NULL,
              subclass = NULL,
              stabilize = FALSE)
```

Arguments

`ps`	A vector, matrix, or data frame of propensity scores. See Details.
`treat`	A vector of treatment status for each individual. See Details.
`estimand`	The desired estimand that the weights should target. Current options include "ATE" (average treatment effect), "ATT" (average treatment effect on the treated), "ATC" (average treatment effect on the control), "ATO" (average treatment effect in the overlap), "ATM" (average treatment effect in the matched sample), and "ATOS" (average treatment effect in the optimal subset).
`focal`	When the estimand is the ATT or ATC, which group should be consider the (focal) "treated" or "control" group, respectively. If not `NULL` and `estimand` is not "ATT" or "ATC", `estimand` will automatically be set to "ATT".
`treated`	When treatment is binary, the value of `treat` that is considered the "treated" group (i.e., the group for which the propensity scores are the probability of being in). If `NULL`, `get_w_from_ps()` will attempt to figure it out on its own using some heuristics. This really only matters when `treat` has values other than 0 and 1 and when `ps` is given as a vector or an unnamed single-column matrix or data frame.
`subclass`	`numeric`; the number of subclasses to use when computing weights using marginal mean weighting through stratification (also known as fine stratification). If `NULL`, standard inverse probability weights (and their extensions) will be computed; if a number greater than 1, subclasses will be formed and weights will be computed based on subclass membership. `estimand` must be ATE, ATT, or ATC if `subclass` is non-`NULL`. See Details.
`stabilize`	`logical`; whether to compute stabilized weights or not. This simply involves multiplying each unit's weight by the proportion of units in their treatment group. For saturated outcome models and in balance checking, this won't make a difference; otherwise, this can improve performance.

Details

`get_w_from_ps` applies the formula for computing weights from propensity scores for the desired estimand. See the References section for information on these estimands and the formulas.

`ps` can be entered a variety of ways. For binary treatments, when `ps` is entered as a vector or unnamed single-column matrix or data frame, `get_w_from_ps` has to know which value of `treat` corresponds to the "treated" group. For 0/1 variables, 1 will be considered treated. For other types of variables, `get_w_from_ps()` will try to figure it out using heuristics, but it's safer to supply an argument to `treated`. When `estimand` is "ATT" or "ATC", supplying a value to `focal` is sufficient (for ATT, `focal` is the treated group, and for ATC, `focal` is the control group). When entered as a matrix or data frame, the columns must be named with the levels of the treatment, and it is assumed that each column corresponds to the probability of being in that treatment group. This is the safest way to supply `ps` unless `treat` is a 0/1 variable.

For multi-category treatments, `ps` can be entered as a vector or a matrix or data frame. When entered as a vector, it is assumed the value corresponds to the probability of being in the treatment actually received; this is only possible when the estimand is "ATE". Otherwise, `ps` must be entered as a named matrix or data frame as described above for binary treatments. When the estimand is "ATT" or "ATC", a value for `focal` must be specified.

When `subclass` is not `NULL`, marginal mean weighting through stratification (MMWS) weights are computed. The implementation differs slightly from that described in Hong (2010, 2012). First, subclasses are formed by finding the quantiles of the propensity scores in the target group (for the ATE, all units; for the ATT or ATC, just the units in the focal group). Any subclasses lacking members of a treatment group will be filled in with them from neighboring subclasses so each subclass will always have at least one member of each treatment group. A new subclass-propensity score matrix is formed, where each unit's subclass-propensity score for each treatment value is computed as the proportion of units with that treatment value in the unit's subclass. For example, if a subclass had 10 treated units and 90 control units in it, the subclass-propensity score for being treated would be .1 and the subclass-propensity score for being control would be .9 for all units in the subclass. For multi-category treatments, the propensity scores for each treatment are stratified separately as described in Hong (2012); for binary treatments, only one set of propensity scores are stratified and the subclass-propensity scores for the other treatment are computed as the complement of the propensity scores for the stratified treatment. After the subclass-propensity scores have been computed, the standard propensity score weighting formulas are used to compute the unstabilized MMWS weights. To estimate MMWS weights equivalent to those described in Hong (2010, 2012), `stabilize` must be set to `TRUE`, but, as with standard propensity score weights, this is optional. Note that MMWS weights are also known as fine stratification weights and described by Desai et al. (2017).

`get_w_from_ps()` is not compatible with continuous treatments.

Value

A vector of weights. When `subclass` is not `NULL`, the subclasses are returned as the `"subclass"` attribute. When `estimand = "ATOS"`, the chosen value of `alpha` (the smallest propensity score allowed to remain in the sample) is returned in the `"alpha"` attribute.

Author(s)

Noah Greifer

References

Binary treatments

- `estimand = "ATO"`

Li, F., Morgan, K. L., & Zaslavsky, A. M. (2018). Balancing covariates via propensity score weighting. Journal of the American Statistical Association, 113(521), 390–400. doi: 10.1080/01621459.2016.1260466

- `estimand = "ATM"`

Li, L., & Greene, T. (2013). A Weighting Analogue to Pair Matching in Propensity Score Analysis. The International Journal of Biostatistics, 9(2). doi: 10.1515/ijb20120030

- `estimand = "ATOS"`

Crump, R. K., Hotz, V. J., Imbens, G. W., & Mitnik, O. A. (2009). Dealing with limited overlap in estimation of average treatment effects. Biometrika, 96(1), 187–199. doi: 10.1093/biomet/asn055

- Other estimands

Austin, P. C. (2011). An Introduction to Propensity Score Methods for Reducing the Effects of Confounding in Observational Studies. Multivariate Behavioral Research, 46(3), 399–424. doi: 10.1080/00273171.2011.568786

- Marginal mean weighting through stratification (MMWS)

Hong, G. (2010). Marginal mean weighting through stratification: Adjustment for selection bias in multilevel data. Journal of Educational and Behavioral Statistics, 35(5), 499–531. doi: 10.3102/1076998609359785

Desai, R. J., Rothman, K. J., Bateman, B. . T., Hernandez-Diaz, S., & Huybrechts, K. F. (2017). A Propensity-score-based Fine Stratification Approach for Confounding Adjustment When Exposure Is Infrequent: Epidemiology, 28(2), 249–257. doi: 10.1097/EDE.0000000000000595

Multinomial Treatments

- `estimand = "ATO"`

Li, F., & Li, F. (2019). Propensity score weighting for causal inference with multiple treatments. The Annals of Applied Statistics, 13(4), 2389–2415. doi: 10.1214/19AOAS1282

- `estimand = "ATM"`

Yoshida, K., Hernández-Díaz, S., Solomon, D. H., Jackson, J. W., Gagne, J. J., Glynn, R. J., & Franklin, J. M. (2017). Matching weights to simultaneously compare three treatment groups: Comparison to three-way matching. Epidemiology (Cambridge, Mass.), 28(3), 387–395. doi: 10.1097/EDE.0000000000000627

- Other estimands

McCaffrey, D. F., Griffin, B. A., Almirall, D., Slaughter, M. E., Ramchand, R., & Burgette, L. F. (2013). A Tutorial on Propensity Score Estimation for Multiple Treatments Using Generalized Boosted Models. Statistics in Medicine, 32(19), 3388–3414. doi: 10.1002/sim.5753

- Marginal mean weighting through stratification

Hong, G. (2012). Marginal mean weighting through stratification: A generalized method for evaluating multivalued and multiple treatments with nonexperimental data. Psychological Methods, 17(1), 44–60. doi: 10.1037/a0024918

See Also

method_ps()

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

ps.fit <- glm(treat ~ age + educ + race + married +
                nodegree + re74 + re75, data = lalonde,
              family = binomial)
ps <- ps.fit$fitted.values

w1 <- get_w_from_ps(ps, treat = lalonde$treat,
                    estimand = "ATT")

treatAB <- factor(ifelse(lalonde$treat == 1, "A", "B"))
w2 <- get_w_from_ps(ps, treat = treatAB,
                    estimand = "ATT", focal = "A")
all.equal(w1, w2)
w3 <- get_w_from_ps(ps, treat = treatAB,
                    estimand = "ATT", treated = "A")
all.equal(w1, w3)

#Using MMWS
w4 <- get_w_from_ps(ps, treat = lalonde$treat,
                    estimand = "ATE", subclass = 20,
                    stabilize = TRUE)

#A multi-category example using GBM predicted probabilities
library(gbm)
T3 <- factor(sample(c("A", "B", "C"), nrow(lalonde), replace = TRUE))

gbm.fit <- gbm(T3 ~ age + educ + race + married +
                 nodegree + re74 + re75, data = lalonde,
               distribution = "multinomial", n.trees = 200,
               interaction.depth = 3)
ps.multi <- drop(predict(gbm.fit, type = "response",
                         n.trees = 200))
w <- get_w_from_ps(ps.multi, T3, estimand = "ATE")
```

---

make_full_rank *Make a design matrix full rank*

Description

When writing user-defined methods for use with weightit(), it may be necessary to take the potentially non-full rank covs data frame and make it full rank for use in a downstream function. This function performs that operation.

Usage

```
make_full_rank(mat,
                with.intercept = TRUE)
```

Arguments

`mat`	a numeric matrix or data frame to be transformed. Typically this contains covariates. `NA`s are not allowed.
`with.intercept`	whether an intercept (i.e., a vector of 1s) should be added to `mat` before making it full rank. If `TRUE`, the intercept will be used in determining whether a column is linearly dependent on others. Regardless, no intercept will be included in the output.

Details

`make_full_rank()` calls `qr()` to find the rank and linearly independent columns of `mat`, which are retained while others are dropped. If `with.intercept` is set to `TRUE`, an intercept column is added to the matrix before calling `qr()`. Note that dependent columns that appear later in `mat` will be dropped first.

See example at `method_user`.

Value

An object of the same type as `mat` containing only linearly independent columns.

Note

Older versions would drop all columns that only had one value. With `with.intercept = FALSE`, if only one column has only one value, it will not be removed, and it will function as though there was an intercept present; if more than only column has only one value, only the first one will remain.

Author(s)

Noah Greifer

See Also

`method_user()`, `model.matrix()`

Examples

```
set.seed(1000)
c1 <- rbinom(10, 1, .4)
c2 <- 1-c1
c3 <- rnorm(10)
c4 <- 10*c3
mat <- data.frame(c1, c2, c3, c4)
```

```
make_full_rank(mat) #leaves c2 and c4

make_full_rank(mat, with.intercept = FALSE) #leaves c1, c2, and c4
```

---

method_bart *Propensity Score Weighting Using BART*

Description

This page explains the details of estimating weights from Bayesian additive regression trees (BART)-based propensity scores by setting `method = "bart"` in the call to `weightit()` or `weightitMSM()`. This method can be used with binary, multinomial, and continuous treatments.

In general, this method relies on estimating propensity scores using BART and then converting those propensity scores into weights using a formula that depends on the desired estimand. This method relies on `bart2()` from the **dbarts** package.

Binary Treatments: For binary treatments, this method estimates the propensity scores using `bart2()`. The following estimands are allowed: ATE, ATT, ATC, ATO, ATM, and ATOS. Weights can also be computed using marginal mean weighting through stratification for the ATE, ATT, and ATC. See `get_w_from_ps()` for details.

Multinomial Treatments: For multinomial treatments, the propensity scores are estimated using several calls to `bart2()`, one for each treatment group; the treatment probabilities are not normalized to sum to 1. The following estimands are allowed: ATE, ATT, ATC, ATO, and ATM. The weights for each estimand are computed using the standard formulas or those mentioned above. Weights can also be computed using marginal mean weighting through stratification for the ATE, ATT, and ATC. See `get_w_from_ps()` for details.

Continuous Treatments: For continuous treatments, the generalized propensity score is estimated using `bart2()`. In addition, kernel density estimation can be used instead of assuming a normal density for the numerator and denominator of the generalized propensity score by setting `use.kernel = TRUE`. Other arguments to `density()` can be specified to refine the density estimation parameters. `plot = TRUE` can be specified to plot the density for the numerator and denominator, which can be helpful in diagnosing extreme weights.

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point.

Sampling Weights: Sampling weights are not supported.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s. The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

Details

BART works by fitting a sum-of-trees model for the treatment or probability of treatment. The number of trees is determined by the `n.trees` argument. Bayesian priors are used for the hyperparameters, so the result is a posterior distribution of predicted values for each unit. The mean of these for each unit is taken for use in computing the (generalized) propensity score. Although the hyperparameters governing the priors can be modified by supplying arguments to `weightit()` that are passed to the BART fitting function, the default values tend to work well and require little modification (though the defaults differ for continuous and categorical treatments; see the `bart2()` documentation for details). Unlike many other machine learning methods, no loss function is optimized and the hyperparameters do not need to be tuned (e.g., using cross-validation), though performance can benefit from tuning. BART tends to balance sparseness with flexibility by using very weak learners as the trees, which makes it suitable for capturing complex functions without specifying a particular functional form and without overfitting.

Additional Arguments

All arguments to `bart2()` can be passed through `weightit()` or `weightitMSM()`, with the following exceptions:

- `test`, `weights`, `subset`, `offset.test` are ignored
- `combine.chains` is always set to `TRUE`
- `sampleronly` is always set to `FALSE`

For continuous treatments only, the following arguments may be supplied:

`density` A function corresponding to the conditional density of the treatment. The standardized residuals of the treatment model will be fed through this function to produce the numerator and denominator of the generalized propensity score weights. If blank, `dnorm()` is used as recommended by Robins et al. (2000). This can also be supplied as a string containing the name of the function to be called. If the string contains underscores, the call will be split by the underscores and the latter splits will be supplied as arguments to the second argument and beyond. For example, if `density = "dt_2"` is specified, the density used will be that of a t-distribution with 2 degrees of freedom. Using a t-distribution can be useful when extreme outcome values are observed (Naimi et al., 2014). Ignored if `use.kernel = TRUE` (described below).

`use.kernel` If `TRUE`, uses kernel density estimation through `density()` to estimate the numerator and denominator densities for the weights. If `FALSE`, the argument to the `density` parameter is used instead.

`bw, adjust, kernel, n` If `use.kernel = TRUE`, the arguments to the `density()` function. The defaults are the same as those in `density` except that `n` is 10 times the number of units in the sample.

`plot` If `use.kernel = TRUE`, whether to plot the estimated density.

Additional Outputs

`obj` When `include.obj = TRUE`, the `bart2` fit(s) used to generate the predicted values. With multinomial treatments, this will be a list of the fits; otherwise, it will be a single fit. The predicted probabilities used to compute the propensity scores can be extracted using `fitted()`.

Note

With version 0.9-19 or below of **dbarts**, special care has to be taken to ensure reproducibility when using `method = "bart"`. Setting a seed (either with `set.seed()` or by supplying an argument to `rngSeed`) will only work when only one thread is requested. The default is to use four threads. To request that only one thread is used, which is necessary for reproducible results, set `n.threads = 1` in the call to `weightit()` and set a seed. Note that the fewer threads are used, the slower the estimation will be. One can set `n.chains` to a lower number (default 4) to speed up the estimation at the possible expense of statistical performance.

With version 0.9-20 and above, setting the seed with `set.seed()` works correctly and results will be reproducible.

References

Hill, J., Weiss, C., & Zhai, F. (2011). Challenges With Propensity Score Strategies in a High-Dimensional Setting and a Potential Alternative. Multivariate Behavioral Research, 46(3), 477–513. doi: 10.1080/00273171.2011.570161

Chipman, H. A., George, E. I., & McCulloch, R. E. (2010). BART: Bayesian additive regression trees. The Annals of Applied Statistics, 4(1), 266–298. doi: 10.1214/09AOAS285

Note that many references that deal with BART for causal inference focus on estimating potential outcomes with BART, not the propensity scores, and so are not directly relevant when using BART to estimate propensity scores for weights.

See `method_ps` for additional references on propensity score weighting more generally.

See Also

`weightit()`, `weightitMSM()`, `get_w_from_ps()`

`method_super` for stacking predictions from several machine learning methods, including BART.

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "bart", estimand = "ATT"))
summary(W1)
bal.tab(W1)

#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "bart", estimand = "ATE"))
summary(W2)
bal.tab(W2)

#Balancing covariates with respect to re75 (continuous)
#assuming t(3) conditional density for treatment
```

```
(W3 <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "bart", density = "dt_3"))
summary(W3)
bal.tab(W3)
```

---

method_cbps *Covariate Balancing Propensity Score Weighting*

Description

This page explains the details of estimating weights from covariate balancing propensity scores by setting `method = "cbps"` in the call to `weightit()` or `weightitMSM()`. This method can be used with binary, multinomial, and continuous treatments.

In general, this method relies on estimating propensity scores using generalized method of moments and then converting those propensity scores into weights using a formula that depends on the desired estimand. This method relies on `CBPS()` from the **CBPS** package.

Binary Treatments: For binary treatments, this method estimates the propensity scores and weights using `CBPS()`. The following estimands are allowed: ATE, ATT, and ATC. The weights are taken from the output of the `CBPS` fit object. When the estimand is the ATE, the return propensity score is the probability of being in the "second" treatment group, i.e., `levels(factor(treat))[2]`; when the estimand is the ATC, the returned propensity score is the probability of being in the control (i.e., non-focal) group.

Multinomial Treatments: For multinomial treatments with three or four categories and when the estimand is the ATE, this method estimates the propensity scores and weights using one call to `CBPS()`. For multinomial treatments with three or four categories or when the estimand is the ATT, this method estimates the propensity scores and weights using multiple calls to `CBPS()`. The following estimands are allowed: ATE and ATT. The weights are taken from the output of the `CBPS` fit objects.

Continuous Treatments: For continuous treatments, the generalized propensity score and weights are estimated using `CBPS()`.

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point. This is not how `CBMSM()` in the **CBPS** package estimates weights for longitudinal treatments.

Sampling Weights: Sampling weights are supported through `s.weights` in all scenarios. See Note about sampling weights.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s (this value is arbitrary and does not affect estimation). The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

Details

CBPS estimates the coefficients of a logistic regression model (for binary treatments), multinomial logistic regression model (form multinomial treatments), or linear regression model (for continuous treatments) that is used to compute (generalized) propensity scores, from which the weights are computed. It involves augmenting the standard regression score equations with the balance constraints in an over-identified generalized method of moments estimation. The idea is to nudge the estimation of the coefficients toward those that produce balance in the weighted sample. The just-identified version (with `exact = FALSE`) does away with the score equations for the coefficients so that only the balance constraints (and the score equation for the variance of the error with a continuous treatment) are used. The just-identified version will therefore produce superior balance on the means (i.e., corresponding to the balance constraints) for binary and multinomial treatments and linear terms for continuous treatments than will the over-identified version.

Note that **WeightIt** provides less functionality than does the **CBPS** package in terms of the versions of CBPS available; for extensions to CBPS, the **CBPS** package may be preferred.

Additional Arguments

All arguments to `CBPS()` can be passed through `weightit()` or `weightitMSM()`, with the following exceptions:

- `method` in `CBPS()` is replaced with the argument `over` in `weightit()`. Setting `over = FALSE` in `weightit()` is the equivalent of setting `method = "exact"` in `CBPS()`.
- `sample.weights` is ignored because sampling weights are passed using `s.weights`.
- `standardize` is ignored.

All arguments take on the defaults of those in `CBPS()`. It may be useful in many cases to set `over = FALSE`, especially with continuous treatments.

Additional Outputs

`obj` When `include.obj = TRUE`, the CB(G)PS model fit. For binary treatments, multinomial treatments with `estimand = "ATE"` and four or fewer treatment levels, and continuous treatments, the output of the call to `CBPS()`. For multinomial treatments with `estimand = "ATT"` or with more than four treatment levels, a list of `CBPS` fit objects.

Note

When sampling weights are used with `CBPS::CBPS()`, the estimated weights already incorporate the sampling weights. When `weightit()` is used with `method = "cbps"`, the estimated weights are separated from the sampling weights, as they are with all other methods.

References

Binary treatments

Imai, K., & Ratkovic, M. (2014). Covariate balancing propensity score. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 76(1), 243–263.

Multinomial Treatments

Imai, K., & Ratkovic, M. (2014). Covariate balancing propensity score. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 76(1), 243–263.

Continuous treatments

Fong, C., Hazlett, C., & Imai, K. (2018). Covariate balancing propensity score for a continuous treatment: Application to the efficacy of political advertisements. The Annals of Applied Statistics, 12(1), 156–177. doi: 10.1214/17AOAS1101

See Also

`weightit()`, `weightitMSM()`

`CBPS::CBPS()` for the fitting function

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "cbps", estimand = "ATT"))
summary(W1)
bal.tab(W1)

## Not run:
#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "cbps", estimand = "ATE"))
summary(W2)
bal.tab(W2)

## End(Not run)

#Balancing covariates with respect to re75 (continuous)
(W3 <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "cbps", over = FALSE))
summary(W3)
bal.tab(W3)
```

`method_ebal` *Entropy Balancing*

Description

This page explains the details of estimating weights using entropy balancing by setting `method = "ebal"` in the call to `weightit()` or `weightitMSM()`. This method can be used with binary, multinomial, and continuous treatments.

In general, this method relies on estimating weights by minimizing the entropy of the weights subject to exact moment balancing constraints. This method relies on code written for **WeightIt** using `optim()`.

Binary Treatments: For binary treatments, this method estimates the weights using `optim()` using formulas described by Hainmueller (2012). The following estimands are allowed: ATE, ATT, and ATC. When the ATE is requested, the optimization is run twice, once for each treatment group.

Multinomial Treatments: For multinomial treatments, this method estimates the weights using `optim()`. The following estimands are allowed: ATE and ATT. When the ATE is requested, `optim()` is run once for each treatment group. When the ATT is requested, `optim()` is run once for each non-focal (i.e., control) group.

Continuous Treatments: For continuous treatments, this method estimates the weights using `optim()` using formulas described by Tübbicke (2020).

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point. This method is not guaranteed to yield exact balance at each time point. NOTE: the use of entropy balancing with longitudinal treatments has not been validated!

Sampling Weights: Sampling weights are supported through `s.weights` in all scenarios.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s (this value is arbitrary and does not affect estimation). The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

Details

Entropy balancing involves the specification of an optimization problem, the solution to which is then used to compute the weights. The constraints of the primal optimization problem correspond to covariate balance on the means (for binary and multinomial treatments) or treatment-covariate covariances (for continuous treatments), positivity of the weights, and that the weights sum to a certain value. It turns out that the dual optimization problem is much easier to solve because it is over only as many variables as there are balance constraints rather than over the weights for each unit and it is unconstrained. Zhao and Percival (2017) found that entropy balancing for the ATT of a binary treatment actually involves the estimation of the coefficients of a logistic regression propensity score model but using a specialized loss function different from that optimized with maximum likelihood. Entropy balancing is doubly robust (for the ATT) in the sense that it is consistent either when the true propensity score model is a logistic regression of the treatment on the covariates or when the true outcome model for the control units is a linear regression of the outcome on the covariates, and it attains a semi-parametric efficiency bound when both are true. Entropy balancing will always yield exact mean balance on the included terms.

Additional Arguments

`moments` and `int` are accepted. See `weightit()` for details.

`base.weights` A vector of base weights, one for each unit. This works for continuous treatments as well. These correspond to the base weights q in Hainmueller (2012). The estimated weights minimize the Kullback entropy divergence from the base weights, defined as $w * log(w/q)$, subject to exact balance constraints. These can be used to supply previously estimated weights so that the newly estimated weights retain the some of the properties of the original weights while ensuring the balance constraints are met. Sampling weights should not be passed to `base.weights` but can be included in a `weightit()` call that includes `s.weights`.

`d.moments` With continuous treatments, the number of moments of the treatment and covariate distributions that are constrained to be the same in the weighted sample as in the original sample. For example, setting `d.moments = 3` ensures that the mean, variance, and skew of the treatment and covariates are the same in the weighted sample as in the unweighted sample. `d.moments` should be greater than or equal to `moments` and will be automatically set accordingly if not (or if not specified). Vegetabile et al. (2020) recommend setting `d.moments = 3`, even if `moments` is less than 3. This argument corresponds to the tuning parameters $r$ and $s$ in Vegetabile et al. (2020) (which here must be equal). Ignored for binary and multi-category treatments.

The arguments `maxit` and `reltol` can be supplied and are passed to the `control` argument of `optim()`. The `"BFGS"` method is used, so the defaults correspond to this.

The `stabilize` argument is ignored; in the past it would reduce the variability of the weights through an iterative process. If you want to minimize the variance of the weights subject to balance constraints, use `method = "optweight"`.

Additional Outputs

`obj` When `include.obj = TRUE`, the output of the call to `optim()`, which contains the dual variables and convergence information. For ATE fits or with multinomial treatments, a list of `optim()` outputs, one for each weighted group.

Author(s)

The code for continuous treatments was kindly contributed by Stefan Tübbicke (`<tuebbicke@empwifo.uni-potsdam.de>`).

References

Binary Treatments

Hainmueller, J. (2012). Entropy Balancing for Causal Effects: A Multivariate Reweighting Method to Produce Balanced Samples in Observational Studies. Political Analysis, 20(1), 25–46. doi: 10.1093/pan/mpr025

Zhao, Q., & Percival, D. (2017). Entropy balancing is doubly robust. Journal of Causal Inference, 5(1). doi: 10.1515/jci20160010

Continuous Treatments

Tübbicke, S. (2020). Entropy Balancing for Continuous Treatments. ArXiv:2001.06281 [Econ, Stat]. https://arxiv.org/abs/2001.06281

Vegetabile, B. G., Griffin, B. A., Coffman, D. L., Cefalu, M., & McCaffrey, D. F. (2020). Nonparametric Estimation of Population Average Dose-Response Curves using Entropy Balancing Weights for Continuous Exposures. ArXiv:2003.02938 [Stat]. https://arxiv.org/abs/2003.02938

See Also

`weightit()`, `weightitMSM()`

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ebal", estimand = "ATT"))
summary(W1)
bal.tab(W1)

#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ebal", estimand = "ATE"))
summary(W2)
bal.tab(W2)

#Balancing covariates and squares with respect to
#re75 (continuous), maintaining 3 moments of the
#covariate and treatment distributions
(W3 <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ebal", moments = 2,
                d.moments = 3))
summary(W3)
bal.tab(W3)
```

`method_ebcw` *Empirical Balancing Calibration Weighting*

Description

This page explains the details of estimating weights using empirical balancing calibration weighting (EBCW) by setting `method = "ebcw"` in the call to `weightit()`. This method can be used with binary and multinomial treatments.

In general, this method relies on estimating weights by minimizing a function of the weights subject to exact moment balancing constraints. This method relies on `ATE()` from the **ATE** package.

Binary Treatments: For binary treatments, this method estimates the weights using `ATE()` with `ATT = TRUE`. The following estimands are allowed: ATE, ATT, and ATC. The weights are taken from the output of the `ATE` fit object. When the ATE is requested, `ATE()` is run twice, once for each treatment group.

Multinomial Treatments: For multinomial treatments, this method estimates the weights using `ATE()` with `ATT = TRUE`. The following estimands are allowed: ATE and ATT. The weights are taken from the output of the `ATE` fit objects. When the ATE is requested, `ATE()` is run once for each treatment group. When the ATT is requested, `ATE()` is run once for each non-focal (i.e., control) group.

Continuous Treatments: Continuous treatments are not supported.

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point. This method is not guaranteed to yield exact balance at each time point. NOTE: the use of EBCW with longitudinal treatments has not been validated!

Sampling Weights: Sampling weights are supported through `s.weights` in all scenarios.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s (this value is arbitrary and does not affect estimation). The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

Additional Arguments

All arguments to `ATE()` can be passed through `weightit()`, with the following exceptions:

`ATT` is ignored because the estimand is passed using `estimand`.

All arguments take on the defaults of those in `ATE`.

Additional Outputs

`obj` When `include.obj = TRUE`, the empirical balancing calibration model fit. For binary treatments with `estimand = "ATT"`, the output of the call to `ATE::ATE()`. For binary treatments with `estimand = "ATE"` and multinomial treatments, a list of outputs of calls to `ATE::ATE()`.

References

Chan, K. C. G., Yam, S. C. P., & Zhang, Z. (2016). Globally efficient non-parametric inference of average treatment effects by empirical balancing calibration weighting. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 78(3), 673–700. doi: 10.1111/rssb.12129

See Also

`weightit()`

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ebcw", estimand = "ATT"))
summary(W1)
bal.tab(W1)

#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ebcw", estimand = "ATE"))
summary(W2)
bal.tab(W2)
```

---

method_energy *Energy Balancing*

Description

This page explains the details of estimating weights using energy balancing by setting `method = "energy"` in the call to `weightit()` or `weightitMSM()`. This method can be used with binary and multinomial treatments.

In general, this method relies on estimating weights by minimizing an energy statistic related to covariate balance. For binary and multinomial treatments, this is the energy distance, which is a multivariate distance between distributions, between treatment groups.

This method relies on code written for **WeightIt** using `osqp()` from the **osqp** package to perform the optimization. This method may be slow or memory-intensive for large datasets.

Binary Treatments: For binary treatments, this method estimates the weights using `osqp()` using formulas described by Huling and Mak (2020). The following estimands are allowed: ATE, ATT, and ATC.

Multinomial Treatments: For multinomial treatments, this method estimates the weights using `osqp()` using formulas described by Huling and Mak (2020). The following estimands are allowed: ATE and ATT.

Continuous Treatments: Continuous treatments are not currently supported.

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point. This method is not guaranteed to yield optimal balance at each time point. NOTE: the use of energy balancing with longitudinal treatments has not been validated!

Sampling Weights: Sampling weights are supported through `s.weights` in all scenarios. In some cases, sampling weights will cause the optimization to fail due to lack of convexity or infeasible constraints.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s (this value is arbitrary and does not affect estimation). The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

Additional Arguments

For binary and multinomial treatments, the following additional arguments can be specified:

`improved logical`; whether to use the improved energy balancing weights as described by Huling and Mak (2020) when `estimand = "ATE"`. This involves optimizing balance not only between each treatment group and the overall sample, but also between each pair of treatment groups. Huling and Mak (2020) found that the improved energy balancing weights generally outperformed standard energy balancing. Default is `TRUE`; set to `FALSE` to use the standard energy balancing weights instead (not recommended).

`dist.mat` a numeric distance matrix to be used instead of the default distance matrix computed by `weightit()`, which uses `dist()` with default arguments. Note that some distance matrices can cause the R session to abort due to a bug within **osqp**, so this argument should be used with caution. A distance matrix must be a square, symmetric, numeric matrix with zeros along the diagonal and a row and column for each unit. Can also be supplied as the output of a call to `dist()`.

`lambda` a positive numeric scalar used to penalize the square of the weights. This value divided by the total sample size is added to the diagonal of the quadratic part of the loss function. Higher values favor weights with less variability. Note this is distinct from the lambda value described in Huling and Mak (2020), which penalizes the complexity of individual treatment rules rather than the weights.

The `moments` argument functions differently for `method = "energy"` from how it does with other methods. When unspecified or set to zero, energy balancing weights are estimated as described by Huling and Mak (2020). When `moments` is set to an integer larger than 0, additional balance constraints on the requested moments of the covariates are also included, guaranteeing exact moment balance on these covariates while minimizing the energy distance of the weighted sample. For binary and multinomial treatments, this involves exact balance on the means of the entered covariates.

Additional Outputs

`obj` When `include.obj = TRUE`, the output of the call to `solve_osqp()`, which contains the dual variables and convergence information.

Note

Sometimes the optimization can fail to converge because the problem is not convex. A warning will be displayed if so. In these cases, try simply re-fitting the weights without changing anything. If the method repeatedly fails, you should try another method or change the supplied parameters (though this is uncommon).

Author(s)

Noah Greifer

References

Huling, J. D., & Mak, S. (2020). Energy Balancing of Covariate Distributions. ArXiv:2004.13962 [Stat]. https://arxiv.org/abs/2004.13962

See Also

weightit(), weightitMSM()

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Examples may not converge, but may after several runs
## Not run:
#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "energy", estimand = "ATE"))
summary(W1)
bal.tab(W1)

#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "energy", estimand = "ATT",
                focal = "black"))
summary(W2)
bal.tab(W2)

## End(Not run)
```

`method_gbm` *Propensity Score Weighting Using Generalized Boosted Models*

Description

This page explains the details of estimating weights from generalized boosted model-based propensity scores by setting `method = "gbm"` in the call to `weightit()` or `weightitMSM()`. This method can be used with binary, multinomial, and continuous treatments.

In general, this method relies on estimating propensity scores using generalized boosted modeling and then converting those propensity scores into weights using a formula that depends on the desired estimand. The algorithm involves using a balance-based or prediction-based criterion to optimize in choosing the value of tuning parameters (the number of trees and possibly others). The method relies on the **gbm** package.

This method mimics the functionality of functions in the **twang** package, but has improved performance and more flexible options. See Details section for more details.

Binary Treatments: For binary treatments, this method estimates the propensity scores using `gbm.fit()` and then selects the optimal tuning parameter values using the method specified in the `stop.method` argument. The following estimands are allowed: ATE, ATT, ATC, ATO, and ATM. The weights are computed from the estimated propensity scores using `get_w_from_ps()`, which implements the standard formulas. Weights can also be computed using marginal mean weighting through stratification for the ATE, ATT, and ATC. See `get_w_from_ps` for details.

Multinomial Treatments: For multinomial treatments, this method estimates the propensity scores using `gbm.fit()` with `distribution = "multinomial"` and then selects the optimal tuning parameter values using the method specified in the `stop.method` argument. The following estimands are allowed: ATE, ATT, ATC, ATO, and ATM. The weights are computed from the estimated propensity scores using `get_w_from_ps()`, which implements the standard formulas. Weights can also be computed using marginal mean weighting through stratification for the ATE, ATT, and ATC. See `get_w_from_ps()` for details.

Continuous Treatments: For continuous treatments, this method estimates the generalized propensity score using `gbm.fit()` and then selects the optimal tuning parameter values using the method specified in the `stop.method` argument.

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point.

Sampling Weights: Sampling weights are supported through `s.weights` in all scenarios.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created that takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The weight estimation then proceeds with this new formula and set of covariates using surrogate splitting as described below. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

`"surr"` Surrogate splitting is used to process `NA`s. No missingness indicators are created. Nodes are split using only the non-missing values of each variable. To generate predicted values for each unit, a non-missing variable that operates similarly to the variable with missingness is used as a surrogate. Missing values are ignored when calculating balance statistics to choose the optimal tree.

Details

Generalized boosted modeling (GBM, also known as gradient boosting machines) is a machine learning method that generates predicted values from a flexible regression of the treatment on the covariates, which are treated as propensity scores and used to compute weights. It does this by building a series of regression trees, each fit to the residuals of the last, minimizing a loss function that depends on the distribution chosen. The optimal number of trees is a tuning parameter that must be chosen; McCaffrey et al. (2004) were innovative in using covariate balance to select this value rather than traditional machine learning performance metrics such as cross-validation accuracy. GBM is particularly effective for fitting nonlinear treatment models characterized by curves and interactions, but performs worse for simpler treatment models. It is unclear which balance measure should be used to select the number of trees, though research has indicated that balance measures tend to perform better than cross-validation accuracy for estimating effective propensity score weights.

WeightIt offers almost identical functionality to **twang**, the first package to implement this method. Compared to the current version of **twang**, **WeightIt** offers more options for the measure of balance used to select the number of trees, improved performance, tuning of hyperparameters, more estimands, and support for continuous treatments. **WeightIt** computes weights for multinomial treatments differently from how **twang** does; rather than fitting a separate binary GBM for each pair of treatments, **WeightIt** fits a single multi-class GBM model and uses balance measures appropriate for multinomial treatments.

Additional Arguments

The following additional arguments can be specified:

`stop.method` A string describing the balance criterion used to select the best weights. See `stop.method` for allowable options for each treatment type. In addition, to optimize the cross-validation error instead of balance, `stop.method` can be set as `"cv{#}"`, where `{#}` is replaced by a number representing the number of cross-validation folds used (e.g., `"cv5"` for 5-fold cross-validation). For binary and multinomial treatments, the default is `"es.mean"`, which minimizes the average absolute standard mean difference among the covariates between treatment groups. For continuous treatments, the default is `"p.mean"`, which minimizes the average absolute Pearson correlation between the treatment and covariates.

`trim.at` A number supplied to `at` in `trim()` which trims the weights from all the trees before choosing the best tree. This can be valuable when some weights are extreme, which occurs especially with continuous treatments. The default is 0 (i.e., no trimming).

`distribution` A string with the distribution used in the loss function of the boosted model. This is supplied to the `distribution` argument in `gbm.fit()`. For binary treatments, `"bernoulli"` and `"adaboost"` are available, with `"bernoulli"` the default. For multinomial treatments, only `"multinomial"` is allowed. For continuous treatments `"gaussian"`, `"laplace"`, and `"tdist"` are available, with `"gaussian"` the default. This argument is tunable.

`n.trees` The maximum number of trees used. This is passed onto the `n.trees` argument in `gbm.fit()`. The default is 10000 for binary and multinomial treatments and 20000 for continuous treatments.

`start.tree` The tree at which to start balance checking. If you know the best balance isn't in the first 100 trees, for example, you can set `start.tree = 101` so that balance statistics are not computed on the first 100 trees. This can save some time since balance checking takes up the bulk of the run time for some balance-based stopping methods, and is especially useful when running the same model adding more and more trees. The default is 1, i.e., to start from the very first tree in assessing balance.

`interaction.depth` The depth of the trees. This is passed onto the `interaction.depth` argument in `gbm.fit()`. Higher values indicate better ability to capture nonlinear and nonadditive relationships. The default is 3 for binary and multinomial treatments and 4 for continuous treatments. This argument is tunable.

`shrinkage` The shrinkage parameter applied to the trees. This is passed onto the `shrinkage` argument in `gbm.fit()`. The default is .01 for binary and multinomial treatments and .0005 for continuous treatments. The lower this value is, the more trees one may have to include to reach the optimum. This argument is tunable.

`bag.fraction` The fraction of the units randomly selected to propose the next tree in the expansion. This is passed onto the `bag.fraction` argument in `gbm.fit()`. The default is 1, but smaller values should be tried. For values less then 1, subsequent runs with the same parameters will yield different results due to random sampling; be sure to seed the seed using `set.seed()` to ensure replicability of results.

All other arguments take on the defaults of those in `gbm.fit()`, and some are not used at all.

The `w` argument in `gbm.fit()` is ignored because sampling weights are passed using `s.weights`.

For continuous treatments only, the following arguments may be supplied:

`density` A function corresponding to the conditional density of the treatment. The standardized residuals of the treatment model will be fed through this function to produce the numerator and denominator of the generalized propensity score weights. If blank, `dnorm()` is used as recommended by Robins et al. (2000). This can also be supplied as a string containing the name of the function to be called. If the string contains underscores, the call will be split by the underscores and the latter splits will be supplied as arguments to the second argument and beyond. For example, if `density = "dt_2"` is specified, the density used will be that of a t-distribution with 2 degrees of freedom. Using a t-distribution can be useful when extreme outcome values are observed (Naimi et al., 2014). Ignored if `use.kernel = TRUE` (described below).

`use.kernel` If `TRUE`, uses kernel density estimation through the `density()` function to estimate the numerator and denominator densities for the weights. If `FALSE` (the default), the argument to the `density` parameter is used instead.

`bw`, `adjust`, `kernel`, `n` If `use.kernel = TRUE`, the arguments to `density()`. The defaults are the same as those in `density` except that `n` is 10 times the number of units in the sample.

`plot` If `use.kernel = TRUE` with continuous treatments, whether to plot the estimated density.

For tunable arguments, multiple entries may be supplied, and `weightit()` will choose the best value by optimizing the criterion specified in `stop.method`. See below for additional outputs that are included when arguments are supplied to be tuned. See Examples for an example of tuning.

Additional Outputs

`info` A list with the following entries:

- `best.tree` The number of trees at the optimum. If this is close to `n.trees`, `weightit()` should be rerun with a larger value for `n.trees`, and `start.tree` can be set to just below `best.tree`. When other parameters are tuned, this is the best tree value in the best combination of tuned parameters. See example.
- `tree.val` A data frame with two columns: the first is the number of trees and the second is the value of the criterion corresponding to that tree. Running `plot()` on this object will plot the criterion by the number of trees and is a good way to see patterns in the relationship between them and to determine if more trees are needed. When other parameters are tuned, these are the number of trees and the criterion values in the best combination of tuned parameters. See example.

If any arguments are to be tuned (i.e., they have been supplied more than one value), the following two additional components are included in `info`:

- `tune` A data frame with a column for each argument being tuned, the best value of the balance criterion for the given combination of parameters, and the number of trees at which the best value was reached.
- `best.tune` A one-row data frame containing the values of the arguments being tuned that were ultimately selected to estimate the returned weights.

`obj` When `include.obj = TRUE`, the `gbm` fit used to generate the predicted values.

References

Binary treatments

McCaffrey, D. F., Ridgeway, G., & Morral, A. R. (2004). Propensity Score Estimation With Boosted Regression for Evaluating Causal Effects in Observational Studies. Psychological Methods, 9(4), 403–425. doi: 10.1037/1082989X.9.4.403

Multinomial Treatments

McCaffrey, D. F., Griffin, B. A., Almirall, D., Slaughter, M. E., Ramchand, R., & Burgette, L. F. (2013). A Tutorial on Propensity Score Estimation for Multiple Treatments Using Generalized Boosted Models. Statistics in Medicine, 32(19), 3388–3414. doi: 10.1002/sim.5753

Continuous treatments

Zhu, Y., Coffman, D. L., & Ghosh, D. (2015). A Boosting Algorithm for Estimating Generalized Propensity Scores with Continuous Treatments. Journal of Causal Inference, 3(1). doi: 10.1515/jci20140022

See Also

`weightit()`, `weightitMSM()`

`stop.method` for allowable arguments to `stop.method`

`gbm::gbm.fit()` for the fitting function

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "gbm", estimand = "ATE",
                stop.method = "es.max"))
summary(W1)
bal.tab(W1)

## Not run:
#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "gbm", estimand = "ATT",
                focal = "hispan", stop.method = "ks.mean"))
summary(W2)
bal.tab(W2)

#Balancing covariates with respect to re75 (continuous)
(W3 <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "gbm", use.kernel = TRUE,
                stop.method = "p.rms", trim.at = .97))
summary(W3)
bal.tab(W3)

#Using a t(3) density and illustrating the search for
#more trees.
W4a <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "gbm", density = "dt_3",
                stop.method = "p.max",
                n.trees = 10000)

W4a$info$best.tree #10000; optimum hasn't been found
plot(W4a$info$tree.val) #decreasing at right edge

W4b <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "gbm", density = "dt_3",
                stop.method = "p.max",
                start.tree = 10000,
                n.trees = 20000)

W4b$info$best.tree #13417; optimum has been found
plot(W4b$info$tree.val) #increasing at right edge

bal.tab(W4b)
```

```
#Tuning hyperparameters
(W5 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "gbm", estimand = "ATT",
                stop.method = "ks.max",
                interaction.depth = 2:4,
                distribution = c("bernoulli", "adaboost")))

W5$info$tune

W5$info$best.tune #Best values of tuned parameters

bal.tab(W5, stats = "k")

## End(Not run)
```

method_npcbps — *Nonparametric Covariate Balancing Propensity Score Weighting*

Description

This page explains the details of estimating weights from nonparametric covariate balancing propensity scores by setting `method = "npcbps"` in the call to `weightit()` or `weightitMSM()`. This method can be used with binary, multinomial, and continuous treatments.

In general, this method relies on estimating weights by maximizing the empirical likelihood of the data subject to balance constraints. This method relies on `npCBPS()` from the **CBPS** package.

Binary Treatments: For binary treatments, this method estimates the weights using `npCBPS()`. The ATE is the only estimand allowed. The weights are taken from the output of the `npCBPS` fit object.

Multinomial Treatments: For multinomial treatments, this method estimates the weights using `npCBPS()`. The ATE is the only estimand allowed. The weights are taken from the output of the `npCBPS` fit object.

Continuous Treatments: For continuous treatments, this method estimates the weights using `npCBPS()`. The weights are taken from the output of the `npCBPS` fit object.

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point. This is not how `CBMSM` in the **CBPS** package estimates weights for longitudinal treatments.

Sampling Weights: Sampling weights are **not** supported with `method = "npcbps"`.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

"ind" **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is NA and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s (this value is arbitrary and does not affect estimation). The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting weightit object will be the original covariates with the NAs.

Details

Nonparametric CBPS involves the specification of a constrained optimization problem over the weights. The constraints correspond to covariate balance, and the loss function is the empirical likelihood of the data given the weights. npCBPS is similar to entropy balancing and will generally produce similar results. Because the optimization problem of npCBPS is not convex it can be slow to converge or not converge at all, so approximate balance is allowed instead using the cor.prior argument, which controls the average deviation from zero correlation between the treatment and covariates allowed.

Additional Arguments

All arguments to npCBPS() can be passed through weightit() or weightitMSM().

All arguments take on the defaults of those in npCBPS().

Additional Outputs

obj When include.obj = TRUE, the nonparametric CB(G)PS model fit. The output of the call to CBPS::npCBPS().

References

Fong, C., Hazlett, C., & Imai, K. (2018). Covariate balancing propensity score for a continuous treatment: Application to the efficacy of political advertisements. The Annals of Applied Statistics, 12(1), 156–177. doi: 10.1214/17AOAS1101

See Also

weightit(), weightitMSM(), method_cbps

CBPS::npCBPS() for the fitting function

Examples

```
# Examples take a long time to run
## Not run:
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "npcbps", estimand = "ATE"))
summary(W1)
```

```
bal.tab(W1)

#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "npcbps", estimand = "ATE"))
summary(W2)
bal.tab(W2)

## End(Not run)
```

---

method_optweight *Optimization-Based Weighting*

Description

This page explains the details of estimating optimization-based weights 9also known as stable balancing weights) by setting method = "optweight" in the call to weightit() or weightitMSM(). This method can be used with binary, multinomial, and continuous treatments.

In general, this method relies on estimating weights by solving a quadratic programming problem subject to approximate or exact balance constraints. This method relies on optweight() from the **optweight** package.

Because optweight() offers finer control and uses the same syntax as weightit(), it is recommended that optweight() be used instead of weightit with method = "optweight".

Binary Treatments: For binary treatments, this method estimates the weights using optweight(). The following estimands are allowed: ATE, ATT, and ATC. The weights are taken from the output of the optweight fit object.

Multinomial Treatments: For multinomial treatments, this method estimates the weights using optweight(). The following estimands are allowed: ATE and ATT. The weights are taken from the output of the optweight fit object.

Continuous Treatments: For binary treatments, this method estimates the weights using optweight(). The weights are taken from the output of the optweight fit object.

Longitudinal Treatments: For longitudinal treatments, optweight() estimates weights that simultaneously satisfy balance constraints at all time points, so only one model is fit to obtain the weights. Using method = "optweight" in weightitMSM() causes is.MSM.method to be set to TRUE by default. Setting it to FALSE will run one model for each time point and multiply the weights together, a method that is not recommended. NOTE: neither use of optimization-based weights with longitudinal treatments has been validated!

Sampling Weights: Sampling weights are supported through s.weights in all scenarios.

Missing Data: In the presence of missing data, the following value(s) for missing are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s (this value is arbitrary and does not affect estimation). The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

Details

Stable balancing weights are weights that solve a constrained optimization problem, where the constraints correspond to covariate balance and the loss function is the variance (or other norm) of the weights. These weights maximize the effective sample size of the weighted sample subject to user-supplied balance constraints. An advantage of this method over entropy balancing is the ability to allow approximate, rather than exact, balance through the `tols` argument, which can increase precision even for slight relaxations of the constraints.

Additional Arguments

All arguments to `optweight()` can be passed through `weightit()` or `weightitMSM()`, with the following exception:

`targets` cannot be used and is ignored.

All arguments take on the defaults of those in `optweight()`.

Additional Outputs

`info` A list with one entry:

`duals` A data frame of dual variables for each balance constraint.

`obj` When `include.obj = TRUE`, the output of the call to `optweight::optweight()`.

Note

The specification of `tols` differs between `weightit()` and `optweight()`. In `weightit()`, one tolerance value should be included per level of each factor variable, whereas in `optweight()`, all levels of a factor are given the same tolerance, and only one value needs to be supplied for a factor variable. Because of the potential for confusion and ambiguity, it is recommended to only supply one value for `tols` in `weightit()` that applies to all variables. For finer control, use `optweight()` directly.

Seriously, just use `optweight()`. The syntax is almost identical and it's compatible with **cobalt**, too.

References

Binary Treatments

Wang, Y., & Zubizarreta, J. R. (2020). Minimal dispersion approximately balancing weights: Asymptotic properties and practical considerations. Biometrika, 107(1), 93–105. doi: 10.1093/biomet/asz050

Zubizarreta, J. R. (2015). Stable Weights that Balance Covariates for Estimation With Incomplete Outcome Data. Journal of the American Statistical Association, 110(511), 910–922. doi: 10.1080/01621459.2015.1023805

Multinomial Treatments

de los Angeles Resa, M., & Zubizarreta, J. R. (2020). Direct and stable weight adjustment in non-experimental studies with multivalued treatments: Analysis of the effect of an earthquake on post-traumatic stress. Journal of the Royal Statistical Society: Series A (Statistics in Society), n/a(n/a). doi: 10.1111/rssa.12561

Continuous Treatments

Greifer, N. (2020). Estimating Balancing Weights for Continuous Treatments Using Constrained Optimization. doi: 10.17615/DYSSB342

See Also

`weightit()`, `weightitMSM()`

`optweight::optweight()` for the fitting function

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "optweight", estimand = "ATT",
                tols = 0))
summary(W1)
bal.tab(W1)

#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "optweight", estimand = "ATE",
                tols = .01))
summary(W2)
bal.tab(W2)

#Balancing covariates with respect to re75 (continuous)
(W3 <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "optweight", tols = .05))
summary(W3)
bal.tab(W3)
```

method_ps *Propensity Score Weighting Using Generalized Linear Models*

Description

This page explains the details of estimating weights from generalized linear model-based propensity scores by setting `method = "ps"` in the call to `weightit()` or `weightitMSM()`. This method can be used with binary, multinomial, and continuous treatments.

In general, this method relies on estimating propensity scores with a parametric generalized linear model and then converting those propensity scores into weights using a formula that depends on the desired estimand. For binary and multinomial treatments, a binomial or multinomial regression model is used to estimate the propensity scores as the predicted probability of being in each treatment given the covariates. For ordinal treatments, an ordinal regression model is used to estimate generalized propensity scores. For continuous treatments, a generalized linear model is used to estimate generalized propensity scores as the conditional density of treatment given the covariates.

Binary Treatments: For binary treatments, this method estimates the propensity scores using `glm()`. An additional argument is `link`, which uses the same options as `link` in `family()`. The default link is "logit", but others, including "probit", are allowed. The following estimands are allowed: ATE, ATT, ATC, ATO, ATM, and ATOS. Weights can also be computed using marginal mean weighting through stratification for the ATE, ATT, and ATC. See `get_w_from_ps()` for details.

Multinomial Treatments: For multinomial treatments, the propensity scores are estimated using multinomial regression from one of a few functions depending on the requested link: for logit (`"logit"`) and probit (`"probit"`) links, `mlogit()` from the **mlogit** package is used; for the Bayesian probit (`"bayes.probit"`) link, `mnp()` from the **MNP** package is used; and for the biased-reduced multinomial logistic regression (`"br.logit"`), `brmultinom()` from the **brglm2** package is used. If the treatment variable is an ordered factor, `polr()` from the **MASS** package is used to fit ordinal regression. Any of the methods allowed in the `method` argument of `polr()` can be supplied to `link`. The following estimands are allowed: ATE, ATT, ATC, ATO, and ATM. The weights for each estimand are computed using the standard formulas or those mentioned above. Weights can also be computed using marginal mean weighting through stratification for the ATE, ATT, and ATC. See `get_w_from_ps()` for details.

Continuous Treatments: For continuous treatments, the generalized propensity score is estimated using linear regression. The conditional density can be specified as normal or another distribution. In addition, kernel density estimation can be used instead of assuming a specific density for the numerator and denominator of the generalized propensity score by setting `use.kernel = TRUE`. Other arguments to `density()` can be specified to refine the density estimation parameters. `plot = TRUE` can be specified to plot the density for the numerator and denominator, which can be helpful in diagnosing extreme weights.

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point.

Sampling Weights: Sampling weights are supported through `s.weights` in all scenarios except for multinomial treatments with `link = "bayes.probit"`. Warning messages may appear otherwise about non-integer successes, and these can be ignored.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s (this value is arbitrary and does not affect estimation). The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

Additional Arguments

The following additional arguments can be specified:

`link` The link used in the generalized linear model for the propensity scores. For binary treatments, `link` can be any of those allowed by `binomial()`. A `br.` prefix can be added (e.g., `"br.logit"`); this changes the fitting method to the bias-corrected generalized linear models implemented in the **brglm2** package. For multinomial treatments, `link` can be "logit", "probit", "bayes.probit", or "br.logit". For ordered treatments, `link` can be any of those allowed by the `method` argument of `polr()`. For continuous treatments, `link` can be any of those allowed by `gaussian()`.

For continuous treatments only, the following arguments may be supplied:

`density` A function corresponding the conditional density of the treatment. The standardized residuals of the treatment model will be fed through this function to produce the numerator and denominator of the generalized propensity score weights. If blank, `dnorm()` is used as recommended by Robins et al. (2000). This can also be supplied as a string containing the name of the function to be called. If the string contains underscores, the call will be split by the underscores and the latter splits will be supplied as arguments to the second argument and beyond. For example, if `density = "dt_2"` is specified, the density used will be that of a t-distribution with 2 degrees of freedom. Using a t-distribution can be useful when extreme outcome values are observed (Naimi et al., 2014). Ignored if `use.kernel = TRUE` (described below).

`use.kernel` If `TRUE`, uses kernel density estimation through the `density()` function to estimate the numerator and denominator densities for the weights. If `FALSE`, the argument to the `density` parameter is used instead.

`bw`, `adjust`, `kernel`, `n` If `use.kernel = TRUE`, the arguments to the `density()` function. The defaults are the same as those in `density` except that `n` is 10 times the number of units in the sample.

`plot` If `use.kernel = TRUE` with continuous treatments, whether to plot the estimated density.

For binary treatments, additional arguments to `glm()` can be specified as well. The `method` argument in `glm()` is renamed to `glm.method`. This can be used to supply alternative fitting functions, such as those implemented in the **glm2** package. Other arguments to `weightit()` are passed to `...` in `glm()`.

For multi-category treatments with `link = "logit"` or `"probit"`, the default is to use multinomial logistic or probit regression using the **mlogit** package. To request that separate binary logistic or probit regressions are run instead, set `use.mlogit = FALSE`. This can be helpful when `mlogit` is slow or fails to converge. With `link = "logit"`, the option `use.mclogit = TRUE` can be specified to request that `mblogit()` from the **mclogit** package is used instead, which can be faster and is recommended.

Additional Outputs

`obj` When `include.obj = TRUE`, the (generalized) propensity score model fit. For binary treatments, the output of the call to `glm()`. For ordinal treatments, the output of the call to `polr()`. For multinomial treatments with `link = "logit"` or `"probit"` and `use.mlogit = TRUE`, the output of the call to `mlogit::mlogit()`. For multinomial treatments with `use.mlogit = FALSE`, a list of the `glm()` fits. For multinomial treatments with `link = "br.logit"`, the output of the call to `brglm2::brmultinom()`. For multinomial treatments with `link = "bayes.probit"`, the output of the call to `MNP::mnp()`. For continuous treatments, the output of the call to `glm()` for the predicted values in the denominator density.

References

Binary treatments

- `estimand = "ATO"`

Li, F., Morgan, K. L., & Zaslavsky, A. M. (2018). Balancing covariates via propensity score weighting. Journal of the American Statistical Association, 113(521), 390–400. doi: 10.1080/01621459.2016.1260466

- `estimand = "ATM"`

Li, L., & Greene, T. (2013). A Weighting Analogue to Pair Matching in Propensity Score Analysis. The International Journal of Biostatistics, 9(2). doi: 10.1515/ijb20120030

- `estimand = "ATOS"`

Crump, R. K., Hotz, V. J., Imbens, G. W., & Mitnik, O. A. (2009). Dealing with limited overlap in estimation of average treatment effects. Biometrika, 96(1), 187–199. doi: 10.1093/biomet/asn055

- Other estimands

Austin, P. C. (2011). An Introduction to Propensity Score Methods for Reducing the Effects of Confounding in Observational Studies. Multivariate Behavioral Research, 46(3), 399–424. doi: 10.1080/00273171.2011.568786

- Marginal mean weighting through stratification

Hong, G. (2010). Marginal mean weighting through stratification: Adjustment for selection bias in multilevel data. Journal of Educational and Behavioral Statistics, 35(5), 499–531. doi: 10.3102/1076998609359785

- Bias-reduced logistic regression

See references for the **brglm2** package.

Multinomial Treatments

- `estimand = "ATO"`

Li, F., & Li, F. (2019). Propensity score weighting for causal inference with multiple treatments. The Annals of Applied Statistics, 13(4), 2389–2415. doi: 10.1214/19AOAS1282

- `estimand = "ATM"`

Yoshida, K., Hernández-Díaz, S., Solomon, D. H., Jackson, J. W., Gagne, J. J., Glynn, R. J., & Franklin, J. M. (2017). Matching weights to simultaneously compare three treatment groups: Comparison to three-way matching. Epidemiology (Cambridge, Mass.), 28(3), 387–395. doi: 10.1097/EDE.0000000000000627

- Other estimands

McCaffrey, D. F., Griffin, B. A., Almirall, D., Slaughter, M. E., Ramchand, R., & Burgette, L. F. (2013). A Tutorial on Propensity Score Estimation for Multiple Treatments Using Generalized Boosted Models. Statistics in Medicine, 32(19), 3388–3414. doi: 10.1002/sim.5753

- Marginal mean weighting through stratification

Hong, G. (2012). Marginal mean weighting through stratification: A generalized method for evaluating multivalued and multiple treatments with nonexperimental data. Psychological Methods, 17(1), 44–60. doi: 10.1037/a0024918

Continuous treatments

Robins, J. M., Hernán, M. Á., & Brumback, B. (2000). Marginal Structural Models and Causal Inference in Epidemiology. Epidemiology, 11(5), 550–560.

- Using non-normal conditional densities

Naimi, A. I., Moodie, E. E. M., Auger, N., & Kaufman, J. S. (2014). Constructing Inverse Probability Weights for Continuous Exposures: A Comparison of Methods. Epidemiology, 25(2), 292–299. doi: 10.1097/EDE.0000000000000053

See Also

`weightit()`, `weightitMSM()`, `get_w_from_ps()`

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ps", estimand = "ATT",
                link = "probit"))
summary(W1)
bal.tab(W1)

#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ps", estimand = "ATE"))
summary(W2)
bal.tab(W2)
```

```
#Balancing covariates with respect to re75 (continuous)
(W3 <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ps", use.kernel = TRUE))
summary(W3)
bal.tab(W3)
```

`method_super` *Propensity Score Weighting Using SuperLearner*

Description

This page explains the details of estimating weights from SuperLearner-based propensity scores by setting `method = "super"` in the call to `weightit()` or `weightitMSM()`. This method can be used with binary, multinomial, and continuous treatments.

In general, this method relies on estimating propensity scores using the SuperLearner algorithm for stacking predictions and then converting those propensity scores into weights using a formula that depends on the desired estimand. For binary and multinomial treatments, one or more binary classification algorithms are used to estimate the propensity scores as the predicted probability of being in each treatment given the covariates. For continuous treatments, regression algorithms are used to estimate generalized propensity scores as the conditional density of treatment given the covariates. This method relies on `SuperLearner()` from the **SuperLearner** package.

Binary Treatments: For binary treatments, this method estimates the propensity scores using `SuperLearner()`. The following estimands are allowed: ATE, ATT, ATC, ATO, ATM, and ATOS. Weights can also be computed using marginal mean weighting through stratification for the ATE, ATT, and ATC. See `get_w_from_ps()` for details.

Multinomial Treatments: For multinomial treatments, the propensity scores are estimated using several calls to `SuperLearner()`, one for each treatment group; the treatment probabilities are not normalized to sum to 1. The following estimands are allowed: ATE, ATT, ATC, ATO, and ATM. The weights for each estimand are computed using the standard formulas or those mentioned above. Weights can also be computed using marginal mean weighting through stratification for the ATE, ATT, and ATC. See `get_w_from_ps()` for details.

Continuous Treatments: For continuous treatments, the generalized propensity score is estimated using `SuperLearner()`. In addition, kernel density estimation can be used instead of assuming a normal density for the numerator and denominator of the generalized propensity score by setting `use.kernel = TRUE`. Other arguments to `density()` can be specified to refine the density estimation parameters. `plot = TRUE` can be specified to plot the density for the numerator and denominator, which can be helpful in diagnosing extreme weights.

Longitudinal Treatments: For longitudinal treatments, the weights are the product of the weights estimated at each time point.

Sampling Weights: Sampling weights are supported through `s.weights` in all scenarios.

Missing Data: In the presence of missing data, the following value(s) for `missing` are allowed:

`"ind"` **(default)** First, for each variable with missingness, a new missingness indicator variable is created which takes the value 1 if the original covariate is `NA` and 0 otherwise. The missingness indicators are added to the model formula as main effects. The missing values in the covariates are then replaced with 0s. The weight estimation then proceeds with this new formula and set of covariates. The covariates output in the resulting `weightit` object will be the original covariates with the `NA`s.

Details

SuperLearner works by fitting several machine learning models to the treatment and covariates and then taking a weighted combination of the generated predicted values to use as the propensity scores, which are then used to construct weights. The machine learning models used are supplied using the `SL.library` argument; the more models are supplied, the higher the chance of correctly modeling the propensity score. The predicted values are combined using the method supplied in the `SL.method` argument (which is nonnegative least squares by default). A benefit of SuperLearner is that, asymptotically, it is guaranteed to perform as well as or better than the best-performing method included in the library. Using Balance SuperLearner by setting `SL.method = "method.balance"` works by selecting the combination of predicted values that minimizes an imbalance measure.

Additional Arguments

`discrete` if `TRUE`, uses discrete SuperLearner, which simply selects the best performing method. Default `FALSE`, which finds the optimal combination of predictions for the libraries using `SL.method`.

An argument to `SL.library` **must** be supplied. To see a list of available entries, use `listWrappers()`.

All arguments to `SuperLearner()` can be passed through `weightit()` or `weightitMSM()`, with the following exceptions:

- `obsWeights` is ignored because sampling weights are passed using `s.weights`.
- `method` in `SuperLearner()` is replaced with the argument `SL.method` in `weightit()`.

For continuous treatments only, the following arguments may be supplied:

`density` A function corresponding to the conditional density of the treatment. The standardized residuals of the treatment model will be fed through this function to produce the numerator and denominator of the generalized propensity score weights. If blank, `dnorm()` is used as recommended by Robins et al. (2000). This can also be supplied as a string containing the name of the function to be called. If the string contains underscores, the call will be split by the underscores and the latter splits will be supplied as arguments to the second argument and beyond. For example, if `density = "dt_2"` is specified, the density used will be that of a t-distribution with 2 degrees of freedom. Using a t-distribution can be useful when extreme outcome values are observed (Naimi et al., 2014). Ignored if `use.kernel = TRUE` (described below).

`use.kernel` If `TRUE`, uses kernel density estimation through the `density()` function to estimate the numerator and denominator densities for the weights. If `FALSE`, the argument to the `density` parameter is used instead.

`bw`, `adjust`, `kernel`, `n` If `use.kernel = TRUE`, the arguments to the `density()` function. The defaults are the same as those in `density` except that `n` is 10 times the number of units in the sample.

`plot` If `use.kernel = TRUE`, whether to plot the estimated density.

Balance SuperLearner: In addition to the methods allowed by `SuperLearner()`, one can specify `SL.method = "method.balance"` to use "Balance SuperLearner" as described by Pirracchio and Carone (2018), wherein covariate balance is used to choose the optimal combination of the predictions from the methods specified with `SL.library`. Coefficients are chosen (one for each prediction method) so that the weights generated from the weighted combination of the predictions optimize a balance criterion, which must be set with the `stop.method` argument, described below.

`stop.method` A string describing the balance criterion used to select the best weights. See `stop.method` for allowable options for each treatment type. For binary and multinomial treatments, the default is `"es.mean"`, which minimizes the average absolute standard mean difference among the covariates between treatment groups. For continuous treatments, the default is `"p.mean"`, which minimizes the average absolute Pearson correlation between the treatment and covariates.

Note that this implementation differs from that of Pirracchio and Carone (2018) in that here, balance is measured only on the terms included in the model formula (i.e., and not their interactions unless specifically included), and balance results from a sample weighted using the estimated predicted values as propensity scores, not a sample matched using propensity score matching on the predicted values. Binary and continuous treatments are supported, but currently multinomial treatments are not.

Additional Outputs

`info` For binary and continuous treatments, a list with two entries, `coef` and `cvRisk`. For multinomial treatments, a list of lists with these two entries, one for each treatment level.

`coef` The coefficients in the linear combination of the predictions from each method in `SL.library`. Higher values indicate that the corresponding method plays a larger role in determining the resulting predicted value, and values close to zero indicate that the method plays little role in determining the predicted value. When `discrete = TRUE`, these correspond to the coefficients that would have been estimated had `discrete` been `FALSE`.

`cvRisk` The cross-validation risk for each method in `SL.library`. Higher values indicate that the method has worse cross-validation accuracy. When `SL.method = "method.balance"`, the sample weighted balance statistic requested with `stop.method`. Higher values indicate worse balance.

`obj` When `include.obj = TRUE`, the SuperLearner fit(s) used to generate the predicted values. For binary and continuous treatments, the output of the call to `SuperLearner::SuperLearner()`. For multinomial treatments, a list of outputs to calls to `SuperLearner::SuperLearner()`.

Note

Some methods formerly available in **SuperLearner** are now in **SuperLearnerExtra**, which can be found on GitHub at `https://github.com/ecpolley/SuperLearnerExtra`.

References

Binary treatments

Pirracchio, R., Petersen, M. L., & van der Laan, M. (2015). Improving Propensity Score Estimators' Robustness to Model Misspecification Using Super Learner. American Journal of Epidemiology, 181(2), 108–119. doi: 10.1093/aje/kwu253

Continuous treatments

Kreif, N., Grieve, R., Díaz, I., & Harrison, D. (2015). Evaluation of the Effect of a Continuous Treatment: A Machine Learning Approach with an Application to Treatment for Traumatic Brain Injury. Health Economics, 24(9), 1213–1228. doi: 10.1002/hec.3189

- Balance SuperLearner (`SL.method = "method.balance"`)

Pirracchio, R., & Carone, M. (2018). The Balance Super Learner: A robust adaptation of the Super Learner to improve estimation of the average treatment effect in the treated based on propensity score matching. Statistical Methods in Medical Research, 27(8), 2504–2518. doi: 10.1177/0962280216682055

See `method_ps` for additional references.

See Also

`weightit()`, `weightitMSM()`, `get_w_from_ps()`

`stop.method` for allowable arguments to `stop.method` when using `SL.method = "method.balance"`

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "super", estimand = "ATT",
                SL.library = c("SL.glm", "SL.stepAIC",
                               "SL.glm.interaction")))
summary(W1)
bal.tab(W1)

#Balancing covariates with respect to race (multinomial)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "super", estimand = "ATE",
                SL.library = c("SL.glm", "SL.stepAIC",
                               "SL.glm.interaction")))
summary(W2)
bal.tab(W2)

#Balancing covariates with respect to re75 (continuous)
#assuming t(8) conditional density for treatment
(W3 <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "super", density = "dt_8",
                SL.library = c("SL.glm", "SL.ridge",
                               "SL.glm.interaction")))
```

```
summary(W3)
bal.tab(W3)

#Balancing covariates between treatment groups (binary)
# using balance SuperLearner to minimize the average
# KS statistic
(W4 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "super", estimand = "ATT",
                SL.library = c("SL.glm", "SL.stepAIC",
                               "SL.lda"),
                SL.method = "method.balance",
                stop.method = "ks.mean"))
summary(W4)
bal.tab(W4)
```

`method_user` *User-Defined Functions for Estimating Weights*

Description

This page explains the details of estimating weights using a user-defined function. The function must take in arguments that are passed to it by `weightit()` or `weightitMSM()` and return a vector of weights or a list containing the weights.

To supply user-defined function, the function object should be entered directly to `method`; for example, for a function `fun`, `method = fun`.

Point Treatments: The following arguments are automatically passed to the user-defined function, which should have named parameters corresponding to them:

- `treat`: a vector of treatment status for each unit. This comes directly from the left hand side of the formula passed to `weightit` and so will have it's type (e.g., numeric, factor, etc.), which may need to be converted.
- `covs`: a data frame of covariate values for each unit. This comes directly from the right hand side of the formula passed to `weightit`. The covariates are processed so that all columns are numeric; all factor variables are split into dummies and all interactions are evaluated. All levels of factor variables are given dummies, so the matrix of the covariates is not full rank. Users can use `make_full_rank`, which accepts a numeric matrix or data frame and removes columns to make it full rank, if a full rank covariate matrix is desired.
- `s.weights`: a numeric vector of sampling weights, one for each unit.
- `ps`: a numeric vector of propensity scores.
- `subset`: a logical vector the same length as `treat` that is `TRUE` for units to be included in the estimation and `FALSE` otherwise. This is used to subset the input objects when `exact` is used. `treat`, `covs`, `s.weights`, and `ps`, if supplied, will already have been subsetted by `subset`.
- `estimand`: a character vector of length 1 containing the desired estimand. The characters will have been converted to uppercase. If "ATC" was supplied to estimand, `weightit` sets `focal` to the control level (usually 0 or the lowest level of `treat`) and sets `estimand` to "ATT".

- focal: a character vector of length 1 containing the focal level of the treatment when the estimand is the ATT (or the ATC as detailed above). weightit() ensures the value of focal is a level of treat.
- stabilize: a logical vector of length 1. It is not processed by weightit() before it reaches the fitting function.
- moments: a numeric vector of length 1. It is not processed by weightit() before it reaches the fitting function except that as.integer is applied to it. This is used in other methods to determine whether polynomials of the entered covariates are to be used in the weight estimation.
- int: a logical vector of length 1. It is not processed by weightit() before it reaches the fitting function. This is used in other methods to determine whether interactions of the entered covariates are to be used in the weight estimation.

None of these parameters are required to be in the fitting function. These are simply those that are automatically available.

In addition, any additional arguments supplied to weightit() will be passed on to the fitting function. weightit() ensures the arguments correspond to the parameters of the fitting function and throws an error if an incorrectly named argument is supplied and the fitting function doesn't include ... as a parameter.

The fitting function must output either a numeric vector of weights or a list (or list-like object) with an entry named wither "w" or "weights". If a list, the list can contain other named entries, but only entries named "w", "weights", "ps", and "fit.obj" will be processed. "ps" is a vector of propensity scores and "fit.obj" should be an object used in the fitting process that a user may want to examine and that is included in the weightit output object as "obj" when include.obj = TRUE. The "ps" and "fit.obj" components are optional, but "weights" or "w" is required.

Longitudinal Treatments: Longitudinal treatments can be handled either by running the fitting function for point treatments for each time point and multiplying the resulting weights together or by running a method that accommodates multiple time points and outputs a single set of weights. For the former, weightitMSM() can be used with the user-defined function just as it is with weightit(). The latter method is not yet accommodated by weightitMSM(), but will be someday, maybe.

See Also

weightit(), weightitMSM()

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#A user-defined version of method = "ps"
my.ps <- function(treat, covs, estimand, focal = NULL) {
  covs <- make_full_rank(covs)
  d <- data.frame(treat, covs)
  f <- formula(d)
  ps <- glm(f, data = d, family = "binomial")$fitted
  w <- get_w_from_ps(ps, treat = treat, estimand = estimand,
                     focal = focal)

  return(list(w = w, ps = ps))
}

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = my.ps, estimand = "ATT"))
summary(W1)
bal.tab(W1)

#Balancing covariates for longitudinal treatments
# my.ps is used at each time point.
library("twang")
data("iptwExWide", package = "twang")
(W2 <- weightitMSM(list(tx1 ~ age + gender + use0,
                        tx2 ~ tx1 + use1 + age + gender + use0,
                        tx3 ~ tx2 + use2 + tx1 + use1 + age + gender + use0),
                   data = iptwExWide,
                   method = my.ps))
summary(W2)
bal.tab(W2)

# Kernel balancing using the KBAL package, available
# using devtools::install_github("chadhazlett/KBAL").
# Only the ATT and ATC are available. Use 'kbal.method'
# instead of 'method' in weightit() to choose between
# "ebal" and "el".

## Not run:
kbal.fun <- function(treat, covs, estimand, focal, ...) {
    args <- list(...)
    if (is_not_null(focal))
        treat <- as.numeric(treat == focal)
    else if (estimand != "ATT")
        stop("estimand must be 'ATT' or 'ATC'.", call. = FALSE)
    if ("kbal.method" %in% names(args)) {
        names(args)[names(args) == "kbal.method"] <- "method"
    }
    args[names(args) %nin% setdiff(names(formals(KBAL::kbal)),
        c("X", "D"))] <- NULL
    k.out <- do.call(KBAL::kbal, c(list(X = covs, D = treat),
        args))
    w <- k.out$w
    return(list(w = w))
}

(Wk <- weightit(treat ~ age + educ + married +
                nodegree + re74, data = lalonde,
                method = kbal.fun, estimand = "ATT",
                kbal.method = "ebal"))
summary(Wk)
```

```
bal.tab(Wk, disp.ks = TRUE)

## End(Not run)
```

ps.cont *Generalized Propensity Score Estimation using GBM*

Description

ps.cont calculates generalized propensity scores and corresponding weights using boosted linear regression as implemented in gbm. This function extends ps in **twang** to continuous treatments. The syntax and output are largely the same. The GBM parameter defaults are those found in Zhu, Coffman, & Ghosh (2015).

Note: ps.cont will phased out when **twang** adds functionality for continuous treatments. All functionality and more is already present in weightit with method_gbm[method = "gbm"].

Usage

```
ps.cont(formula, data,
        n.trees = 20000,
        interaction.depth = 4,
        shrinkage = 0.0005,
        bag.fraction = 1,
        print.level = 0,
        verbose = FALSE,
        stop.method,
        sampw = NULL,
        optimize = 1,
        use.kernel = FALSE,
        ...)
## S3 method for class 'ps.cont'
summary(object, ...)
## S3 method for class 'ps.cont'
plot(x, ...)
## S3 method for class 'ps.cont'
boxplot(x, ...)
```

Arguments

formula	A formula for the propensity score model with the treatment indicator on the left side of the formula and the potential confounding variables on the right side.
data	The dataset in the form of a data frame, which should include treatment assignment as well as the covariates specified in formula.
n.trees	The number of GBM iterations passed on to gbm. The more, the better the final solution will be, but the more time it will take.

`interaction.depth`	The `interaction.depth` passed on to `gbm`.
`shrinkage`	The `shrinkage` passed on to `gbm`.
`bag.fraction`	The `bag.fraction` passed on to `gbm`.
`print.level`	Currently ignored.
`verbose`	If `TRUE`, information will be printed to monitor the the progress of the fitting.
`stop.method`	A method or methods of measuring and summarizing balance across pretreatment variables. Current options are `p.max`, `p.mean`, `p.rms`, `s.max`, `s.mean`, and `s.rms`. `p` refers to the Pearson correlation and `s` refers to the Spearman correlation. These are summarized across the pretreatment variables by the maximum (`max`), the mean (`mean`), or the square root of the mean of the squares (`rms`).
`sampw`	Optional sampling weights.
`optimize`	A numeric value, either `0`, `1`, or `2`. If `0`, balance will be checked for every tree, and the tree with the best balance will be the one used to generate the final weights. If `1`, the default, balance will be checked for a subset of trees, and then `optimize` will be used to find the tree with the best balance within the tree interval chosen. If `2`, `optimize` will be used to find the tree that yields the best balance. `0` takes the longest but is guaranteed to find the best balance among the trees. `2` is the quickest but will often choose a tree that that suboptimal balance, though not by much. `1` is a compromise between speed and comprehensiveness and is the algorithm implemented in **twang**.
`use.kernel`	Whether to use kernel density estimation as implemented in `density` to estimate the numerator of the weights. If `TRUE`, `density` will be used. If `FALSE`, the default, a normal density will be assumed and will be estimated using `dnorm()`.
`object, x`	A `ps.cont` object.
`...`	For `ps.cont`, if `use.density = TRUE`, additional arguments to `density`, which is used to produce the density for the numerator of the weights. These include `bw`, `adjust`, `kernel`, and `n`. The default values are the defaults for `density`, except `n`, which is 10 times the number of units. For `summary.ps.cont`, additional arguments affecting the summary produced.

Details

`ps.cont` extends `ps` in **twang** to continuous treatments. It estimates weights from a series of trees and then outputs the weights that optimize a user-set criterion. The criterion employed involves the correlation between the treatment and each covariate. In a fully balanced sample, the treatment will have a correlation of 0 with covariates sufficient for removing confounding. Zhu, Coffman, & Ghosh (2015), who were the first to describe GBM for propensity score weighting with continuous treatments, recommend this procedure and provided R code to implement the methods they describe. `ps.cont` adapts their syntax to make it consistent with that of `ps` in **twang**. As in Zhu et al. (2015), when the Pearson correlation is requested, weighted biserial correlations will be computed for binary covariates.

The weights are estimated as the marginal density of the treatment divided by the conditional density of the treatment on the covariates for each unit. For the marginal density, a kernel density estimator can be implemented using the `density` function. For the conditional density, a Gaussian density is assumed. Note that with treatment with outlying values, extreme weights can be produced, so it is important to examine the weights and trim them if necessary.

It is recommended to use as many trees as possible, though this requires more computation time, especially with `use.optimize` set to `0`. There is little difference between using Pearson and Spearman correlations or between using the raw correlations and the Z-transformed correlations. Typically the only `gbm`-related options that should be changed are the interaction depth and number of trees.

Missing data is not allowed in the covariates because of the ambiguity in computing correlations with missing values.

`summary.ps.cont` compresses the information in the `desc` component of the `ps.cont` object into a short summary table describing the size of the dataset and the quality of the generalized propensity score weights, in a similar way to `summary.ps`.

`plot.ps.cont` and `boxplot.ps.cont` function almost identically to `plot.ps` and `boxplot.ps`. See the help pages there for more information. Note that for `plot.ps`, only options 1, 2, and 6 are available for the `plots` argument. When `use.optimize = 2`, option 1 is not available.

Value

Returns an object of class `ps` and `ps.cont`, a list containing

`gbm.obj`	The returned `gbm` object.
`treat`	The treatment variable.
`desc`	a list containing balance tables for each method selected in `stop.method`. Includes a component for the unweighted analysis names "unw". Each `desc` component includes a list with the following components: **ess** The effective sample size **n** The number of subjects **max.p.cor** The largest absolute Pearson correlation across the covariates **mean.p.cor** The mean absolute Pearson correlation of the covariates **rmse.p.cor** The root mean squared Pearson correlation across the covariates **max.s.cor** The largest absolute Spearman correlation across the covariates **mean.s.cor** The mean absolute Spearman correlation of the covariates **rmse.s.cor** The root mean squared Spearman correlation across the covariates **bal.tab** a table summarizing the quality of the weights for yielding low treatment-covariate correlations. This table is best extracted using `bal.table`. **n.trees** The estimated optimal number of `gbm` iterations to optimize the loss function for the associated `stop.methods`
`ps`	a data frame containing the estimated generalized propensity scores. Each column is associated with one of the methods selected in `stop.methods`.
`w`	a data frame containing the propensity score weights. Each column is associated with one of the methods selected in `stop.methods`. If sampling weights are given then these are incorporated into the weights.
`estimand`	`NULL`
`datestamp`	Records the date of the analysis.
`parameters`	Saves the `ps.cont` call.

`alerts`	`NULL`
`iters`	A sequence of iterations used in the GBM fits used by `plot.ps.cont`.
`balance`	The balance summary for each tree examined, with a column for each stop.method. If `optimize` = `0`, this will contain balance summaries for all trees. If `optimize` = `1`, this will contain balance summaries for the subset of trees corresponding to `iters`. If `optimize` = `2`, this will be NULL.
`n.trees`	Maximum number of trees considered in GBM fit.
`data`	Data as specified in the `data` argument.

The `NULL` entries exist so the output object is similar to that of `ps` in **twang**.

Author(s)

Noah Greifer

`ps.cont` is heavily adapted from the R code in Zhu, Coffman, & Ghosh (2015). In contrast with their code, `ps.cont` uses weighted Pearson and Spearman correlations rather than probability weighted bootstrapped correlations, allows for different degrees of optimization in searching for the best solution, and allows for the use of kernel density estimation for the generalized propensity score. `ps.cont` also takes inspiration from `ps` in **twang**.

References

Zhu, Y., Coffman, D. L., & Ghosh, D. (2015). A Boosting Algorithm for Estimating Generalized Propensity Scores with Continuous Treatments. Journal of Causal Inference, 3(1). doi: 10.1515/jci20140022

See Also

`weightit` and `method_gbm` for its implementation using `weightit` syntax.

`ps` and `mnps` for GBM with binary and multinomial treatments.

`gbm` for the underlying machinery and explanation of the parameters.

Examples

```
# Examples take a long time
## Not run:
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates with respect to re75
psc.out <- ps.cont(re75 ~ age + educ + married +
                nodegree + race + re74, data = lalonde,
                stop.method = c("p.mean", "p.max"),
                use.optimize = 2)
summary(psc.out)
twang::bal.table(psc.out) #twang's bal.table

## End(Not run)
```

sbps *Subgroup Balancing Propensity Score*

Description

Implements the subgroup balancing propensity score (SBPS), which is an algorithm that attempts to achieve balance in subgroups by sharing information from the overall sample and subgroups. (Dong, Zhang, Zeng, & Li, 2020; DZZL). Each subgroup can use either weights estimated using the whole sample, weights estimated using just that subgroup, or a combination of the two. The optimal combination is chosen as that which minimizes an imbalance criterion that includes subgroup as well as overall balance.

Usage

```
sbps(obj, obj2 = NULL,
     moderator = NULL,
     formula = NULL,
     data = NULL,
     smooth = FALSE,
     full.search)

## S3 method for class 'weightit.sbps'
print(x, ...)

## S3 method for class 'weightit.sbps'
summary(object, top = 5,
        ignore.s.weights = FALSE, ...)

## S3 method for class 'summary.weightit.sbps'
print(x, ...)
```

Arguments

`obj`	a `weightit` object containing weights estimated in the overall sample.
`obj2`	a `weightit` object containing weights estimated in the subgroups. Typically this has been estimated by including `by` in the call to `weightit()`. Either `obj2` or `moderator` must be specified.
`moderator`	optional; a string containing the name of the variable in `data` for which weighting is to be done within subgroups or a one-sided formula with the subgrouping variable on the right-hand side. This argument is analogous to the `by` argument in `weightit()`, and in fact it is passed on to `by`. Either `obj2` or `moderator` must be specified.
`formula`	an optional formula with the covariates for which balance is to be optimized. If not specified, the formula in `obj$call` will be used.

`data`	an optional data set in the form of a data frame that contains the variables in `formula` or `moderator`.
`smooth`	`logical`; whether the smooth version of the SBPS should be used. This is only compatible with `weightit` methods that return a propensity score.
`full.search`	`logical`; when `smooth = FALSE`, whether every combination of subgroup and overall weights should be evaluated. If `FALSE`, a stochastic search as described in DZZL will be used instead. If `TRUE`, all 2^R combinations will be checked, where R is the number of subgroups, which can take a long time with many subgroups. If unspecified, will default to `TRUE` if R <= 8 and `FALSE` otherwise.
`x`	a `weightit.sbps` or `summary.weightit.sbps` object; the output of a call to `sbps()` or `summary.weightit.sbps()`.
`object`	a `weightit.sbps` object; the output of a call to `sbps()`.
`top`	how many of the largest and smallest weights to display. Default is 5.
`ignore.s.weights`	whether or not to ignore sampling weights when computing the weight summary. If `FALSE`, the default, the estimated weights will be multiplied by the sampling weights (if any) before values are computed.
`...`	for `print`, arguments passed to `print()`. Ignored otherwise.

Details

The SBPS relies on two sets of weights: one estimated in the overall sample and one estimated within each subgroup. The algorithm decides whether each subgroup should use the weights estimated in the overall sample or those estimated in the subgroup. There are 2^R permutations of overall and subgroup weights, where R is the number of subgroups. The optimal permutation is chosen as that which minimizes a balance criterion as described in DZZL. The balance criterion used here is, for binary and multinomial treatments, the sum of the squared standardized mean differences within subgroups and overall, which are computed using `col_w_smd()` in **cobalt**, and for continuous treatments, the sum of the squared correlations between each covariate and treatment within subgroups and overall, which are computed using `col_w_corr()` in **cobalt**.

The smooth version estimates weights that determine the relative contribution of the overall and subgroup propensity scores to a weighted average propensity score for each subgroup. If P_O are the propensity scores estimated in the overall sample and P_S are the propensity scores estimated in each subgroup, the smooth SBPS finds R coefficients C so that for each subgroup, the ultimate propensity score is $C * P_S + (1 - C) * P_O$, and weights are computed from this propensity score. The coefficients are estimated using `optim()` with `method = "L-BFGS-B"`. When C is estimated to be 1 or 0 for each subgroup, the smooth SBPS coincides with the standard SBPS.

If `obj2` is not specified and `moderator` is, `sbps()` will attempt to refit the model specified in `obj` with the `moderator` in the `by` argument. This relies on the environment in which `obj` was created to be intact and can take some time if `obj` was hard to fit. It's safer to estimate `obj` and `obj2` (the latter simply by including the moderator in the `by` argument) and supply these to `sbps()`.

Value

A `weightit.sbps` object, which inherits from `weightit`. This contains all the information in `obj` with the weights, propensity scores, call, and possibly covariates updated from `sbps()`. In addition, the `prop.subgroup` component contains the values of the coefficients C for the subgroups (which are either 0 or 1 for the standard SBPS), and the `moderator` component contains a data.frame with the moderator.

This object has its own summary methods and is compatible with **cobalt** functions. The `cluster` argument should be used with **cobalt** functions to accurately reflect the performance of the weights in balancing the subgroups.

Author(s)

Noah Greifer

References

Dong, J., Zhang, J. L., Zeng, S., & Li, F. (2020). Subgroup balancing propensity score. Statistical Methods in Medical Research, 29(3), 659–676. doi: 10.1177/0962280219870836

See Also

`weightit()`, `summary.weightit()`

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups within races
(W1 <- weightit(treat ~ age + educ + married +
                nodegree + race + re74, data = lalonde,
                method = "ps", estimand = "ATT"))

(W2 <- weightit(treat ~ age + educ + married +
                nodegree + race + re74, data = lalonde,
                method = "ps", estimand = "ATT",
                by = "race"))
S <- sbps(W1, W2)
print(S)
summary(S)
bal.tab(S, cluster = "race")

#Could also have run
#  sbps(W1, moderator = "race")

S_ <- sbps(W1, W2, smooth = TRUE)
print(S_)
summary(S_)
bal.tab(S_, cluster = "race")
```

`stop.method` *Balance criteria for tuning*

Description

Some methods involve tuning of parameters and use some measure of covariate balance as the criterion to select the optimal parameter values. For example with `method = "gbm"`, a balance criterion can be used to select the optimal number of trees to use. In **WeightIt**, the argument `stop.method` controls which criterion is optimized, using the naming conventions originally used in **twang**. This page documents the available arguments to `stop.method`. There has been some research on which balance criteria perform better than others; see the References section for some articles. Of course, how each method fares in any given dataset depends on a variety of factors, and multiple methods should be tried and compared before moving forward with a set of weights.

Each treatment type has its own balance criteria available.

Binary Treatments:

`"es.mean"`, `"es.max"`, `"es.rms"` The average, maximum, or root mean squared absolute standardized mean difference (ASMD) among the covariates, respectively. The ASMD is computed using `col_w_smd()` in **cobalt**. All covariates are standardized, including binary covariates (note that in **cobalt**, raw mean differences are used for binary variables by default). The standardization factor (i.e., `s.d.denom`) depends on the estimand requested and follows the conventions used in **cobalt**.

`"ks.mean"`, `"ks.max"`, `"ks.rms"` The average, maximum, or root mean squared Kolmogorov-Smirnov (KS) statistic among the covariates, respectively. The KS statistic is computed using `col_w_ks()` in **cobalt**.

`"mahalanobis"` The sample Mahalanobis distance in the weighted sample. This is similar to `"es.rms"` except that variables that are redundant with each other will be downweighted. The sample Mahalanobis distance is computed using the (generalized) inverse of the unweighted covariance matrix computed in the focal group when the estimand is the ATT or ATC and using the (generalized) inverse of the average of the unweighted covariance matrices computed within each treatment group (analogous the the ASMD, which uses the average of the group variances in its denominator).

`"energy.dist"` The energy distance between the weighted samples, as described by Huling & Mak (2020). The "improved" energy distance is used when the estimand is not the ATT. (Note that weights directly minimizing the energy distance can be found using `method = "energy"`.)

`"r2"` The pseudo-R2 of a logistic regression of the treatment on the covariates with the weights applied. Franklin et al. (2014) consider a similar metric, the post-matching C-statistic, but the pseudo-R2 accomplishes the same goal without requiring a decision boundary. The pseudo-R2 used is the McKelvy & Zavoina pseudo-R2.

`"L1.med"` The L1 statistic of the weighted samples, which is half the average absolute difference in proportion for categories of a multidimensional histogram formed by coarsening the covariates. The coarsening used is the one that yields the median unweighted L1 statistic among 101 random coarsenings of the data. Each continuous covariate is coarsened into between 2 and 12 bins, and each categorical covariate is combined into between 2 and 12 levels (or however many levels are available). Because the coarsening is random, a seed should be set to ensure results are replicable.

Multinomial Treatments:

`"es.mean"`, `"es.max"`, `"es.rms"` The average, maximum, or root mean squared absolute standardized mean difference (ASMD) among the covariates, respectively, across all pairs of treatments. The ASMD is computed using `col_w_smd()` in **cobalt**. All covariates are standardized, including binary covariates (note that in **cobalt**, raw mean differences are used for binary variables by default). The standardization factor (i.e., `s.d.denom`) depends on the estimand requested and follows the conventions used in **cobalt**. The same standardization factor is used across treatment pairs for each covariate. When the estimand is the ATT, only differences between the focal group and each other group are computed.

`"ks.mean"`, `"ks.max"`, `"ks.rms"` The average, maximum, or root mean squared Kolmogorov-Smirnov (KS) statistic among the covariates, respectively, across all pairs of treatments. The KS statistic is computed using `col_w_ks()` in **cobalt**. When the estimand is the ATT, only differences between the focal group and each other group are computed.

`"energy.dist"` The total energy distance among the weighted samples, as described by Huling & Mak (2020). The "improved" energy distance is used when the estimand is not the ATT. (Note that weights directly minimizing the energy distance can be found using `method = "energy"`.)

`"L1.med"` The L1 statistic of the weighted samples, which is the average absolute difference in proportion for categories of a multidimensional histogram formed by coarsening the covariates, divided by the number of treatment levels. The coarsening used is the one that yields the median unweighted L1 statistic among 101 random coarsenings of the data. Each continuous covariate is coarsened into between 2 and 12 bins, and each categorical covariate is combined into between 2 and 12 levels (or however many levels are available). Because the coarsening is random, a seed should be set to ensure results are replicable.

Continuous Treatments:

`"p.mean"`, `"p.max"`, `"p.rms"` The average, maximum, or root mean squared absolute Pearson correlation between the treatment and covariates, respectively. The Pearson correlation is computed using `col_w_cov()` in **cobalt**. The correlation uses the unweighted standard deviations of the treatment and covariates in the denominator.

`"s.mean"`, `"s.max"`, `"s.rms"` The average, maximum, or root mean squared absolute Spearman correlation between the treatment and covariates, respectively. The Spearman correlation is computed using `col_w_cov()` in **cobalt**. The correlation uses the unweighted standard deviations of the rank-transformed treatment and covariates in the denominator.

`"r2"` The model R2 of a linear regression of the treatment on the covariates with the weights applied. The standard R2 is used.

`"L1.med"` The L1 statistic of the weighted samples, which is the average absolute difference in proportion for categories of a multidimensional histogram formed by coarsening the covariates nd treatment, divided by the number of levels of the coarsened treatment. The coarsening used is the one that yields the median unweighted L1 statistic among 101 random coarsenings of the data. The treatment and each continuous covariate are coarsened into between 2 and 12 bins, and each categorical covariate is combined into between 2 and 12 levels (or however many levels are available). Because the coarsening is random, a seed should be set to ensure results are replicable.

References

Ali, M. S., Groenwold, R. H. H., Pestman, W. R., Belitser, S. V., Roes, K. C. B., Hoes, A. W., de Boer, A., & Klungel, O. H. (2014). Propensity score balance measures in pharmacoepidemiology: A simulation study. Pharmacoepidemiology and Drug Safety, 23(8), 802–811. doi: 10.1002/pds.3574

Belitser, S. V., Martens, E. P., Pestman, W. R., Groenwold, R. H. H., de Boer, A., & Klungel, O. H. (2011). Measuring balance and model selection in propensity score methods. Pharmacoepidemiology and Drug Safety, 20(11), 1115–1129. doi: 10.1002/pds.2188

Franklin, J. M., Rassen, J. A., Ackermann, D., Bartels, D. B., & Schneeweiss, S. (2014). Metrics for covariate balance in cohort studies of causal effects. Statistics in Medicine, 33(10), 1685–1699. doi: 10.1002/sim.6058

Huling, J. D., & Mak, S. (2020). Energy Balancing of Covariate Distributions. ArXiv:2004.13962 [Stat]. `https://arxiv.org/abs/2004.13962`

Griffin, B. A., McCaffrey, D. F., Almirall, D., Burgette, L. F., & Setodji, C. M. (2017). Chasing Balance and Other Recommendations for Improving Nonparametric Propensity Score Models. Journal of Causal Inference, 5(2). doi: 10.1515/jci20150026

See Also

`method_gbm`, `method_super`, which use these balance criteria.

`balance summary` in **cobalt** for details of some calculations.

`summary.weightit` *Print and Summarize Output*

Description

`summary()` generates a summary of the `weightit` or `weightitMSM` object to evaluate the properties of the estimated weights. `plot()` plots the distribution of the weights.

Usage

```
## S3 method for class 'weightit'
summary(object, top = 5,
        ignore.s.weights = FALSE, ...)

## S3 method for class 'summary.weightit'
print(x, ...)

## S3 method for class 'summary.weightit'
plot(x, binwidth = NULL, bins = NULL, ...)

## S3 method for class 'weightitMSM'
summary(object, top = 5,
        ignore.s.weights = FALSE, ...)

## S3 method for class 'summary.weightitMSM'
print(x, ...)
```

Arguments

`object`	a `weightit` or `weightitMSM` object; the output of a call to `weightit()` or `weightitMSM()`.
`top`	how many of the largest and smallest weights to display. Default is 5.
`ignore.s.weights`	whether or not to ignore sampling weights when computing the weight summary. If `FALSE`, the default, the estimated weights will be multiplied by the sampling weights (if any) before values are computed.
`binwidth, bins`	arguments passed to `geom_histogram()` to control the size and/or number of bins.
`x`	a `summary.weightit` or `summary.weightitMSM` object; the output of a call to `summary.weightit()` or `summary.weightitMSM()`.
`...`	for `print()`, arguments passed to `print()`. For `plot()`, additional arguments passed to `hist()` to determine the number of bins, though `geom_histogram()` from **ggplot2** is actually used to create the plot.

Value

For point treatments (i.e., `weightit` objects), a `summary.weightit` object with the following elements:

`weight.range`	The range (minimum and maximum) weight for each treatment group.
`weight.top`	The units with the greatest weights in each treatment group; how many are included is determined by `top`.
`coef.of.var (Coef of Var)`	The coefficient of variation (standard deviation divided by mean) of the weights in each treatment group and overall.
`scaled.mad (MAD)`	The mean absolute deviation of the weights in each treatment group and overall divided by the mean of the weights in the corresponding group.
`negative entropy (Entropy)`	The negative entropy ($\sum wlog(w)$) of the weights in each treatment group and overall divided by the mean of the weights in the corresponding group.
`num.zeros`	The number of weights equal to zero.
`effective.sample.size`	The effective sample size for each treatment group before and after weighting. See `ESS()`.

For longitudinal treatments (i.e., `weightitMSM` objects), a list of the above elements for each treatment period.

`plot()` returns a `ggplot` object with a histogram displaying the distribution of the estimated weights. If the estimand is the ATT or ATC, only the weights for the non-focal group(s) will be displayed (since the weights for the focal group are all 1). A dotted line is displayed at the mean of the weights.

Author(s)

Noah Greifer

See Also

`weightit()`, `weightitMSM()`, `summary()`

Examples

```
# See example at ?weightit or ?weightitMSM
```

`trim` *Trim Large Weights*

Description

Trims (i.e., truncates) large weights by setting all weights higher than that at a given quantile to the weight at the quantile. This can be useful in controlling extreme weights, which can reduce effective sample size by enlarging the variability of the weights.

Usage

```
## S3 method for class 'weightit'
trim(w, at = 0, lower = FALSE, ...)

## S3 method for class 'numeric'
trim(w, at = 0, lower = FALSE, treat = NULL, ...)
```

Arguments

`w`	A `weightit` object or a vector of weights.
`at`	`numeric`; either the quantile of the weights above which weights are to be trimmed. A single number between .5 and 1, or the number of weights to be trimmed (e.g., `at = 3` for the top 3 weights to be set to the 4th largest weight).
`lower`	`logical`; whether also to trim at the lower quantile (e.g., for `at = .9`, trimming at both .1 and .9, or for `at = 3`, trimming the top and bottom 3 weights).
`treat`	A vector of treatment status for each unit. This should always be included when `w` is numeric, but you can get away with leaving it out if the treatment is continuous or the estimand is the ATE for binary or multinomial treatments.
`...`	Not used.

Details

`trim()` takes in a `weightit` object (the output of a call to `weightit()` or `weightitMSM()`) or a numeric vector of weights and trims them to the specified quantile. All weights above that quantile are set to the weight at that quantile. If `lower = TRUE`, all weights below 1 minus the quantile are to set the weight at 1 minus the quantile. In general, trimming weights decreases balance but also decreases the variability of the weights, improving precision at the potential expense of unbiasedness (Cole & Hernán, 2008). See Lee, Lessler, and Stuart (2011) and Thoemmes and Ong (2015) for discussions and simulation results of trimming weights at various quantiles. Note that trimming weights can also change the target population and therefore the estimand.

When using `trim()` on a numeric vector of weights, it is helpful to include the treatment vector as well. The helps determine the type of treatment and estimand, which are used to specify how trimming is performed. In particular, if the estimand is determined to be the ATT or ATC, the weights of the target (i.e., focal) group are ignored, since they should all be equal to 1. Otherwise, if the estimand is the ATE or the treatment is continuous, all weights are considered for trimming. In general, weights for any group for which all the weights are the same will not be considered in the trimming.

Value

If the input is a `weightit` object, the output will be a `weightit` object with the weights replaced by the trimmed weights, while will have an additional attribute, `"trim"`, equal to the quantile of trimming.

If the input is a numeric vector of weights, the output will be a numeric vector of the trimmed weights, again with the aforementioned attribute.

Author(s)

Noah Greifer

References

Cole, S. R., & Hernán, M. Á. (2008). Constructing Inverse Probability Weights for Marginal Structural Models. American Journal of Epidemiology, 168(6), 656–664.

Lee, B. K., Lessler, J., & Stuart, E. A. (2011). Weight Trimming and Propensity Score Weighting. PLoS ONE, 6(3), e18174.

Thoemmes, F., & Ong, A. D. (2016). A Primer on Inverse Probability of Treatment Weighting and Marginal Structural Models. Emerging Adulthood, 4(1), 40–59.

See Also

`weightit()`, `weightitMSM()`

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

(W <- weightit(treat ~ age + educ + married +
                 nodegree + re74, data = lalonde,
               method = "ps", estimand = "ATT"))
summary(W)

#Trimming the top and bottom 5 weights
trim(W, at = 5, lower = TRUE)

#Trimming at 90th percentile
(W.trim <- trim(W, at = .9))

summary(W.trim)
#Note that only the control weights were trimmed

#Trimming a numeric vector of weights
all.equal(trim(W$weights, at = .9, treat = lalonde$treat),
          W.trim$weights)

#Using made up data and as.weightit()
treat <- rbinom(500, 1, .3)
weights <- rchisq(500, df = 2)
W <- as.weightit(weights = weights, treat = treat,
                 estimand = "ATE")
summary(W)
summary(trim(W, at = .95))
```

weightit *Generate Balancing Weights*

Description

`weightit()` allows for the easy generation of balancing weights using a variety of available methods for binary, continuous, and multi-category treatments. Many of these methods exist in other packages, which `weightit()` calls; these packages must be installed to use the desired method. Also included are `print()` and `summary()` methods for examining the output.

Usage

```
weightit(formula,
         data = NULL,
         method = "ps",
         estimand = "ATE",
         stabilize = FALSE,
         focal = NULL,
         by = NULL,
         s.weights = NULL,
         ps = NULL,
         moments = NULL,
         int = FALSE,
         subclass = NULL,
         missing = NULL,
         verbose = FALSE,
         include.obj = FALSE,
         ...)

## S3 method for class 'weightit'
print(x, ...)
```

Arguments

`formula`	a formula with a treatment variable on the left hand side and the covariates to be balanced on the right hand side. See `glm()` for more details. Interactions and functions of covariates are allowed.
`data`	an optional data set in the form of a data frame that contains the variables in `formula`.
`method`	a string of length 1 containing the name of the method that will be used to estimate weights. See Details below for allowable options. The default is `"ps"` for propensity score weighting.
`estimand`	the desired estimand. For binary and multi-category treatments, can be `"ATE"`, `"ATT"`, `"ATC"`, and, for some methods, `"ATO"`, `"ATM"`, or `"ATOS"`. The default for both is `"ATE"`. This argument is ignored for continuous treatments. See the individual pages for each method for more information on which estimands are allowed with each method and what literature to read to interpret these estimands.
`stabilize`	`logical`; whether or not to stabilize the weights. For the methods that involve estimating propensity scores, this involves multiplying each unit's weight by the proportion of units in their treatment group. Default is `FALSE`.
`focal`	when multi-category treatments are used and ATT weights are requested, which group to consider the "treated" or focal group. This group will not be weighted, and the other groups will be weighted to be more like the focal group. If specified, `estimand` will automatically be set to `"ATT"`.
`by`	a string containing the name of the variable in `data` for which weighting is to be done within categories or a one-sided formula with the stratifying variable on the right-hand side. For example, if `by = "gender"` or `by = ~gender`, weights will be generated separately within each level of the variable `"gender"`. The argument used to be called `exact`, which will still work but with a message. Only one `by` variable is allowed; to stratify by multiply variables simultaneously, create a new variable that is a full cross of those variables using `interaction()`.
`s.weights`	A vector of sampling weights or the name of a variable in `data` that contains sampling weights. These can also be matching weights if weighting is to be used on matched data. See the individual pages for each method for information on whether sampling weights can be supplied.
`ps`	A vector of propensity scores or the name of a variable in `data` containing propensity scores. If not `NULL`, `method` is ignored, and the propensity scores will be used to create weights. `formula` must include the treatment variable in `data`, but the listed covariates will play no role in the weight estimation. Using

	`ps` is similar to calling `get_w_from_ps()` directly, but produces a full `weightit` object rather than just producing weights.
`moments`	`numeric`; for some methods, the greatest power of each covariate to be balanced. For example, if `moments = 3`, for each non-categorical covariate, the covariate, its square, and its cube will be balanced. This argument is ignored for other methods; to balance powers of the covariates, appropriate functions must be entered in `formula`. See the individual pages for each method for information on whether they accept `moments`.
`int`	`logical`; for some methods, whether first-order interactions of the covariates are to be balanced. This argument is ignored for other methods; to balance interactions between the variables, appropriate functions must be entered in `formula`. See the individual pages for each method for information on whether they accept `int`.
`subclass`	`numeric`; the number of subclasses to use for computing weights using marginal mean weighting with subclasses (MMWS). If `NULL`, standard inverse probability weights (and their extensions) will be computed; if a number greater than 1, subclasses will be formed and weights will be computed based on subclass membership. Attempting to set a non-`NULL` value for methods that don't compute a propensity score will result in an error; see each method's help page for information on whether MMWS weights are compatible with the method. See `get_w_from_ps()` for details and references.
`missing`	`character`; how missing data should be handled. The options and defaults depend on the `method` used. Ignored if no missing data is present. It should be noted that multiple imputation outperforms all available missingness methods available in `weightit()` and should probably be used instead. Consider the **MatchThem** package for the use of `weightit()` with multiply imputed data.
`verbose`	whether to print additional information output by the fitting function.
`include.obj`	whether to include in the output any fit objects created in the process of estimating the weights. For example, with `method = "ps"`, the `glm` objects containing the propensity score model will be included. See the individual pages for each method for information on what object will be included if `TRUE`.
`...`	other arguments for functions called by `weightit()` that control aspects of fitting that are not covered by the above arguments. See Details.
`x`	a `weightit` object; the output of a call to `weightit()`.

Details

The primary purpose of `weightit()` is as a dispatcher to functions that perform the estimation of balancing weights using the requested `method`. Below are the methods allowed and links to pages containing more information about them, including additional arguments and outputs (e.g., when `include.obj = TRUE`), how missing values are treated, which estimands are allowed, and whether sampling weights are allowed.

- `"ps"` - Propensity score weighting using generalized linear models.
- `"gbm"` - Propensity score weighting using generalized boosted modeling.
- `"cbps"` - Covariate Balancing Propensity Score weighting.

- `"npcbps"` - Non-parametric Covariate Balancing Propensity Score weighting.
- `"ebal"` - Entropy balancing.
- `"ebcw"` - Empirical balancing calibration weighting.
- `"optweight"` - Optimization-based weighting.
- `"super"` - Propensity score weighting using SuperLearner.
- `"bart"` - Propensity score weighting using Bayesian additive regression trees (BART).
- `"energy"` - Energy balancing.

`method` can also be supplied as a user-defined function; see `method_user` for instructions and examples.

When using `weightit()`, please cite both the **WeightIt** package (using `citation("WeightIt")`) and the paper(s) in the references section of the method used.

Value

A `weightit` object with the following elements:

`weights`	The estimated weights, one for each unit.
`treat`	The values of the treatment variable.
`covs`	The covariates used in the fitting. Only includes the raw covariates, which may have been altered in the fitting process.
`estimand`	The estimand requested.
`method`	The weight estimation method specified.
`ps`	The estimated or provided propensity scores. Estimated propensity scores are returned for binary treatments and only when `method` is `"ps"`, `"gbm"`, `"cbps"`, `"super"`, or `"bart"`.
`s.weights`	The provided sampling weights.
`focal`	The focal variable if the ATT was requested with a multi-category treatment.
`by`	A data.frame containing the `by` variable when specified.
`obj`	When `include.obj = TRUE`, the fit object.
`info`	Additional information about the fitting. See the individual methods pages for what is included.

Author(s)

Noah Greifer

See Also

`weightitMSM()` for estimating weights with sequential (i.e., longitudinal) treatments for use in estimating marginal structural models (MSMs).

`weightit.fit()`, which is a lower-level dispatcher function that accepts a matrix of covariates and a vector of treatment statuses rather than a formula and data frame and performs minimal argument checking and processing. It may be useful for speeding up simulation studies for which the correct arguments are known. In general `weightit()` should be used.

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
(W1 <- weightit(treat ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ps", estimand = "ATT"))
summary(W1)
bal.tab(W1)

#Balancing covariates with respect to race (multi-category)
(W2 <- weightit(race ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "ebal", estimand = "ATE"))
summary(W2)
bal.tab(W2)

#Balancing covariates with respect to re75 (continuous)
(W3 <- weightit(re75 ~ age + educ + married +
                  nodegree + re74, data = lalonde,
                method = "cbps", over = FALSE))
summary(W3)
bal.tab(W3)
```

`weightit.fit` *Generate Balancing Weights with Minimal Input Processing*

Description

`weightit.fit()` dispatches one of the weight estimation methods determined by `method`. It is an internal function called by `weightit()` and should probably not be used except in special cases. Unlike `weightit()`, `weightit.fit()` does not accept a formula and data frame interface and instead requires the covariates and treatment to be supplied as a numeric matrix and atomic vector, respectively. In this way, `weightit.fit()` is to `weightit()` what `lm.fit()` is to `lm()` - a thinner, slightly faster interface that performs minimal argument checking.

Usage

```
weightit.fit(covs,
             treat,
             method = "ps",
             s.weights = NULL,
             by.factor = NULL,
             estimand = "ATE",
             focal = NULL,
             stabilize = FALSE,
             ps = NULL,
             moments = NULL,
             int = FALSE,
             subclass = NULL,
             is.MSM.method = FALSE,
             missing = NULL,
             verbose = FALSE,
             include.obj = FALSE,
             ...)
```

Arguments

`covs`	a numeric matrix of covariates.
`treat`	a vector of treatment statuses.
`method`	a string of length 1 containing the name of the method that will be used to estimate weights. See `weightit()` for allowable options. The default is `"ps"` for propensity score weighting.
`s.weights`	a numeric vector of sampling weights. See the individual pages for each method for information on whether sampling weights can be supplied.
`by.factor`	a factor variable for which weighting is to be done within levels. Corresponds to the by argument in `weightit()`.
`estimand`	the desired estimand. For binary and multi-category treatments, can be "ATE", "ATT", "ATC", and, for some methods, "ATO", "ATM", or "ATOS". The default for both is "ATE". This argument is ignored for continuous treatments. See the individual pages for each method for more information on which estimands are allowed with each method and what literature to read to interpret these estimands.
`stabilize`	`logical`; whether or not to stabilize the weights. For the methods that involve estimating propensity scores, this involves multiplying each unit's weight by the proportion of units in their treatment group. Default is `FALSE`.
`focal`	when multi-category treatments are used and ATT weights are requested, which group to consider the "treated" or focal group. This group will not be weighted, and the other groups will be weighted to be more like the focal group. Must be non-`NULL` if `estimand = "ATT"` or `"ATC"`.
`ps`	a vector of propensity scores. If specified, `method` will be ignored and set to `"ps"`.
`moments, int, subclass`	arguments to customize the weight estimation. See `weightit()` for details.
`is.MSM.method`	see `weightitMSM()`. Typically can be ignored.
`missing`	`character`; how missing data should be handled. The options depend on the `method` used. If `NULL`, `covs` covs will be checked for `NA` values, and if present, `missing` will be set to `"ind"`. If `""`, `covs` covs will not be checked for `NA` values; this can be faster when it is known there are none.
`verbose`	whether to print additional information output by the fitting function.
`include.obj`	whether to include in the output any fit objects created in the process of estimating the weights. For example, with `method = "ps"`, the `glm` objects containing

	the propensity score model will be included. See the individual pages for each method for information on what object will be included if `TRUE`.
`...`	other arguments for functions called by `weightit.fit()` that control aspects of fitting that are not covered by the above arguments.

Details

`weightit.fit()` is called by `weightit()` after the arguments to `weightit()` have been checked and processed. `weightit.fit()` dispatches the function used to actually estimate the weights, passing on the supplied arguments directly. `weightit.fit()` is not meant to be used by anyone other than experienced users who have a specific use case in mind. The returned object doesn't contain any information about the supplied arguments or details of the estimation method; all that is processed by `weightit`.

Less argument checking or processing occurs in `weightit.fit()` than does in `weightit()`, which means supplying incorrect arguments can result in errors, crashes, and invalid weights, and error and warning messages may not be helpful in diagnosing the problem. `weightit.fit()` does check to make sure weights were actually estimated, though.

`weightit.fit()` may be most useful in speeding up simulation simulation studies that use `weightit()` because the covariates can be supplied as a numeric matrix, which is often how they are generated in simulations, without having to go through the potentially slow process of extracting the covariates and treatment from a formula and data frame. If the user is certain the arguments are valid (e.g., by ensuring the estimated weights are consistent with those estimated from `weightit()` with the same arguments), less time needs to be spent on processing the arguments. Also, the returned object is much smaller than a `weightit` object because the covariates are not returned alongside the weights.

Value

A `weightit.fit` object with the following elements:

`weights`	The estimated weights, one for each unit.
`ps`	The estimated or provided propensity scores. Estimated propensity scores are returned for binary treatments and only when `method` is `"ps"`, `"gbm"`, `"cbps"`, `"super"`, or `"bart"`.
`fit.obj`	When `include.obj = TRUE`, the fit object.
`info`	Additional information about the fitting. See the individual methods pages for what is included.

The `weightit.fit` object does not have specialized `print()`, `summary()`, or `plot()` methods. It is simply a list containing the above components.

Author(s)

Noah Greifer

See Also

`weightit()`, which you should use for estimating weights unless you know better.

Examples

```
library("cobalt")
data("lalonde", package = "cobalt")

#Balancing covariates between treatment groups (binary)
covs_mat <- as.matrix(lalonde[c("age", "educ", "married",
                               "nodegree", "re74", "re75")])
wf1 <- weightit.fit(covs_mat, treat = lalonde$treat,
                    method = "ps", estimand = "ATT")
str(wf1)
```

`weightitMSM` *Generate Balancing Weights for Longitudinal Treatments*

Description

`weightitMSM()` allows for the easy generation of balancing weights for marginal structural models for time-varying treatments using a variety of available methods for binary, continuous, and multinomial treatments. Many of these methods exist in other packages, which `weightit()` calls; these packages must be installed to use the desired method. Also included are `print()` and `summary()` methods for examining the output.

Currently only "wide" data sets, where each row corresponds to a unit's entire variable history, are supported. You can use `reshape()` or other functions to transform your data into this format; see example below.

Usage

```
weightitMSM(formula.list,
            data = NULL,
            method = "ps",
            stabilize = FALSE,
            by = NULL,
            s.weights = NULL,
            num.formula = NULL,
            moments = NULL,
            int = FALSE,
            missing = NULL,
            verbose = FALSE,
            include.obj = FALSE,
            is.MSM.method,
            weightit.force = FALSE,
            ...)

## S3 method for class 'weightitMSM'
print(x, ...)
```

Arguments

`formula.list`	a list of formulas corresponding to each time point with the time-specific treatment variable on the left hand side and pre-treatment covariates to be balanced on the right hand side. The formulas must be in temporal order, and must contain all covariates to be balanced at that time point (i.e., treatments and covariates featured in early formulas should appear in later ones). Interactions and functions of covariates are allowed.
`data`	an optional data set in the form of a data frame that contains the variables in the formulas in `formula.list`. This must be a wide data set with exactly one row per unit.
`method`	a string of length 1 containing the name of the method that will be used to estimate weights. See `weightit()` for allowable options. The default is `"ps"`, which estimates the weights using generalized linear models.
`stabilize`	`logical`; whether or not to stabilize the weights. Stabilizing the weights involves fitting a model predicting treatment at each time point from treatment status at prior time points. If `TRUE`, a saturated model will be fit, essentially using the observed treatment probabilities in the numerator (for binary and multinomial treatments). This may yield an error if some combinations are not observed. Default is `FALSE`. To manually specify stabilization model formulas, use `num.formula`.
`num.formula`	optional; a one-sided formula with the stabilization factors (other than the previous treatments) on the right hand side, which adds, for each time point, the stabilization factors to a model saturated with previous treatments. See Cole & Hernán (2008) for a discussion of how to specify this model; including stabilization factors can change the estimand without proper adjustment, and should be done with caution. Unless you know what you are doing, we recommend setting `stabilize = TRUE` and ignoring `num.formula`.
`by`	a string containing the name of the variable in `data` for which weighting is to be done within categories or a one-sided formula with the stratifying variable on the right-hand side. For example, if `by = "gender"` or `by = ~ gender`, weights will be generated separately within each level of the variable `"gender"`. The argument used to be called `exact`, which will still work but with a message. Only one `by` variable is allowed.
`s.weights`	a vector of sampling weights or the name of a variable in `data` that contains sampling weights. These are ignored for some methods.
`moments`	`numeric`; for some methods, the greatest power of each covariate to be balanced. For example, if `moments = 3`, for each non-categorical covariate, the covariate, its square, and its cube will be balanced. This argument is ignored for other methods; to balance powers of the covariates, appropriate functions must be entered in `formula`. See the specific methods help pages for information on whether they accept `moments`.
`int`	`logical`; for some methods, whether first-order interactions of the covariates are to be balanced. This argument is ignored for other methods; to balance interactions between the variables, appropriate functions must be entered in `formula`. See the specific methods help pages for information on whether they accept `int`.

`missing`	`character`; how missing data should be handled. The options and defaults depend on the `method` used. Ignored if no missing data is present. It should be noted that multiple imputation outperforms all available missingness methods available in `weightit()` and should probably be used instead. See the **MatchThem** package for the use of `weightit()` with multiply imputed data.
`verbose`	whether to print additional information output by the fitting function.
`include.obj`	whether to include in the output a list of the fit objects created in the process of estimating the weights at each time point. For example, with `method = "ps"`, a list of the `glm` objects containing the propensity score models at each time point will be included. See the help pages for each method for information on what object will be included if `TRUE`.
`is.MSM.method`	whether the method estimates weights for multiple time points all at once (`TRUE`) or by estimating weights at each time point and then multiplying them together (`FALSE`). This is only relevant for `method = "optweight")`, which estimates weights for longitudinal treatments all at once, and for user-specified functions.
`weightit.force`	several methods are not valid for estimating weights with longitudinal treatments, and will produce an error message if attempted. Set to `TRUE` to bypass this error message.
`...`	other arguments for functions called by `weightit()` that control aspects of fitting that are not covered by the above arguments. See Details at `weightit()`.
`x`	a `weightitMSM` object; the output of a call to `weightitMSM()`.

Details

In general, `weightitMSM()` works by separating the estimation of weights into separate procedures for each time period based on the formulas provided. For each formula, `weightitMSM()` simply calls `weightit()` to that formula, collects the weights for each time period, and multiplies them together to arrive at longitudinal balancing weights.

Each formula should contain all the covariates to be balanced on. For example, the formula corresponding to the second time period should contain all the baseline covariates, the treatment variable at the first time period, and the time-varying covariates that took on values after the first treatment and before the second. Currently, only wide data sets are supported, where each unit is represented by exactly one row that contains the covariate and treatment history encoded in separate variables.

The `"cbps"` method, which calls `CBPS()` in **CBPS**, will yield different results from `CBMSM()` in **CBPS** because `CBMSM()` takes a different approach to generating weights than simply estimating several time-specific models.

Value

A `weightitMSM` object with the following elements:

`weights`	The estimated weights, one for each unit.
`treat.list`	A list of the values of the time-varying treatment variables.
`covs.list`	A list of the covariates used in the fitting at each time point. Only includes the raw covariates, which may have been altered in the fitting process.
`data`	The data.frame originally entered to `weightitMSM()`.

`estimand`	"ATE", currently the only estimand for MSMs with binary or multinomial treatments.
`method`	The weight estimation method specified.
`ps.list`	A list of the estimated propensity scores (if any) at each time point.
`s.weights`	The provided sampling weights.
`by`	A data.frame containing the by variable when specified.
`stabilization`	The stabilization factors, if any.

Author(s)

Noah Greifer

References

Cole, S. R., & Hernán, M. A. (2008). Constructing Inverse Probability Weights for Marginal Structural Models. American Journal of Epidemiology, 168(6), 656–664. doi: 10.1093/aje/kwn164

See Also

`weightit()` for information on the allowable methods.

Examples

```
library("twang")
# library("cobalt")

data("iptwExWide", package = "twang")
(W <- weightitMSM(list(tx1 ~ age + gender + use0,
                      tx2 ~ tx1 + use1 + age + gender + use0,
                      tx3 ~ tx2 + use2 + tx1 + use1 + age + gender + use0),
                 data = iptwExWide,
                 method = "ps"))
summary(W)
# bal.tab(W)

##Going from long to wide data
data("iptwExLong", package = "twang")
wide_data <- reshape(iptwExLong$covariates,    #long data
                     timevar = "time",         #time variable
                     v.names = c("use", "tx"), #time-varying
                     idvar = "ID",             #time-stable
                     direction = "wide",
                     sep = "")

(W2 <- weightitMSM(list(tx1 ~ age + gender + use1,
                      tx2 ~ tx1 + use2 + age + gender + use1,
                      tx3 ~ tx2 + use3 + tx1 + use2 + age +
                          gender + use1),
                 data = wide_data,
                 method = "ps"))
```

```
summary(W2)

all.equal(W$weights, W2$weights)
```

Index