Package 'optweight'

January 16, 2019

Type Package

Title Targeted Stable Balancing Weights Using Optimization

Version 0.2.1

Author Noah Greifer [aut, cre]

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

Description Use optimization to estimate weights that balance covariates for binary, multinomial, continuous, and longitudinal treatments in the spirit of Zubizarreta (2015) <doi:10.1080/01621459.2015.1023805>. The degree of balance can be specified for each covariate. In addition, sampling weights can be estimated that allow a sample to generalize to a population specified with given target moments of covariates.

Depends R (>= 3.4.0)

Imports rosqp (>= 0.1.0), Matrix (>= 1.2-13), ggplot2 (>= 3.0.0)

Suggests cobalt (>= 3.6.1), twang (>= 1.5)

License GPL

Encoding UTF-8

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2019-01-16 06:40:03 UTC

R topics documented:

check.targets 2
check.tols 3
optweight 5
optweight.fit 11
optweight.svy 14
optweight.svy.fit 17
plot.optweight 20
summary.optweight 21

Index **24**

check.targets *Construct and Check Targets Input*

Description

Checks whether proposed target population means values for `targets` are suitable in number and order for submission to `optweight` and `optweight.svy`. Users should include one value per variable in `formula`. For factor variables, one value per level of the variable is required. The output of `check.targets` can also be used as an input to `targets` in `optweight` and `optweight.svy`.

Usage

```
check.targets(formula,
              data = NULL,
              targets,
              stop = FALSE)

## S3 method for class 'optweight.targets'
print(x, digits = 5, ...)
```

Arguments

`formula`	A formula with the covariates to be balanced with `optweight` 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`.
`targets`	A vector of target population means values for each covariate. These should be in the order corresponding to the order of the corresponding variable in `formula`, except for interactions, which will appear after all lower-order terms. For factor variables, a target value must be specified for each level of the factor, and these values must add up to 1. If empty, the current sample means will be produced. If `NULL`, an `NA` vector named with the covariate names will be produced.
`stop`	`logical`; if `TRUE`, an error will be thrown if the number of values in `targets` is not equal to the correct number of (expanded) covariates in `formula`, and no messages will be displayed if the `targets` input is satisfactory. If `FALSE`, a message will be displayed if the number of values in `targets` is not equal to the correct number of covariates in `formula`, and other messages will be displayed.
`x`	An `optweight.targets` object; the output of a call to `check.targets`.
`digits`	How many digits to print.
`...`	Ignored.

Details

The purpose of `check.targets` is to allow users to ensure that their proposed input to `targets` in `optweight` and `optweight.svy` is correct both in the number of entries and their order. This is especially important when factor variables and interactions are included in the formula because factor variables are split into several dummies and interactions are moved to the end of the variable list, both of which can cause some confusion and potential error when entering `targets` values.

Factor variables are internally split into a dummy variable for each level, so the user must specify a target population mean value for each level of the factor. These must add up to 1, and an error will be displayed if they do not. These values represent the proposrtion of units in the target population with each factor level.

Interactions (e.g., `a:b` or `a*b` in the `formula` input) are always sent to the end of the variable list even if they are specified elsewhere in the `formula`. It is important to run `check.targets` to ensure the order of the proposed `targets` corresponds to the represented order of covariates used in the formula. You can run `check.targets` with `targets = NULL` to see the order of covariates that is required without specifying any targets.

Value

An `optweight.targets` object, which is a named vector of target population mean values, one for each (expanded) covariate specified in `formula`. This should be used as user inputs to `optweight` and `optweight.svy`.

Author(s)

Noah Greifer

Examples

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

#Checking if the correct number of entries are included:
check.targets(treat ~ age + race + married +
                nodegree + re74,
                data = lalonde,
                targets = c(25, .4, .1, .5, .3,
                            .5, 4000))
#Notice race is split into three values (.4, .1, and .5)
```

---

check.tols *Construct and Check Tolerance Input*

Description

Checks whether proposed tolerance values for `tols` are suitable in number and order for submission to `optweight`. Users should include one value per item in `formula`. The output can also be used as an input to `tols` in `optweight`.

Usage

```
check.tols(formula,
           data = NULL,
           tols,
           stop = FALSE)

## S3 method for class 'optweight.tols'
print(x, internal = FALSE, digits = 5, ...)
```

Arguments

formula	A formula with the covariates to be balanced with optweight on the right hand side. See glm for more details. Interactions and functions of covariates are allowed. Lists of formulas are not allowed; multiple formulas must be checked one at a time.
data	An optional data set in the form of a data frame that contains the variables in formula.
tols	A vector of balance tolerance values in standardized mean difference units for each covariate. These should be in the order corresponding to the order of the corresponding variable in formula, except for interactions, which will appear after all lower-order terms. If only one value is supplied, it will be applied to all covariates.
stop	logical; if TRUE, an error will be thrown if the number of values in tols is not equal to the correct number of covariates in formula, and no messages will be displayed if the tols input is satisfactory. If FALSE, a message will be displayed if the number of values in tols is not equal to the correct number of covariates in formula, and other messages will be displayed.
x	An optweight.tols object; the output of a call to check.tols.
internal	logical; whether to print the tolerance values that are to be used internally by optweight. See Value section.
digits	How many digits to print.
...	Ignored.

Details

The purpose of check.tols is to allow users to ensure that their proposed input to tols in optweight is correct both in the number of entries and their order. This is especially important when factor variables and interactions are included in the formula because factor variables are split into several dummies and interactions are moved to the end of the variable list, both of which can cause some confusion and potential error when entering tols values.

Factor variables are internally split into a dummy variable for each level, but the user only needs to specify one tolerance value per original variable; check.tols automatically expands the tols input to match the newly created variables.

Interactions (e.g., a:b or a*b in the formula input) are always sent to the end of the variable list even if they are specified elsewhere in the formula. It is important to run check.tols to ensure the

order of the proposed `tols` corresponds to the represented order of covariates used in `optweight`. You can run `check.tols` with no `tols` input to see the order of covariates that is required.

`check.tols` was designed to be used primarily for its message printing and `print` method, but you can also assign its output to an object for use as an input to `tols` in `optweight`.

Note that only one formula and vector of tolerance values can be assessed at a time; for multiple treatment periods, each formula and tolerance vector must be entered seperately.

Value

An `optweight.tols` object, which is a named vector of tolerance values, one for each variable specified in `formula`. This should be used as user inputs to `optweight`. The `"internal.tols"` attribute contains the tolerance values to be used internally by `optweight`. These will differ from the vector values when there are factor variables that are split up; the user only needs to submit one tolerance per factor variable, but seperate tolerance values are produced for each new dummy created.

Author(s)

Noah Greifer

Examples

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

#Checking if the correct number of entries are included:
check.tols(treat ~ age + educ + married +
                nodegree + re74, data = lalonde,
                tols = c(.01, .02, .03, .04))

#Checking the order of interactions; notice they go
#at the end even if specified at the beginning. The
#.09 values are where the interactions might be expected
#to be, but they are in fact not.
c <- check.tols(treat ~ age:educ + married*race +
                nodegree + re74, data = lalonde,
                tols = c(.09, .01, .01, .09, .01, .01))

print(c, internal = TRUE)
```

`optweight` *Estimate Balancing Weights Using Optimization*

Description

Estimate balancing weights for treatments and covariates specified in `formula`. The degree of balance for each covariate is specified by `tols` and the target population can be specified with `targets` or `estimand`. See Zubizarreta (2015), Wang & Zubizarreta (2017), and Yiu & Su (2018) for details of the properties of the weights and the methods used to fit them.

Usage

```
optweight(formula,
          data = NULL,
          tols = 0,
          estimand = "ATE",
          targets = NULL,
          s.weights = NULL,
          focal = NULL,
          verbose = FALSE,
          force = FALSE,
          ...)

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

## S3 method for class 'optweightMSM'
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, or a list thereof. 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.
tols	A vector of balance tolerance values for each covariate, or a list thereof. The resulting weighted balance statistics will be at least as small as these values. If only one value is supplied, it will be applied to all covariates. Can also be the output of a call to check.tols for point treatments. See Details.
estimand	The desired estimand, which determines the target population. For binary treatments, can be "ATE", "ATT", "ATC", or NULL. For multinomial treatments, can be "ATE", "ATT", or NULL. For continuous treatments, can be "ATE" or NULL. The default for both is "ATE". For longitudinal treatments, only "ATE" is supported. estimand is ignored when targets is non-NULL. If both estimand and targets are NULL, no targeting will take place. See Details.
targets	A vector of target populaton mean values for each baseline covariate. The resulting weights will yield sample means within tols/2 units of the target values for each covariate. If NULL or all NA, estimand will be used to determine targets. Otherwise, estimand is ignored. If any target values are NA, the corresponding variable will not be targeted and its weighted mean will be wherever the weights yield the smallest variance. Can also be the output of a call to check.targets. See Details.
s.weights	A vector of sampling weights or the name of a variable in data that contains sampling weights. Optimization occurs on the product of the sampling weights and the estimated weights.

`focal`	When multinomial treatments are used and the "ATT" is 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.
`verbose`	Whether information on the optimization problem solution should be printed. This information contains how many iterations it took to estimate the weights and whether the solution is optimal.
`force`	optweights are currently not valid for use with longitudinal treatments, and will produce an error message if attempted. Set to `TRUE` to bypass this error message.
`...`	For `optweight`, arguments passed to `optweight.fit`. Ignored otherwise.
`x`	An `optweight` or `optweightMSM` object; the output of a call to `optweight()`.

Details

The optimization is performed by the lower-level function `optweight.fit` using `solve_osqp` in the **rosqp** package, which provides a straightforward interface to specifying the constraints and objective function for quadratic optimization problems and uses a fast and flexible solving algorithm.

For binary and multinomial treatments, weights are estimated so that the weighted mean differences of the covariates are within the given tolerance thresholds (unless `std.binary` or `std.cont` are `TRUE`, in which case standardized mean differences are considered for binary and continuous variables, respectively). For a covariate x with specified tolerance δ, the weighted means of each each group will be within δ of each other. Additionally, when the ATE is specified as the estimand or a target population is specified, the weighted means of each group will each be within $\delta/2$ of the target means; this ensures generalizability to the same population from which the original sample was drawn.

If standardized tolerance values are requested, the standardization factor corresponds to the estimand requested: when the ATE is requested or a target population specified, the standardization factor is the square root of the average variance for that covariate across treatment groups, and when the ATT or ATC are requested, the standardization factor is the standard deviation of the covariate in the focal group. The standardization factor is always unweighted.

For continuous treatments, weights are estimated so that the weighted correlation between the treatment and each covariate is within the specified tolerance threshold. If the ATE is requested or a target population is specified, the means of the weighted covariates and treatment are restricted to be equal to those of the target population to ensure generalizability to the desired target population. The weighted correlation is computed as the weighted covariance divided by the product of the *unweighted* standard deviations. The means used to center the variables in computing the covariance are those specified in the target population.

For longitudinal treatments, 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 in the documentation for `weightitMSM` in the **WeightIt** package. Currently, longtiduinal treatments are not recommended as optweight's use with them has not been validated.

Dual Variables: Two types of constriants may be associated with each covariate: target constraints and balance constraints. Target constraints require the mean of the covariate to be at (or near) a specific target value in each treatment group (or for the whole group when treatment is continuous). Balance constraints require the means of the covariate in pairs of treatments to be

near each other. For binary and multinomial treatments, balance constraints are redundant if target constraints are provided for a variable. For continuous variables, balance constraints refer to the correlation between treatment and the covariate and are not redundant with target constraints. In the `duals` component of the output, each covariate has a dual variable for each nonredundant constraint placed on it.

The dual variable for each constraint is the instantaneous rate of change of the objective function at the optimum due to a change in the constraint. Because this relationship is not linear, large changes in the constraint will not exactly map onto corresponding changes in the objective function at the optimum, but will be close for small changes in the constraint. For example, for a covariate with a balance constraint of .01 and a corresponding dual variable of .4, increasing (i.e., relaxing) the constraint to .025 will decrease the value of the objective function at the optimum by approximately (.025 - .01) * .4 = .006. When the L2 norm is used, this change corresponds to a change in the variance of the weights, which directly affects the effective sample size (though the magnitude of this effect depends on the original value of the effective sample size).

For factor variables, `optweight` takes the sum of the absolute dual variables for the constraints for all levels and reports it as the the single dual variable for the variable itself. This summed dual variable works the same way as dual variables for continuous variables do.

Solving Convergence Failure:

Sometimes the optimization will fail to converge at a solution. There are a variety of reasons why this might happen, which include that the constraints are nearly impossible to satisfy or that the optimization surface is relatively flat. It can be hard to know the exact cause or how to solve it, but this section offers some solutions one might try.

Rarely is the problem too few iterations, though this is possible. Most problems can be solved in the default 200,000 iterations, but sometimes it can help to increase this number with the `max_iter` argument. Usually, though, this just ends up taking more time without a solution found.

If the problem is that the constraints are too tight, it can be helpful to loosen the constraints. Sometimes examining the dual variables of a solution that has failed to converge can reveal which constraints are causing the problem.

Sometimes a suboptimal solution is possible; such a solution does not satisfy the constraints exactly but will come pretty close. To allow these solutions, the arguments `eps_abs` and `eps_rel` can be increased from 1E-8 to larger values. These should be adjusted together since they both must be satisfied for convergence to occur.

With continuous treatments, solutions that failed to converge may still be useable. Make sure to assess balance and examine the weights even after a optimal solution is not found, because the solution that is fond may be good enough.

Value

If only one time point is specified, an `optweight` 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.
`s.weights`	The provided sampling weights.
`estimand`	The estimand requested.

`focal`	The focal variable if the ATT was requested with a multinomial treatment.
`call`	The function call.
`tols`	The tolerance values for each covariate.
`duals`	A data.frame containing the dual variables for each covariate. See Details for interpretation of these values.
`info`	The `info` component of the output of `solve_osqp`, which contains information on the performance of the optimization at termination.

Otherwise, if multiple time points are specified, an `optmatchMSM` object with the following elements:

`weights`	The estimated weights, one for each unit.
`treat.list`	A list of the values of the treatment variables at each time point.
`covs.list`	A list of the covariates at each time point used in the fitting. Only includes the raw covariates, which may have been altered in the fitting process.
`s.weights`	The provided sampling weights.
`estimand`	The estimand requested; "ATE" for longtiduinal treatments.
`call`	The function call.
`tols`	A list of tolerance values for each covariate at each time point.
`duals`	A list of data.frames containing the dual variables for each covariate at each time point. See Details for interpretation of these values.
`info`	The `info` component of the output of `solve_osqp`, which contains information on the performance of the optimization at termination.

Author(s)

Noah Greifer

References

Anderson, E. (2018). rosqp: Quadratic Programming Solver using the 'OSQP' Library. R package version 0.1.0. https://CRAN.R-project.org/package=rosqp

Wang, Y., & Zubizarreta, J. R. (2017). Approximate Balancing Weights: Characterizations from a Shrinkage Estimation Perspective. ArXiv:1705.00998 [Math, Stat]. Retrieved from http://arxiv.org/abs/1705.00998

Yiu, S., & Su, L. (2018). Covariate association eliminating weights: a unified weighting framework for causal effect estimation. Biometrika. doi: 10.1093/biomet/asy015

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

See Also

https://osqp.org/docs/index.html for more information on **rosqp**, the underlying solver, and the options for `solve_osqp`.

`osqpSettings` for details on options for `solve_osqp`.

`optweight.fit`, the lower-level function that performs the fitting.

Examples

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

#Balancing covariates between treatment groups (binary)
(ow1 <- optweight(treat ~ age + educ + married +
                nodegree + re74, data = lalonde,
                tols = c(.01, .02, .03, .04, .05),
                estimand = "ATE"))
bal.tab(ow1)

#Exactly alancing covariates with respect to race (multinomial)
(ow2 <- optweight(race ~ age + educ + married +
                nodegree + re74, data = lalonde,
                tols = 0, estimand = "ATT", focal = "black"))
bal.tab(ow2)

# #Balancing covariates with longitudinal treatments
# #NOT VALID; DO NOT DO THIS.
# library("twang")
# data("iptwExWide")
#
# ##Weighting more recent covariates more strictly
# (ow3 <- optweight(list(tx1 ~ use0 + gender + age,
#                        tx2 ~ tx1 + use1 + use0  + gender +
#                          age,
#                        tx3 ~ tx2 + use2 + tx1 + use1 +
#                          use0 + gender + age),
#                   data = iptwExWide,
#                   tols = list(c(.001, .001, .001),
#                               c(.001, .001, .01, .01, .01),
#                               c(.001, .001, .01, .01,
#                                 .1, .1, .1))))
# bal.tab(ow3)

#Balancing covariates between treatment groups (binary)
#and requesting a specified target population
(ow4a <- optweight(treat ~ age + educ + married +
                nodegree + re74, data = lalonde,
                tols = 0,
                targets = c(26, 12, .4, .5, 1000),
                estimand = NULL))
bal.tab(ow4a, disp.means = TRUE)

#Balancing covariates between treatment groups (binary)
#and not requesting a target population
(ow4b <- optweight(treat ~ age + educ + married +
                nodegree + re74, data = lalonde,
                tols = 0,
                targets = NULL,
                estimand = NULL))
bal.tab(ow4b, disp.means = TRUE)
```

`optweight.fit` *Fitting Function for Optweight*

Description

`optweight.fit` performs the optimization (via **rosqp**; Anderson, 2018) for `optweight` and should, in most coses, not be used directly. No processing of inputs is performed, so they must be given exactly as described below.

Usage

```
optweight.fit(treat.list,
              covs.list,
              tols,
              estimand = "ATE",
              targets = NULL,
              s.weights = NULL,
              focal = NULL,
              norm = "l2",
              std.binary = FALSE,
              std.cont = TRUE,
              min.w = 1E-8,
              verbose = FALSE,
              force = FALSE,
              ...)
```

Arguments

`treat.list`	A list containing one vector of treatment statuses for each time point. Non-numeric (i.e., factor or character) vectors are allowed.
`covs.list`	A list containing one matrix of covariates to be balanced for each time point. All matrices must be numeric but do not have to be full rank.
`tols`	A list containing one vector of balance tolerance values for each time point.
`estimand`	The desired estimand, which determines the target population. For binary treatments, can be "ATE", "ATT", "ATC", or `NULL`. For multinomial treatments, can be "ATE", "ATT", or `NULL`. For continuous treatments, can be "ATE" or `NULL`. The default for both is "ATE". For longitudinal treatments, only "ATE" is supported. `estimand` is ignored when `targets` is non-`NULL`. If both `estimand` and `targets` are `NULL`, no targeting will take place. See Details.
`targets`	A vector of target populaton mean values for each baseline covariate. The resulting weights will yield sample means within `tols`/2 units of the target values for each covariate. If `NULL` or all `NA`, `estimand` will be used to determine targets. Otherwise, `estimand` is ignored. If any target values are `NA`, the corresponding variable will not be targeted and its weighted mean will be wherever the weights yield the smallest variance.

`s.weights` A vector of sampling weights. Optimization occurs on the product of the sampling weights and the estimated weights.

`focal` When multinomial treatments are used and the "ATT" is 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.

`norm` A string containing the name of the norm corresponding to the objective function to minimize. The options are `"l1"` for the L1 norm, `"l2"` for the L2 norm (the default), and `"linf"` for the L∞ norm. The L1 norm minimizes the average absolute distance between each weight and the mean of the weights; the L2 norm minimizes the variance of the weights; the L∞ norm minimizes the largest weight. The L2 norm has a direct correspondence with the effective sample size, making it ideal if this is your criterion of interest.

`std.binary, std.cont`
`logical`; whether the tolerances are in standardized mean units (`TRUE`) or raw units (`FALSE`) for binary variables and continuous variables, respectively. The default is `FALSE` for `std.binary` because raw proportion differences make more sense than standardized mean difference for binary variables. These arguments are analogous to the `binary` and `continuous` arguments in `bal.tab` in **cobalt**.

`min.w` A single `numeric` value between 0 and 1 for the smallest allowable weight. Some analyses require nonzero weights for all units, so a small, nonzero minimum may be desirable. Doing so will likely (slightly) increase the variance of the resulting weights depending on the magntiude of the minimum. The default is 1e-8, which does not materially change the properties of the weights from a minimum of 0 but prevents warnings in some packages that use weights to estimate treatment effects.

`verbose` Whether information on the optimization problem solution should be printed. This information contains how many iterations it took to estimate the weights and whether the solution is optimal.

`force` optweights are currently not valid for use with longitudinal treatments, and will produce an error message if attempted. Set to `TRUE` to bypass this error message.

`...` Options that are passed to `osqpSettings` for use in the `par` arguments of `solve_osqp`. See Details for defaults.

Details

`optweight.fit` transforms the inputs into the required inputs for `solve_osqp`, which are (sparse) matrices and vectors, and then supplies the outputs (the weights, duals variables, and convergence information) back to `optweight`. No processing of inputs is performed, as this is normally handled by `optweight`.

The default values for some of the parameters sent to `solve_osqp` are not the same as those in `osqpSettings`. The following are the differences: `max_iter` is set to 20000 and `eps_abs` and `eps_rel` are set to 1E-8 (i.e., 10^-8). All other values are the same.

Note that optweights with longitudinal treatments are not valid and should not be used until further research is done. `optweight.fit` will not provide any error messages, but

Value

An `optweight.fit` object with the following elements:

`w`	The estimated weights, one for each unit.
`duals`	A data.frame containing the dual variables for each covariate, or a list thereof. See Zubizarreta (2015) for interpretation of these values.
`info`	The `info` component of the output of `solve_osqp`, which contains information on the performance of the optimization at termination.

Author(s)

Noah Greifer

References

Anderson, E. (2018). rosqp: Quadratic Programming Solver using the 'OSQP' Library. R package version 0.1.0. https://CRAN.R-project.org/package=rosqp

Wang, Y., & Zubizarreta, J. R. (2017). Approximate Balancing Weights: Characterizations from a Shrinkage Estimation Perspective. ArXiv:1705.00998 [Math, Stat]. Retrieved from http://arxiv.org/abs/1705.00998

Yiu, S., & Su, L. (2018). Covariate association eliminating weights: a unified weighting framework for causal effect estimation. Biometrika. doi: 10.1093/biomet/asy015

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

See Also

`optweight` which you should use for estimating the balancing weights, unless you know better.

https://osqp.org/docs/index.html for more information on **rosqp**, the underlying solver, and the options for `solve_osqp`.

`osqpSettings` for details on options for `solve_osqp`.

Examples

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

treat.list <- list(lalonde$treat)
covs.list <- list(splitfactor(lalonde[2:8], drop.first = "if2"))
tols.list <- list(rep(.01, ncol(covs.list[[1]])))

ow.fit <- optweight.fit(treat.list,
                        covs.list,
                        tols = tols.list,
                        estimand = "ATE",
                        norm = "l2")
```

optweight.svy *Estimate Targeting Weights Using Optimization*

Description

Estimate targeting weights for covariates specified in `formula`. The target means are specified with `targets` and the maximum distance between each weighted covariate mean and the corresponding target mean is specified by `tols`. See Zubizarreta (2015) for details of the properties of the weights and the methods used to fit them.

Usage

```
optweight.svy(formula,
              data = NULL,
              tols = 0,
              targets = NULL,
              s.weights = NULL,
              verbose = FALSE,
              ...)

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

Arguments

`formula`	A formula with nothing on the left hand side and the covariates to be targeted 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`.
`tols`	A vector of target balance tolerance values for each covariate. The resulting weighted covariate means will be no further away from the targets than the specified values. If only one value is supplied, it will be applied to all covariates. Can also be the output of a call to `check.tols`. See Details.
`targets`	A vector of target populaton mean values for each covariate. The resulting weights will yield sample means within `tols` units of the target values for each covariate. If any target values are `NA`, the corresponding variable will not be targeted and its weighted mean will be wherever the weights yield the smallest variance. To ensure the weighted mean for a covairate is equal to its unweighted mean (i.e., so that its orginal mean is its target mean), its original mean must be supplied as a target. See Details.
`s.weights`	A vector of sampling weights or the name of a variable in `data` that contains sampling weights. Optimization occurs on the product of the sampling weights and the estimated weights.

verbose	Whether information on the optimization problem solution should be printed. This information contains how many iterations it took to estimate the weights and whether the solution is optimal.
...	For optweight.svy, arguments passed to optweight.svy.fit. Ignored otherwise.
x	An optweight.svy object; the output of a call to optweight.svy().

Details

The optimization is performed by the lower-level function optweight.svy.fit using solve_osqp in the **rosqp** package, which provides a straightforward interface to specifying the constraints and objective function for quadratic optimization problems and uses a fast and flexible solving algorithm.

Weights are estimated so that the standardized differences between the weighted covariate means and the corresponding targets are within the given tolerance thresholds (unless std.binary or std.cont are FALSE, in which case unstandardized mean differences are considered for binary and continuous variables, respectively). For a covariate x with specified tolerance δ, the weighted mean will be within δ of teh target. If standardized tolerance values are requested, the standardization factor is the standard deviation of the covariate in the whole sample. The standardization factor is always unweighted.

Dual Variables: Each covariate is associated with a target constraint. In the duals component of the output, each covariate has a dual variable for the constraint placed on it. The dual variable for each constraint is the instantaneous rate of change of the objective function at the optimum due to a change in the constraint. Because this relationship is not linear, large changes in the constraint will not exactly map onto corresponding changes in the objective function at the optimum, but will be close for small changes in the constraint. For example, for a covariate with a target balance constraint of .01 and a corresponding dual variable of .4, increasing (i.e., relaxing) the constraint to .025 will decrease the value of the objective function at the optimum by approximately (.025 - .01) * .4 = .006. When the L2 norm is used, this change corresponds to a change in the variance of the weights, which directly affects the effective sample size (though the magntidue of this effect depends on the original value of the effective sample size).

For factor variables, optweight takes the sum of the absolute dual variables for the constraints for all levels and reports it as the the single dual variable for the variable itself. This summed dual variable works the same way as dual variables for continuous variables do.

Solving Convergence Failure:

Sometimes the opimization will fail to converge at a solution. There are a variety of reasons why this might happen, which include that the constraints are nearly impossible to satisfy or that the optimization surface is relatively flat. It can be hard to know the exact cause or how to solve it, but this section offers some solutions one might try.

Rarely is the problem too few iterations, though this is possible. Most problems can be solved in the default 200,000 iterations, but sometimes it can help to increase this number with the max_iter argument. Usually, though, this just ends up taking more time without a solution found.

If the problem is that the constraints are too tight, it can be helpful to loosen the constraints. Sometimes examining the dual variables of a solution that has failed to converge can reveal which constraints are causing the problem.

Sometimes a suboptimal solution is possible; such a solution does not satisfy the constraints exactly but will come pretty close. To allow these solutions, the arguments `eps_abs` and `eps_rel` can be increased from 1E-9 to larger values. These should be adjusted together since they both must be satisfied for converge to occur.

Value

An `optweight.svy` object with the following elements:

`weights`	The estimated weights, one for each unit.
`covs`	The covariates used in the fitting. Only includes the raw covariates, which may have been altered in the fitting process.
`s.weights`	The provided sampling weights.
`call`	The function call.
`tols`	The tolerance values for each covariate.
`duals`	A data.frame containing the dual variables for each covariate. See Details for interpretation of these values.
`info`	The `info` component of the output of `solve_osqp`, which contains information on the performance of the optimization at termination.

Author(s)

Noah Greifer

References

Anderson, E. (2018). rosqp: Quadratic Programming Solver using the 'OSQP' Library. R package version 0.1.0. https://CRAN.R-project.org/package=rosqp

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

See Also

https://osqp.org/docs/index.html for more information on **rosqp**, the underlying solver, and the options for `solve_osqp`.

`osqpSettings` for details on options for `solve_osqp`.

`optweight.svy.fit`, the lower-level function that performs the fitting.

Examples

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

cov.formula <- ~ age + educ + race + married +
                      nodegree

targets <- check.targets(cov.formula, data = lalonde,
```

```
                        targets = c(23, 9, .3, .3, .4,
                                    .2, .5))

tols <- check.tols(cov.formula, data = lalonde,
                   tols = 0)

ows <- optweight.svy(cov.formula,
                     data = lalonde,
                     tols = tols,
                     targets = targets)
ows

covs <- splitfactor(lalonde[c("age", "educ", "race",
                   "married", "nodegree")],
                   drop.first = FALSE)
#Unweighted means
apply(covs, 2, mean)

#Weighted means; same as targets
apply(covs, 2, weighted.mean, w = ows$weights)
```

`optweight.svy.fit` *Fitting Function for Optweight for Survey Weights*

Description

`optweight.svy.fit` performs the optimization (via **rosqp**; Anderson, 2018) for `optweight.svy` and should, in most coses, not be used directly. No processing of inputs is performed, so they must be given exactly as described below.

Usage

```
optweight.svy.fit(covs,
                  tols = 0,
                  targets,
                  s.weights = NULL,
                  norm = "l2",
                  std.binary = FALSE,
                  std.cont = TRUE,
                  min.w = 1E-8,
                  verbose = FALSE,
                  ...)
```

Arguments

`covs`	A matrix of covariates to be targeted. Should must be numeric but does not have to be full rank.
`tols`	A vector of target balance tolerance values.

`targets` A vector of target populaton mean values for each covariate. The resulting weights will yield sample means within `tols` units of the target values for each covariate. If any target values are `NA`, the corresponding variable will not be targeted and its weighted mean will be wherever the weights yield the smallest variance. To ensure the weighted mean for a covairate is equal to its unweighted mean (i.e., so that its orginal mean is its target mean), its original mean must be supplied as a target.

`s.weights` A vector of sampling weights. Optimization occurs on the product of the sampling weights and the estimated weights.

`norm` A string containing the name of the norm corresponding to the objective function to minimize. The options are `"l1"` for the L1 norm, `"l2"` for the L2 norm (the default), and `"linf"` for the L∞ norm. The L1 norm minimizes the average absolute distance between each weight and the mean of the weights; the L2 norm minimizes the variance of the weights; the L∞ norm minimizes the largest weight. The L2 norm has a direct correspondence with the effective sample size, making it ideal if this is your criterion of interest.

`std.binary, std.cont`
`logical`; whether the tolerances are in standardized mean units (`TRUE`) or raw units (`FALSE`) for binary variables and continuous variables, respectively. The default is `FALSE` for `std.binary` because raw proportion differences make more sense than standardized mean difference for binary variables.

`min.w` A single `numeric` value between 0 and 1 for the smallest allowable weight. Some analyses require nonzero weights for all units, so a small, nonzero minimum may be desirable. Doing so will likely (slightly) increase the variance of the resulting weights depending on the magntiude of the minimum. The default is 1e-8, which does not materially change the properties of the weights from a minimum of 0 but prevents warnings in some packages that use weights to estimate treatment effects.

`verbose` Whether information on the optimization problem solution should be printed. This information contains how many iterations it took to estimate the weights and whether the solution is optimal.

`...` Options that are passed to `osqpSettings` for use in the `par` arguments of `solve_osqp`.

Details

`optweight.svy.fit` transforms the inputs into the required inputs for `solve_osqp`, which are (sparse) matrices and vectors, and then supplies the outputs (the weights, duals variables, and convergence information) back to `optweight.svy`. No processing of inputs is performed, as this is normally handled by `optweight.svy`.

Value

An `optweight.svy.fit` object with the following elements:

`w` The estimated weights, one for each unit.

`duals` A data.frame containing the dual variables for each covariate. See Zubizarreta (2015) for interpretation of these values.

`info` The `info` component of the output of `solve_osqp`, which contains information on the performance of the optimization at termination.

Author(s)

Noah Greifer

References

Anderson, E. (2018). rosqp: Quadratic Programming Solver using the 'OSQP' Library. R package version 0.1.0. https://CRAN.R-project.org/package=rosqp

Wang, Y., & Zubizarreta, J. R. (2017). Approximate Balancing Weights: Characterizations from a Shrinkage Estimation Perspective. ArXiv:1705.00998 [Math, Stat]. Retrieved from http://arxiv.org/abs/1705.00998

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

See Also

`optweight.svy` which you should use for estimating the balancing weights, unless you know better.

https://osqp.org/docs/index.html for more information on **rosqp**, the underlying solver, and the options for `solve_osqp`.

`osqpSettings` for details on options for `solve_osqp`.

Examples

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

covs <- splitfactor(lalonde[c("age", "educ", "race",
                  "married", "nodegree")],
                  drop.first = FALSE)

targets <- c(23, 9, .3, .3, .4, .2, .5)

tols <- rep(0, 7)

ows.fit <- optweight.svy.fit(covs,
                             tols = tols,
                             targets = targets,
                             norm = "l2")

#Unweighted means
apply(covs, 2, mean)

#Weighted means; same as targets
apply(covs, 2, weighted.mean, w = ows.fit$w)
```

`plot.optweight` *Plot Dual Variables for Assessing Balance Constraints*

Description

Plots the dual variables resulting from `optweight` in a way similar to figure 2 of Zubizarreta (2015), which explained how to interpret these values. These represent the cost of changing the constraint on the variance of the resulting weights. For covariates with large values of the dual variable, tightening the constraint will increase the variability of the weights, and loosening the constraint will decrease the variability of the weights, both to a greater extent than would doing the same for covariate with small values of the dual variable.

Usage

```
## S3 method for class 'optweight'
plot(x, which.time = 1, ...)

## S3 method for class 'optweight.svy'
plot(x, ...)
```

Arguments

`x`	An `optweight` or `optweight.svy` object; the output of a call to `optweight` or `optweight.svy`.
`which.time`	For longitudinal treatments, which time period to display. Only one may be displayed at a time.
`...`	Ignored.

Value

A `ggplot` object that can be used with other **ggplot2** functions.

Author(s)

Noah Greifer

References

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

See Also

`optweight` or `optweight.svy` to estimate the weights and the dual variables `plot.summary.optweight` for plots of the distribution of weights

Examples

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

#Balancing covariates between treatment groups (binary)
ow1 <- optweight(treat ~ age + educ + married +
                 nodegree + re74, data = lalonde,
                 tols = c(.1, .1, .1, .1, .1),
                 estimand = "ATT")

summary(ow1) # Note the coefficient of variation
             # and effective sample size (ESS)

plot(ow1) # age has a low value, married is high

ow2 <- optweight(treat ~ age + educ + married +
                 nodegree + re74, data = lalonde,
                 tols = c(0, .1, .1, .1, .1),
                 estimand = "ATT")

summary(ow2) # Notice that tightening the constraint
             # on age had a negligible effect on the
             # variability of the weights and ESS

ow3 <- optweight(treat ~ age + educ + married +
                 nodegree + re74, data = lalonde,
                 tols = c(.1, .1, 0, .1, .1),
                 estimand = "ATT")

summary(ow3) # In contrast, tightening the constraint
             # on married had a large effect on the
             # variability of the weights, shrinking
             # the ESS
```

`summary.optweight` *Summarize, print, and plot information about estimated weights*

Description

These functions summarize the weights resulting from a call to `optweight` or `optweight.svy`. `summary` produces summary statistics on the distribution of weights, including their range and variability, and the effective sample size of the weighted sample (computing using the formula in McCaffrey, Rudgeway, & Morral, 2004). `plot` creates a histogram of the weights.

Usage

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

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

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

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

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

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

## S3 method for class 'summary.optweight'
plot(x, ...)
```

Arguments

`object`	An `optweight`, `optweightMSM`, or `optweight.svy` object; the output of a call to `optweight` or `optweight.svy`.
`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.
`x`	A `summary.optweight`, `summary.optweightMSM`, or `summary.optweight.svy` object; the output of a call to `summary.optweight`, `summary.optweightMSM`, or `summary.optweight.svy`.
`...`	Additional arguments. 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., `optweight` objects), `summary` returns a `summary.optweight` 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`	The coefficient of variation (standard deviation divided by mean) of the weights in each treatment group and overall. When no sampling weights are used, this is simply the standard deviation of the weights.
`mean.abs.dev`	The mean absolute deviation of the weights in each treatment group and overall.

`effective.sample.size`
: The effective sample size for each treatment group before and after weighting.

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

For `optweight.svy` objects, a list of the above elements but with no treatment group divisions.

`plot` returns a `ggplot` object with a histogram displaying the distribution of the estimaed 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 (usually 1).

Author(s)

Noah Greifer

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

See Also

`plot.optweight` for plotting the values of the dual variables.

Examples

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

#Balancing covariates between treatment groups (binary)
(ow1 <- optweight(treat ~ age + educ + married +
                nodegree + re74, data = lalonde,
                tols = .001,
                estimand = "ATT"))

(s <- summary(ow1))

plot(s, breaks = 12)
```

Index

bal.tab, *12*

check.targets, 2, *6*
check.tols, 3, *6*, *14*

geom_histogram, *22*
glm, *2*, *4*, *6*, *14*

hist, *22*

optweight, *2–5*, 5, *13*, *20–22*
optweight.fit, *7*, *9*, 11
optweight.svy, *2*, *3*, 14, *19–22*
optweight.svy.fit, *15*, *16*, 17
osqpSettings, *9*, *12*, *13*, *16*, *18*, *19*

plot.optweight, 20, *23*
plot.summary.optweight, *20*
plot.summary.optweight
        *(*summary.optweight*)*, 21
print.optweight *(*optweight*)*, 5
print.optweight.svy *(*optweight.svy*)*, 14
print.optweight.targets
        *(*check.targets*)*, 2
print.optweight.tols *(*check.tols*)*, 3
print.optweightMSM *(*optweight*)*, 5
print.summary.optweight
        *(*summary.optweight*)*, 21
print.summary.optweightMSM
        *(*summary.optweight*)*, 21

reshape, *7*

solve_osqp, *7*, *9*, *12*, *13*, *15*, *16*, *18*, *19*
summary.optweight, 21
summary.optweightMSM
        *(*summary.optweight*)*, 21