Package 'cobalt'

January 16, 2019

Title Covariate Balance Tables and Plots

Version 3.6.1

Author Noah Greifer [aut, cre]

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

Description Generate balance tables and plots for covariates of groups preprocessed through matching, weighting or subclassification, for example, using propensity scores. Includes integration with 'MatchIt', 'twang', 'Matching', 'optmatch', 'CBPS', 'ebal', 'WeightIt', and 'designmatch' for assessing balance on the output of their preprocessing functions. Users can also specify data for balance assessment not generated through the above packages. Also included are methods for assessing balance in clustered or multiply imputed data sets or data sets with longitudinal treatments.

Depends R (>= 3.3.0)

Imports ggplot2 (>= 3.0.0), ggstance, crayon, backports (>= 1.1.1), methods

Suggests MatchIt, WeightIt (>= 0.5.0), twang, Matching, optmatch, ebal, CBPS (>= 0.17), designmatch, optweight, mice, mlogit (>= 0.3-0), knitr, rmarkdown

License GPL (>= 2)

Encoding UTF-8

LazyData true

VignetteBuilder knitr

URL https://github.com/ngreifer/cobalt

BugReports https://github.com/ngreifer/cobalt/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2019-01-16 05:50:03 UTC

R topics documented:

`bal.plot` *Generate Univariate Balance Plots*

Description

Generates density plots, bar graphs, or scatterplots displaying distributional balance between treatment and covariates using **ggplot2**.

Usage

```
bal.plot(obj,
    var.name,
    ...,
    which,
    which.sub = NULL,
    cluster = NULL,
    which.cluster = NULL,
    imp = NULL,
    which.imp = NULL,
```

```
    which.treat = NULL,
    which.time = NULL,
    size.weight = FALSE,
    mirror = FALSE,
    type = c("density", "histogram"),
    colors = NULL)
```

Arguments

obj	the object for which balance is to be assessed; can be any object for which there is support in bal.tab.
var.name	character; the name of the variable whose values are to be plotted. To view distributions of the distance measure (e.g., propensity score), if any, use "distance" as the argument unless the distance variable has been named. If there are duplicate variable names across inputs, bal.plot() will first look in the covariate data.frame from obj, followed by addl, and then distance, if any. If not specified, will use the first covariate available with a warning.
...	other arguments to define the variable, treatment, and weights. Some inputs are required depending on the method. See Additional Arguments. Can also be used to supply the bw, adjust, kernel, and n arguments for geom_density and the bins argument for geom_histogram.
which	whether to display distributional balance for the adjusted ("adjusted") or unadjusted sample ("unadjusted") or both at the same time ("both"). The default is to display balance for the adjusted sample only unless no weights, subclasses, or matching strata are specified. Abbreviations allowed.
which.sub	numeric; if subclassification is used, a vector corresponding to the subclass(es) for which the distributions are to be displayed. If NULL (the default), distributions from all subclasses are displayed in a grid.
cluster	optional; a vector of cluster membership, or the name of a variable in an available data set passed to bal.plot() that contains cluster membership.
which.cluster	if clusters are used, which cluster(s) to display. Can be cluster names or numerical indices for which to display balance. Indices correspond to the alphabetical order of cluster names. If NULL (the default) or NA, all clusters are displayed.
imp	optional; a vector of imputation indices, or the name of a variable in an available data set passed to bal.plot() that contains imputation indices.
which.imp	if imputations are used, which imputations(s) to display. Must be numerical indices for which to display balance. If NULL (the default) or NA, all imputations are displayed.
which.treat	which treatment groups to display. If NULL (the default) or NA, all treatment groups are displayed.
which.time	for longitudinal treatments, which time points to display. Can be treatment names or time period indices. If NULL (the default) or NA, all time points are displayed.
size.weight	logical; if both the treatment and the covariate are continuous, whether points should be sized according to their weight.

mirror	logical; if the treatment is binary, whether to display mirrored densities/histograms or overlapping densities/histograms. Ignored otherwise.
type	for binary and multinomial treatments, whether to display histograms or densities. The default is to display densities. Abbreviations are allowed.
colors	a vector of colors for the plotted densities/histograms. See 'Color Specification' at par. Defaults to the default **ggplot2** colors.

Details

bal.plot() uses ggplot() from the **ggplot2** package, and (invisibly) returns a "ggplot" object. For categorical treatments with continuous covariates or continuous treatments with categorical covariates, density plots are created using **ggplot2**'s geom_density() method; for categorical treatments with categorical covariates, bar graphs are created using **ggplot2**'s geom_bar() method; for continuous treatments with continuous covariates, scatterplots are created using **ggplot2**'s geom_point() method.

For continuous treatments with continuous covariates, three additional lines are presented for aid in balance assessment. The dashed blue line is the linear fit line. The solid blue line is a Loess fit curve generated with **ggplot2**'s geom_smooth(method = "loess"). The solid black line is a horizontal reference line intercepting the (weighted) treatment mean. Proximity of the fit lines to the reference line is indicative of independence between the covariate and treatment variable.

When multiple plots are to be displayed (i.e., when requesting subclass balance, cluster balance, or imputation balance, or when multiple sets of weights are provided or which = "both", or when treatment is longitudinal), the plots will be displayed in a grid using **ggplot2**'s facet_grid(). Subclassification cannot be used with clusters or multiply imputed data.

Value

A "ggplot" object, returned invisibly.

Additional Arguments

bal.plot() works like bal.tab() in that it can take a variety of types of inputs and yield the same output for each. Depending on what kind of input is given, different additional parameters are required in .... For details on what is required and allowed for each additional input and their defaults, see the help file for the bal.tab method associated with the input. The following are the required additional arguments based on each input type (optional arguments in parentheses):

For matchit objects: None

For weightit objects: None

For ps, ps.cont, mnps, and iptw objects: (stop.method; see defaults).

For Match objects: formula and data or covs and treat.

For optmatch objects: formula and data or covs and treat.

For CBPS objects: None

For ebalance objects: formula and data or covs and treat.

For formulas: data, weights, (distance), (method; see defaults)

For data.frames: treat, (data), weights, (distance), (method; see defaults)

For designmatch objects: formula and data or covs and treat.

For other objects processed through bal.tab's default method, whichever arguments are required to identify treatment, variables, and a conditioning method (if any).

Author(s)

Noah Greifer

See Also

bal.tab

Examples

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

#Nearest Neighbor Matching
m.out <- matchit(treat ~ age + educ + race +
                 married + nodegree + re74 + re75,
                 data = lalonde)

bal.plot(m.out, "age", which = "both")
bal.plot(m.out, "race", which = "both")
bal.plot(m.out, "distance", which = "both", mirror = TRUE,
         type = "histogram", colors = c("white", "black"))
```

---

bal.tab *Generates Balance Statistics*

Description

Generates balance statistics on covariates in relation to an observed treatment variable. It is a generic function that dispatches to the method corresponding to the class of the first argument. This page links to each method page and documents the calculation and details of aspects all bal.tab methods use. For information on the use of bal.tab with specific types of objects, use the following links:

- bal.tab.matchit for the method for objects returned by **MatchIt**.

- bal.tab.weightit for the method for weightit and weightitMSM objects returned by **WeightIt**.

- bal.tab.ps for the method for ps, mnps, and iptw objects returned by **twang** and for ps.cont objects returned by **WeightIt**.

- bal.tab.Match for the method for objects returned by **Matching**.

- bal.tab.optmatch for the method for objects returned by **optmatch**.

- bal.tab.CBPS for the method for objects returned by **CBPS**.

- bal.tab.ebalance for the method for objects returned by **ebal**.

- bal.tab.designmatch for the method for objects returned by **designmatch**.

- bal.tab.formula and bal.tab.data.frame for the methods for formula and data frame interfaces when the user has covariate values and weights (including matching weights) or subclasses or

wants to evaluate balance on an unconditioned data set. For data that corresponds to a longitudinal treatment (i.e., to be analyzed with a marginal structural model), see `bal.tab.time.list`.

- `bal.tab.default` for the method for objects that do not come from one of the explicitly supported conditioning packages.

Usage

```
bal.tab(...)
```

Arguments

`...`	Arguments passed to other methods. These arguments may be data-related, computation-related, or print-related. The first argument must be the object corresponding to the method to be dispatched; for example, the output of a call to a balancing function in another package or a formula or data frame.

Details

`bal.tab()` performs various calculations on the the data objects given, and some of these calculations are not transparent on the help pages of the individual methods. This page details the calculations that are used across `bal.tab` methods.

With Binary Point Treatments:

Prior to computation, all variables are checked for variable type, which allows users to differentiate balance statistic calculations based on type using the arguments to `continuous` and `binary`. First, if a given covariate is numeric and has only 2 levels, it is converted into a binary (0,1) variable. If 0 is a value in the original variable, it retains its value and the other value is converted to 1; otherwise, the lower value is converted to 0 and the other to 1. Next, if the covariate is not numeric or logical (i.e., is a character or factor variable), it will be split into new binary variables, named with the original variable and the value, separated by an underscore. Otherwise, the covariate will be used as is and treated as a continuous variable.

The default balance statistic for mean differences for continuous variables is the standardized difference, which is the difference in the means divided by a measure of spread (i.e., a d-type effect size measure). This is the default because it puts the mean differences on the same scale for comparison with each other and with a given threshold. It can be helpful to see the raw mean differences if the analyst is familiar with the true significance of the mean difference irrespective of the spread of the variable in the sample. For binary variables, the default balance statistic is the raw difference in proportion. Although standardized differences can be computed, proportion differences for binary variables are already on the same scale, and computing the standardized difference can obscure the true difference in proportion by dividing the difference in proportion by a number that is itself a function of the observed proportions. For example, if $XT = .2$ and $XC = .3$, the standardized difference in proportion would be different from that if $XT = .5$ and $XC = .6$, which seems counterintuitive and not a useful distinction. However, to remain in line with the methodological literature and for comparability with other balance assessment tools (e.g., MatchIt's `summary()`), the option to use standardized differences for binary variables remains.

Standardized differences are calculated as follows: the numerator is the mean of the treated group minus the mean of the control group, and the denominator is a measure of spread calculated in accordance with the argument to `s.d.denom` or the default of the specific method used. Common approaches in the literature include using the standard deviation of the treated group or using the

"pooled" standard deviation (i.e., the square root of the mean of the group variances) in calculating standardized mean differences. The computed spread `bal.tab()` uses is always that of the full, unadjusted sample (i.e., before matching, weighting, or subclassification), as recommended by Stuart (2010) and instituted in **MatchIt**, though some analysts and other packages use the spread of the sample in question (i.e., before or after adjustment). One reason to favor the use of the spread of the unadjusted sample is that it prevents the paradoxical situation of adjustment decreasing both the mean difference and the spread of the sample, yielding a larger standardized mean difference than that prior to adjusting even though the adjusted groups are now more similar.

When `m.threshold` is specified, it is compared to the absolute mean difference. `bal.tab()` takes the absolute value of the input to `m.threshold` as well before comparing.

Variance ratios are computed within-sample, with the larger of the two variances in the numerator, always yielding values greater than or equal to 1. Variance ratios are not calculated for binary variables since they are only a function of the group proportions and thus provide the same information as differences in proportion. `bal.tab()` takes the reciprocal of the input to `v.threshold` if it is less than 1.

When matching is used for conditioning, the presented sample sizes are calculated simply by summing the number of observations in each group with matching weights greater than 0. When weighting is used, an "effective sample size" is calculated for each group using the following formula: $(\sum w)^2 / \sum w^2$, as is used in **twang**. The effective sample size is "approximately the number of observations from a simple random sample that yields an estimate with sampling variation equal to the sampling variation obtained with the weighted comparison observations" (Ridgeway et al., 2016). The calculated number tends to underestimate the true effective sample size of the weighted samples. The number depends on the variability of the weights, so sometimes trimming units with large weights can actually increase the effective sample size, even though units are being down-weighted.

When subclassification is used, the balance tables for each subclass stored in `$Subclass.Balance` use values calculated as described above. For the aggregate balance table stored in `$Balance.Across.Subclass`, the values of each statistic are computed as a weighted average of the statistic across subclasses, weighted by the proportion of units in each subclass. See `bal.tab.subclass` for more details.

With Continuous Point Treatments:

When continuous treatment variables are considered, the balance statistic calculated is the Pearson correlation between the covariate and treatment. The correlation after adjustment is computed as the weighted covariance between the covariate and treatment divided by the product of the standard deviations of the unweighted covariate and treatment, in an analogous way to how how the weighted standardized mean difference uses an unweighted measure of spread in its denominator, with the purpose of avoiding the analogous paradox (i.e., where the covariance decreases but is accompanied by a change in the standard deviations, thereby distorting the actual resulting balance computed using the weighted standard deviations).

With Multinomial Point Treatments:

For information on using `bal.tab()` with multiple categorical treatments, see `bal.tab.multi`. Essentially, `bal.tab()` compares pairs of treatment groups in a standard way.

With Longitudinal Treatments:

For information on using `bal.tab()` with longitudinal treatments, see `bal.tab.msm`. Essentially, `bal.tab()` summarizes balance at each time point and summarizes across time points.

With Clustered or Multiply Imputed Data:

For information on using `bal.tab()` with clustered data, see `bal.tab.cluster`. For information on using `bal.tab()` with multiply imputed data, see `bal.tab.imp`.

Quick:

Calculations can take some time, especially when there are many variables, interactions, or clusters. One reason for this is that `bal.tab()` computes all values that it can, even if they are not requested by the user. For example, even if `un = FALSE`, which is the default, values for the unadjusted samples are still calculated. This can be useful if the output is to be further examined with `print()` or `love.plot()` or is to be used in some other way after the original call to `bal.tab()`. To avoid these extra calculations if they are not needed, users can set `quick = TRUE`, which will often quite dramatically speed up calculation and still display all statistics the user requests. For simple and quick model comparisons, it may be quite useful to do so, but for model reporting and graphical displays, it may be more useful to leave `quick = FALSE`, which is the default.

Missing Data:

If there is missing data in the covariates (i.e., `NA`s in the covariates provided to `bal.tab()`), a few additional things happen. A warning will appear mentioning that missing values were present in the data set. The computed balance summaries will be for the variables ignoring the missing values. New variables will be created representing missingness indicators for each variable, named `var:<NA>` (with `var` replaced by the actual name of the variable). If `int = TRUE`, balance for the pairwise interactions between the missingness indicators will also be computed. These variables are treated like regular variables once created.

Value

An object of class `"bal.tab"`. The use of continuous treatments, subclasses, clusters, and/or imputations will also cause the object to inherit other classes. The class `"bal.tab"` has its own `print` method (`print.bal.tab`), which formats the output nicely and in accordance with print-related options given in the call to `bal.tab()`, and which can be called with its own options. Each inherited class also has its own `print` method.

For scenarios with binary point treatments and no subclasses, imputations, or clusters, the following are the elements of the `bal.tab` object:

`Balance` A data frame containing balance information for each covariate. Balance contains the following columns:

- `Type`: Whether the covariate is binary, continuous, or a measure of distance (e.g., the propensity score).
- `M.0.Un`: The mean of the control group prior to adjusting.
- `SD.0.Un`: The standard deviation of the control group prior to adjusting.
- `M.1.Un`: The mean of the treated group prior to adjusting.
- `SD.1.Un`: The standard deviation of the treated group prior to adjusting.
- `Diff.Un`: The (standardized) difference in means between the two groups prior to adjusting.
- `V.Ratio.Un`: The ratio of the variances of the two groups prior to adjusting. `NA` for binary variables. If less than 1, the reciprocal is reported.
- `KS.Un`: The KS statistic of the two groups prior to adjusting. `NA` for binary variables.

- M.0.Adj: The mean of the control group after adjusting.
- SD.0.Adj: The standard deviation of the control group after adjusting.
- M.1.Adj: The mean of the treated group after adjusting.
- SD.1.Adj: The standard deviation of the treated group after adjusting.
- Diff.Adj: The (standardized) difference in means between the two groups after adjusting.
- M.Threshold: Whether or not the calculated mean difference after adjusting exceeds or is within the threshold given by m.threshold. If m.threshold is NULL, this column will be NA.
- V.Ratio.Adj: The ratio of the variances of the two groups after adjusting. NA for binary variables. If less than 1, the reciprocal is reported.
- V.Threshold: Whether or not the calculated variance ratio after adjusting exceeds or is within the threshold given by v.threshold for continuous variables. If v.threshold is NULL, this column will be NA.
- KS.Adj: The KS statistic of the two groups after adjusting. NA for binary variables.
- KS.Threshold: Whether or not the calculated KS statistic after adjusting exceeds or is within the threshold given by ks.threshold for continuous variables. If ks.threshold is NULL, this column will be NA.

Balanced.Means
: If m.threshold is specified, a table tallying the number of variables that exceed or are within the threshold for mean differences.

Max.Imbalance.Means
: If m.threshold is specified, a table displaying the variable with the greatest absolute mean difference.

Balanced.Variances
: If v.threshold is specified, a table tallying the number of variables that exceed or are within the threshold for variance ratios.

Max.Imbalance.Variance
: If v.threshold is specified, a table displaying the variable with the greatest variance ratio.

Balanced.KS
: If ks.threshold is specified, a table tallying the number of variables that exceed or are within the threshold for KS statistics.

Max.Imbalance.KS
: If ks.threshold is specified, a table displaying the variable with the greatest KS statistic.

Observations
: A table displaying the sample sizes before and after adjusting.

call
: The original function call, if adjustment was performed by a function in another package.

print.options
: A list of print options passed to print.bal.tab.

If the treatment is continuous, instead of producing mean differences, variance ratios, and KS statistics, bal.tab() will produce correlations between the covariates and the treatment. The corresponding entries in the output will be "Corr.Un", "Corr.Adj", and "R.threshold" (and accordingly for the balance tally and maximum imbalance tables).

If multiple weights are supplied, "Adj" in Balance will be replaced by the provided names of the sets of weights, and extra columns will be added for each set of weights. Additional columns and rows for other items in the output will be created as well.

For bal.tab output with subclassification, see bal.tab.subclass.

Author(s)

Noah Greifer

References

Ridgeway, G., McCaffrey, D., Morral, A., Burgette, L., & Griffin, B. A. (2016). Toolkit for Weighting and Analysis of Nonequivalent Groups: A tutorial for the twang package. R vignette. RAND.

Stuart, E. A. (2010). Matching Methods for Causal Inference: A Review and a Look Forward. Statistical Science, 25(1), 1-21. doi: 10.1214/09STS313

---

bal.tab.CBPS *Balance statistics for CBPS Objects*

Description

Generates balance statistics for CBPS and CBMSM objects from the **CBPS** package. Note that several arguments that used to be documented here are now documented in display options. They are still available.

Usage

```
## S3 method for class 'CBPS'
bal.tab(cbps,
    int = FALSE,
    poly = 1,
    distance = NULL,
    addl = NULL,
    data = NULL,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
    v.threshold = NULL,
    ks.threshold = NULL,
    r.threshold = NULL,
    cluster = NULL,
    pairwise = TRUE,
    focal = NULL,
    s.weights = NULL,
    abs = FALSE,
    subset = NULL,
```

```
    quick = FALSE,
    ...)
```

Arguments

cbps	a CBPS or CBMSM object; the output of a call to CBPS() or CBMSM() from the **CBPS** package.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well. In older versions of **cobalt**, setting int = TRUE displayed squares of covariates; to replicate this behavior, set int = 2.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	Optional; either a vector or data.frame containing distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in data. Note that the propensity scores generated by CBPS() and CBMSM() are automatically included.
addl	An optional data frame or the quoted names of additional covariates for which to present balance. These may be covariates included in the original dataset but not included in the call to CBPS() or CBMSM(). If variable names are specified, bal.tab() will look first in the argument to data, if specified, and next in the input object. For CBMSM objects, must be a list of additional covariate values as described above, with one list entry per time period. Each data set must have one row per individual, unlike the data frame in the original call to CBMSM().
data	an optional data frame containing variables that might be named in arguments to addl and cluster.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options.
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options.
s.d.denom	whether the denominator for standardized differences (if any are calculated) should be the standard deviation of the treated group ("treated"), the standard deviation of the control group ("control"), or the pooled standard deviation ("pooled"), computed as the square root of the mean of the group variances. Abbreviations allowed. If not specified, bal.tab() will use "treated" if the estimand of the call to CBPS() (or that specified by the user to estimand) is the ATT and "pooled" if the estimand is the ATE.
m.threshold	a numeric value for the threshold for mean differences. .1 is recommended.
v.threshold	a numeric value for the threshold for variance ratios. Will automatically convert to the inverse if less than 1.

ks.threshold	a numeric value for the threshold for Kolmogorov-Smirnov statistics. Must be between 0 and 1.
r.threshold	a numeric value for the threshold for correlations between covariates and treatment when treatment is continuous.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the CBPS object. See bal.tab.cluster for details.
pairwise	Whether balance should be computed for pairs of treatments or for each treatment against all others combined. See bal.tab.multi for details.
focal	The name of the focal treatment when multiple categorical treatments are used. See bal.tab.multi for details.
s.weights	Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data or the CBPS object. If the original call to CBPS() included sampling weights, they should be specified again here to ensure correct computation of balance statistics and unadjusted values. See Details below.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	A logical vector denoting whether each observation should be included. It should be the same length as the variables in the original call to CBPS() or CBMSM(). NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Leave FALSE if computed values not displayed will be used later.
...	Further arguments to control display of output. See display options for details.

Details

bal.tab.CBPS generates a list of balance summaries for the CBPS or CBMSM object given and functions similarly to balance() in **CBPS**.

All balance statistics are calculated whether they are displayed by print or not, unless quick = TRUE. The threshold values (m.threshold, v.threshold, ks.threshold, and r.threshold) control whether extra columns should be inserted into the Balance table describing whether the balance statistics in question exceeded or were within the threshold. Including these thresholds also creates summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure.

The CBPS object does not return sampling weights even if they are used; rather, the weights returned already have the sampling weights combined within them. Because some of the checks and defaults in bal.tab() rely on patterns in these weights, using sampling weights in CBPS() without specifying them in bal.tab() can lead to incorrect results. If sampling weights are used in CBPS(), it is important that they are specified in bal.tab() as well.

Value

For point treatments, if clusters are not specified, an object of class "bal.tab" containing balance summaries for the CBPS object. See bal.tab for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See bal.tab.cluster for details.

If treatment is continuous, means, mean differences, and variance ratios are replaced by (weighted) Pearson correlations between each covariate and treatment. The r.threshold argument works the same as m.threshold, v.threshold, or ks.threshold, adding an extra column to the balance table output and creating additional summaries for balance tallies and maximum imbalances. All arguments related to the calculation or display of mean differences or variance ratios are ignored. The int, addl, un, distance, and cluster arguments are still used as described above.

If multiple categorical treatments are used, an object of class "bal.tab.multi" containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See bal.tab.multi for details.

If CBPS() is used with multiple categorical treatments, an object of class "bal.tab.multi" containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See bal.tab.multi for details.

If CBMSM() is used for longitudinal treatments, an object of class "bal.tab.msm" containing balance summaries for each time period and a summary of balance across time periods. See bal.tab.msm for details.

Author(s)

Noah Greifer

See Also

bal.tab for details of calculations. bal.tab.cluster for more information on clustered data. bal.tab.multi for more information on multiple categorical treatments. bal.tab.msm for more information on longitudinal treatments.

Examples

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

## Using CBPS() for generating covariate balancing
## propensity score weights
cbps.out <- CBPS(treat ~ age + educ + married + race +
             nodegree + re74 + re75, data = lalonde)
bal.tab(cbps.out)
```

---

bal.tab.default *Balance Statistics for Other Objects*

Description

Generates balance statistics using an object for which there is not a defined method.

Usage

```
## Default S3 method:
bal.tab(obj, ...)
```

Arguments

obj	An object containing information about conditioning. See Details.
...	Arguments that would be passed to bal.tab.formula, bal.tab.data.frame, or bal.tab.time.list. See Details.

Details

bal.tab.default() processes its input and attempt to extract enough information from it to display covariate balance for obj. The goal of this method was to allow users who have created their own objects containing conditioning information (i.e., weights, subclasses, treatments, covariates, etc.) to access the capabilities of bal.tab() without having a special method written for them. By including the correct items in obj, bal.tab.default can present balance tables as if the input was the output of one of the specifically supported packages (e.g., **MatchIt**, **twang**, etc.).

The function will search obj for the following named items and attempt to process them:

treat A vector (numeric, character, factor) containing the values of the treatment for each unit or the name of the column in data containing them. Essentially the same input to treat in bal.tab.data.frame.

treat.list A list of vectors (numeric, character, factor) containing, for each time point, the values of the treatment for each unit or the name of the column in data containing them. Essentially the same input to treat.list in bal.tab.time.list.

covs A data.frame containing the values of the covariates for each unit. Essentially the same input to covs in bal.tab.data.frame.

covs.list A list of data.frames containing, for each time point, the values of the covariates for each unit. Essentially the same input to covs.list in bal.tab.time.list.

formula A formula with the treatment variable as the response and the covariates for which balance is to be assessed as the terms. Essentially the same input to formula in bal.tab.formula.

formula.list A list of formulas with, for each time point, the treatment variable as the response and the covariates for which balance is to be assessed as the terms. Essentially the same input to formula.list in bal.tab.time.list.

data A data.frame containing variables with the names used in other arguments and components (e.g., formula, weights, etc.). Essentially the same input to data in bal.tab.formula, bal.tab.data.frame, or bal.tab.time.list.

weights A vector, list, or data.frame containing weights for each unit or a string containing the names of the weights variables in data. Essentially the same input to weights in bal.tab.data.frame or bal.tab.time.list.

distance A vector or data.frame containing distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in data. Essentially the same input to distance in bal.tab.data.frame.

`formula.list` A list of vectors or `data.frames` containing, for each time point, distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in `data`. Essentially the same input to `distance.list` in `bal.tab.time.list`.

`subclass` A vector containing subclass membership for each unit or a string containing the name of the subclass variable in `data`. Essentially the same input to `subclass` in `bal.tab.data.frame`.

`match.strata` A vector containing matching stratum membership for each unit or a string containing the name of the matching stratum variable in `data`. Essentially the same input to `match.strata` in `bal.tab.data.frame`.

`estimand` A `character` vector; whether the desired estimand is the "ATT", "ATC", or "ATE" for each set of weights. Essentially the same input to `estimand` in `bal.tab.data.frame`.

`s.weights` A vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in `data`. Essentially the same input to `s.weights` in `bal.tab.data.frame` or `bal.tab.time.list`.

`focal` The name of the focal treatment when multiple categorical treatments are used. Essentially the same input to `focal` in `bal.tab.data.frame`.

`call` A `call` object containing the function call, usually generated by using `match.call` inside the function that created `obj`.

Any of these items can also be supplied directly to `bal.tab.default`, e.g., `bal.tab.default( obj, formula = treat ~ x`. If supplied, it will override the object with the same role in `obj`. In addition, any arguments to `bal.tab.formula`, `bal.tab.data.frame`, and `bal.tab.time.list` are allowed and perform the same function.

At least some inputs containing information to create the treatment and covariates are required (e.g., `formula` and `data` or `covs` and `treat`). All other arguments are optional and have the same defaults as those in `bal.tab.data.frame` or `bal.tab.time.list`. If `treat.list`, `covs.list`, or `formula.list` are supplied in `obj` or as an argument to `bal.tab.default`, the function will proceed considering a longitudinal treatment. Otherwise, it will proceed considering a point treatment.

`bal.tab.default`, like other `bal.tab` methods, is just a shortcut to supply arguments to `bal.tab.data.frame` or `bal.tab.time.list`. Therefore, any matters regarding argument priority or function are described in the documentation for these methods.

Value

For point treatments, if clusters and imputations are not specified, an object of class `"bal.tab"` containing balance summaries for the specified treatment and covariates. See `bal.tab` for details.

If clusters are specified, an object of class `"bal.tab.cluster"` containing balance summaries within each cluster and a summary of balance across clusters. See `bal.tab.cluster` for details.

If imputations are specified, an object of class `"bal.tab.imp"` containing balance summaries for each imputation and a summary of balance across imputations, just as with clusters. See `bal.tab.imp` for details.

If both clusters and imputations are specified, an object of class `"bal.tab.imp.cluster"` containing summaries between and across all clusters and imputations.

If treatment is continuous, then means, mean differences, and variance ratios are replaced by (weighted) Pearson correlations between each covariate and treatment. The `r.threshold` argument works the same as `m.threshold`, `v.threshold`, or `ks.threshold`, adding an extra column to

the balance table output and creating additional summaries for balance tallies and maximum imbalances. All arguments related to the calculation or display of mean differences or variance ratios are ignored. The `int`, `distance`, `addl`, `un`, cluster and imputation arguments are still used as described above.

If multiple categorical treatments are used, an object of class `"bal.tab.multi"` containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See `bal.tab.multi` for details.

If longitudinal treatments are used, an object of class `bal.tab.msm` containing balance summaries at each time point. Each balance summary is its own `bal.tab` object. See `bal.tab.msm` for more details. Currently, clusters and multiply imputed data are not compatible with longitudinal treatments.

Author(s)

Noah Greifer

See Also

`bal.tab.data.frame` and `link{bal.tab.time.list}` for additional arguments to be supplied. `bal.tab` for output and details of calculations. `bal.tab.cluster` for more information on clustered data. `bal.tab.imp` for more information on multiply imputed data. `bal.tab.multi` for more information on multiple categorical treatments.

Examples

```
data("lalonde", package = "cobalt")
covs <- subset(lalonde,  select = -c(treat, re78))

##Writing a function the produces output for direct
##use in bal.tab.default

ate.weights <- function(treat, covs) {
    data <- data.frame(treat, covs)
    formula <- formula(data)
    ps <- glm(formula, data = data,
              family = "binomial")$fitted.values
    weights <- treat/ps + (1-treat)/(1-ps)
    call <- match.call()
    out <- list(treat = treat,
                covs = covs,
                distance = ps,
                weights = weights,
                estimand = "ATE",
                call = call)
    return(out)
}

out <- ate.weights(lalonde$treat, covs)

bal.tab(out, un = TRUE)
```

`bal.tab.df.formula` *Balance Statistics for Data Sets*

Description

Generates balance statistics for unadjusted, matched, weighted, or stratified data using either a `data.frame` or formula interface. Note that several arguments that used to be documented here are now documented in display options. They are still available.

Usage

```
## S3 method for class 'data.frame'
bal.tab(covs,
    treat,
    data = NULL,
    weights = NULL,
    distance = NULL,
    subclass = NULL,
    match.strata = NULL,
    method,
    int = FALSE,
    poly = 1,
    addl = NULL,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
    v.threshold = NULL,
    ks.threshold = NULL,
    r.threshold = NULL,
    cluster = NULL,
    imp = NULL,
    pairwise = TRUE,
    focal = NULL,
    s.weights = NULL,
    estimand = NULL,
    abs = FALSE,
    subset = NULL,
    quick = FALSE,
    ...)

## S3 method for class 'formula'
bal.tab(formula,
    data = NULL,
    ...)
```

Arguments

covs	A data.frame containing covariate values for each unit.
treat	Either a vector containing treatment status values for each unit or a string containing the name of the treatment variable in data.
formula	a formula with the treatment variable as the response and the covariates for which balance is to be assessed as the terms. All terms must be present as variable names in data or the global environment.
data	Optional; a data.frame containing variables with the names used in formula, treat, weights, distance, addl, subclass, match.strata, cluster, and/or imp if any.
weights	Optional; a vector, list, or data.frame containing weights for each unit or a string containing the names of the weights variables in data. These can be weights generated by, e.g., inverse probability weighting or matching weights resulting from a matching algorithm. This must be specified in method. If weights=NULL, subclass=NULL and match.strata=NULL, balance information will be presented only for the unadjusted sample.
distance	Optional; either a vector or data.frame containing distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in data.
subclass	Optional; either a vector containing subclass membership for each unit or a string containing the name of the subclass variable in data. If weights=NULL, subclass=NULL and match.strata=NULL, balance information will be presented only for the unadjusted sample.
match.strata	Optional; either a vector containing matching stratum membership for each unit or a string containing the name of the matching stratum variable in data. If weights=NULL, subclass=NULL and match.strata=NULL, balance information will be presented only for the unadjusted sample.
method	A character vector containing the method of adjustment, if any. If weights are specified, the user must specify either "matching" or "weighting"; "weighting" is the default. If multiple sets of weights are used, each must have a corresponding value for method, but if they are all of the same type, only one value is required. If subclass is specified, "subclassification" is the default. Abbreviations allowed.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well. In older versions of **cobalt**, setting int = TRUE displayed squares of covariates; to replicate this behavior, set int = 2.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
addl	A vector, data.frame, or the quoted names of additional covariates for which to present balance. These may be covariates included in the original dataset but not included in covs. In general, it makes more sense to include all desired

variables in covs than in addl. If the argument is a vector, the variable will be displayed as "addl" in the output.

continuous — whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options.

binary — whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options.

s.d.denom — A character vector denoting whether the denominator for standardized differences (if any are calculated) should be the standard deviation of the treated group ("treated"), the standard deviation of the control group ("control"), or the pooled standard deviation ("pooled"), computed as the square root of the mean of the group variances. Abbreviations allowed. If weights are supplied, each set of weights should have a corresponding entry to s.d.denom. If left blank and weights are supplied, bal.tab() will try to determine whether the ATT, ATC, or ATE is being estimated based on the pattern of weights and supply s.d.denom accordingly. If matching or subclassification are used, the default is "treated"; if weighting is used, the default is "pooled". If left blank, bal.tab() will try to use the estimand argument.

m.threshold — A numeric value for the threshold for mean differences. .1 is recommended.

v.threshold — A numeric value for the threshold for variance ratios. Will automatically convert to the inverse if less than 1.

ks.threshold — a numeric value for the threshold for Kolmogorov-Smirnov statistics. Must be between 0 and 1.

r.threshold — A numeric value for the threshold for correlations between covariates and treatment when treatment is continuous.

cluster — either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the CBPS object. See bal.tab.cluster for details.

imp — either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in data. See bal.tab.imp for details.

pairwise — Whether balance should be computed for pairs of treatments or for each treatment against all others combined. See bal.tab.multi for details.

focal — The name of the focal treatment when multiple categorical treatments are used. See bal.tab.multi for details.

s.weights — Optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.

estimand — character; whether the desired estimand is the "ATT", "ATC", or "ATE" for each set of weights. This argument can be used in place of s.d.denom to specify how standardized differences are calculated.

abs — logical; whether displayed balance statistics should be in absolute value or not.

subset	A logical vector denoting whether each observation should be included. It should be the same length as the treatment and covariates. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Leave FALSE if computed values not displayed will be used later.
...	For bal.tab.formula, other arguments to be passed to bal.tab.data.frame. Otherwise, further arguments to control display of output. See display options for details.

Details

bal.tab.data.frame() generates a list of balance summaries for the covariates and treatment status values given. bal.tab.formula() does the same but uses a formula interface instead. When the formula interface is used, the formula and data are reshaped into a treatment vector and data.frame of covariates and then simply passed through the data.frame method.

The argument to match.strata corresponds to a factor vector containing the name or index of each pair/stratum for units conditioned through matching, for example, using the **optmatch** package. If more than one of weights, subclass, or match.strata are specified, bal.tab() will attempt to figure out which one to apply. Currently only one of these can be applied ta a time. bal.tab() behaves differently depending on whether subclasses are used in conditioning or not. If they are used, bal.tab() creates balance statistics for each subclass and for the sample in aggregate. See bal.tab.subclass for more information.

All balance statistics are calculated whether they are displayed by print or not, unless quick = TRUE. The threshold values (m.threshold, v.threshold, ks.threshold, and, for continuous treatments, r.threshold) control whether extra columns should be inserted into the Balance table describing whether the balance statistics in question exceeded or were within the threshold. Including these thresholds also creates summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure. When subclassification is used, the extra threshold columns are placed within the balance tables for each subclass as well as in the aggregate balance table, and the summary tables display balance for each subclass.

The inputs (if any) to covs and data must be a data.frame; if more than one variable is included, this is straightforward (i.e., because data[,c("v1", "v2")] is already a data.frame), but if only one variable is used with the matrix subsetting syntax (e.g., data[,"v1"]), R will coerce it to a vector, thus making it unfit for input. To avoid this, make sure to use the list subsetting syntax (e.g., data["v1"]) if only one variable is to be added (this can also be used for multiple variables and is good practice in general). Again, when more than one variable is included, the input is generally already a data.frame and nothing needs to be done.

Multiple sets of weights can be supplied simultaneously by entering a data.frame or a character vector containing the names of weight variables found in data or a list of weights vectors or names. The arguments to method, s.d.denom, and estimand, if any, must be either the same length as the number of sets of weights or of length one, where the sole entry is applied to all sets. When standardized differences are computed for the unadjusted group, they are done using the first entry to s.d.denom or estimand. When only one set of weights is supplied, the output for the adjusted group will simply be called "Adj", but otherwise will be named after each corresponding

set of weights. Specifying multiple sets of weights will also add components to other output of `bal.tab()`.

Clusters and imputations can be used at the same time, but the resulting output may be quite large. Setting `which.cluster` or `which.imp` to `NA` can help keep the output clean.

Value

For point treatments, if clusters and imputations are not specified, an object of class `"bal.tab"` containing balance summaries for the specified treatment and covariates. See `bal.tab` for details.

If clusters are specified, an object of class `"bal.tab.cluster"` containing balance summaries within each cluster and a summary of balance across clusters. See `bal.tab.cluster` for details.

If imputations are specified, an object of class `"bal.tab.imp"` containing balance summaries for each imputation and a summary of balance across imputations, just as with clusters. See `bal.tab.imp` for details.

If both clusters and imputations are specified, an object of class `"bal.tab.imp.cluster"` containing summaries between and across all clusters and imputations.

If treatment is continuous, then means, mean differences, and variance ratios are replaced by (weighted) Pearson correlations between each covariate and treatment. The `r.threshold` argument works the same as `m.threshold`, `v.threshold`, or `ks.threshold`, adding an extra column to the balance table output and creating additional summaries for balance tallies and maximum imbalances. All arguments related to the calculation or display of mean differences or variance ratios are ignored. The `int`, `distance`, `addl`, `un`, cluster and imputation arguments are still used as described above.

If multiple categorical treatments are used, an object of class `"bal.tab.multi"` containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See `bal.tab.multi` for details.

Author(s)

Noah Greifer

See Also

`bal.tab` for output and details of calculations. `bal.tab.cluster` for more information on clustered data. `bal.tab.imp` for more information on multiply imputed data. `bal.tab.multi` for more information on multiple categorical treatments.

Examples

```
data("lalonde", package = "cobalt")
lalonde$p.score <- glm(treat ~ age + educ + race, data = lalonde,
            family = "binomial")$fitted.values
covariates <- subset(lalonde,
                     select = c(age, educ, race))

## Propensity score weighting using IPTW
lalonde$iptw.weights <- ifelse(lalonde$treat==1,
                               1/lalonde$p.score,
```

```
                                1/(1-lalonde$p.score))

# data frame interface:
bal.tab(covariates, treat = "treat", data = lalonde,
      weights = "iptw.weights", method = "weighting",
      s.d.denom = "pooled")

# Formula interface:
bal.tab(treat ~ age + educ + race, data = lalonde,
      weights = "iptw.weights", method = "weighting",
      s.d.denom = "pooled")

## Propensity score subclassification
lalonde$subclass <- findInterval(lalonde$p.score,
                        quantile(lalonde$p.score,
                        (0:6)/6), all.inside = TRUE)

# data frame interface:
bal.tab(covariates, treat = "treat", data = lalonde,
      subclass = "subclass", method = "subclassification",
      disp.subclass = TRUE, s.d.denom = "pooled")

# Formula interface:
bal.tab(treat ~ age + educ + race, data = lalonde,
      subclass = "subclass", method = "subclassification",
      disp.subclass = TRUE, s.d.denom = "pooled")
```

`bal.tab.df.formula.list`

Balance Statistics for Longitudinal Treatments

Description

Generates balance statistics for data coming from a longitudinal treatment scenario. The primary input is in the form of a list of formulas or data.frames contain the covariates at each time point. `bal.tab` automatically classifies this list as either a `data.frame.list` or `formula.list`, respectively. Note that several arguments that used to be documented here are now documented in display options. They are still available.

Usage

```
## S3 method for class 'data.frame.list'
bal.tab(covs.list,
    treat.list = NULL,
    data = NULL,
    weights = NULL,
    int = FALSE,
    poly = 1,
    distance.list = NULL,
```

```
    addl.list = NULL,
    method,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
    v.threshold = NULL,
    ks.threshold = NULL,
    r.threshold = NULL,
    pairwise = TRUE,
    s.weights = NULL,
    estimand = "ATE",
    abs = FALSE,
    subset = NULL,
    quick = FALSE,
    ...)

## S3 method for class 'formula.list'
bal.tab(formula.list,
    data = NULL,
    ...)
```

Arguments

covs.list	a list of data frames containing all the covariates to be assessed at each time point. Covariates to be assessed at multiple points must be included in the data frames for each time point. Data must be in the "wide" format, with one row per unit.
treat.list	treatment status for each unit at each time point. This can be specified as a list or data frame of vectors, each of which contains the treatment status of each individual at each time point, or a list or vector of the names of variables in data that contain treatment at each time point.
formula.list	a list of formulas with the treatment for each time period on the left and the covariates for which balance is to be displayed on the right. An argument to data is required unless all objects in the formulas exist in the environment.
data	for bal.tab.data.frame.list: optional; a data frame containing variables with the names used in treat.list, weights, distance.list, and/or addl.list, if any. For bal.tab.formula.list: required; a data frame containing all covariates named in the formulas and variables with the names used in the arguments mentioned above. If all objects in the formula.list formulas are present in the environment, can be omitted. data must be in the "wide" format, with one row per unit.
weights	optional; a vector, list, or data frame containing weights for each unit or a string containing the names of the weights variables in data. These can be weights generated by, e.g., inverse probability weighting. If weights=NULL, balance information will be presented only for the unadjusted sample.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well. In

older versions of **cobalt**, setting `int = TRUE` displayed squares of covariates; to replicate this behavior, set `int = 2`.

`poly` `numeric`; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If `int` is numeric, `poly` will take on the value of `int`.

`distance.list` optional; distance values (e.g., propensity scores) for each unit. These can be specified as a list of vectors or data frames containing the distance values (one for each time point), or as a single vector or data frame to be applied at all times points. The vectors or data frames can be replaced with the names of variables in `data` containing the distance values. If a list is used and some time points are not to have distance values, these can be replaced with NULL in the list.

`addl.list` optional; additional covariates for which to present balance. These may be covariates included in the original dataset but not included in `time.list`. In general, it makes more sense to include all desired variables in `time.list` than in `addl.list`. The arguments can be entered the same ways as those to `distance.list`.

`method` a character vector containing the method of adjustment, if any. Currently only "weighting" is supported.

`continuous` whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using `set.cobalt.options`.

`binary` whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using `set.cobalt.options`.

`s.d.denom` a character vector denoting whether the denominator for standardized differences (if any are calculated) should be the standard deviation of the treated group ("treated"), the standard deviation of the control group ("control"), or the pooled standard deviation ("pooled"), computed as the square root of the mean of the group variances. Abbreviations allowed. If weights are supplied, each set of weights should have a corresponding entry to `s.d.denom`. If left blank and weights are supplied, `bal.tab()` will try to determine whether the ATT, ATC, or ATE is being estimated based on the pattern of weights and supply `s.d.denom` accordingly. If left blank, `bal.tab()` will try to use the `estimand` argument. It is recommended not to set this argument for longitudinal treatments.

`m.threshold` a numeric value for the threshold for mean differences. .1 is recommended.

`v.threshold` a numeric value for the threshold for variance ratios. Will automatically convert to the inverse if less than 1.

`ks.threshold` a numeric value for the threshold for Kolmogorov-Smirnov statistics. Must be between 0 and 1.

`r.threshold` a numeric value for the threshold for correlations between covariates and treatment when treatment is continuous.

`pairwise` whether balance should be computed for pairs of treatments or for each treatment against all others combined when treatment is multinomial. See `bal.tab.multi` for details.

s.weights	optional; either a vector containing sampling weights for each unit or a string containing the name of the sampling weight variable in data. These function like regular weights except that both the adjusted and unadjusted samples will be weighted according to these weights if weights are used.
estimand	the causal estimand of interest. This value is used to set s.d.denom, and should not be changed from "ATE".
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical vector denoting whether each observation should be included. It should be the same length as the treatment and covariates. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Leave FALSE if computed values not displayed will be used later.
...	For bal.tab.formula.list, other arguments to be passed to bal.tab.data.frame.list. Otherwise, further arguments to control display of output. See display options for details.

Details

bal.tab.formula.list() and bal.tab.data.frame.list() generate a list of balance summaries for each time point based on the treatments and covariates provided. All data must be in the "wide" format, with exactly one row per unit and columns representing variables at different time points. See the weightitMSM documentation for an example of how to transform long data into wide data using reshape.

All balance statistics are calculated whether they are displayed by print or not, unless quick = TRUE. The threshold values (m.threshold, v.threshold, ks.threshold, and r.threshold) control whether extra columns should be inserted into the Balance table describing whether the balance statistics in question exceeded or were within the threshold. Including these thresholds also creates summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure. When subclassification is used, the extra threshold columns are placed within the balance tables for each subclass as well as in the aggregate balance table, and the summary tables display balance for each subclass.

Multiple sets of weights can be supplied simultaneously by including entering a data frame or a character vector containing the names of weight variables found in data or a list thereof. The arguments to method, s.d.denom, and estimand, if any, must be either the same length as the number of sets of weights or of length one, where the sole entry is applied to all sets. When standardized differences are computed for the unadjusted group, they are done using the first entry to s.d.denom or estimand. When only one set of weights is supplied, the output for the adjusted group will simply be called "Adj", but otherwise will be named after each corresponding set of weights. Specifying multiple sets of weights will also add components to other output of bal.tab().

Value

An object of class bal.tab.msm containing balance summaries at each time point. Each balance summary is its own bal.tab object. See bal.tab.msm for more details.

Currently, clusters and multiply imputed data are not compatible with longitudinal treatments.

See bal.tab base methods for more detailed information on the value of the bal.tab objects produced for each time point.

Author(s)

Noah Greifer

See Also

bal.tab base methods for details of calculations.

bal.tab.msm for output and related options.

Examples

```
data("iptwExWide", package = "twang")
library("cobalt")

## Estimating longitudinal propensity scores and weights
ps1 <- glm(tx1 ~ age + gender + use0,
           data = iptwExWide,
           family = "binomial")$fitted.values
w1 <- ifelse(iptwExWide$tx1 == 1, 1/ps1, 1/(1-ps1))
ps2 <- glm(tx2 ~ age + gender + use0 + tx1 + use1,
           data = iptwExWide,
           family = "binomial")$fitted.values
w2 <- ifelse(iptwExWide$tx2 == 1, 1/ps2, 1/(1-ps2))
ps3 <- glm(tx3 ~ age + gender + use0 + tx1 + use1 + tx2 + use2,
           data = iptwExWide,
           family = "binomial")$fitted.values
w3 <- ifelse(iptwExWide$tx3 == 1, 1/ps3, 1/(1-ps3))

w <- w1*w2*w3

# data frame interface:
bal.tab(list(iptwExWide[c("use0", "gender")],
             iptwExWide[c("use0", "gender", "use1", "tx1")],
             iptwExWide[c("use0", "gender", "use1", "tx1", "use2", "tx2")]),
        treat.list = iptwExWide[c("tx1", "tx2", "tx3")],
        weights = w,
        distance.list = list(ps1, ps2, ps3),
        addl.list = iptwExWide["age"],
        un = TRUE)

# Formula interface:
bal.tab(list(tx1 ~ use0 + gender,
             tx2 ~ use0 + gender + use1 + tx1,
             tx3 ~ use0 + gender + use1 + tx1 + use2 + tx2),
        data = iptwExWide,
        weights = w,
        distance.list = list(ps1, ps2, ps3),
        addl.list = "age",
        un = TRUE)
```

bal.tab.Match *Balance Statistics for* Matching, optmatch, ebal, *and* designmatch *Objects*

Description

Generates balance statistics for output objects from **Matching**, **optmatch**, **ebal**, and **designmatch**. Note that several arguments that used to be documented here are now documented in display options. They are still available.

Usage

```
## S3 method for class 'Match'
bal.tab(M,
    formula = NULL,
    data = NULL,
    treat = NULL,
    covs = NULL,
    int = FALSE,
    poly = 1,
    distance = NULL,
    addl = NULL,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
    v.threshold = NULL,
    ks.threshold = NULL,
    cluster = NULL,
    abs = FALSE,
    subset = NULL,
    quick = FALSE,
    ...)

## S3 method for class 'optmatch'
bal.tab(optmatch, ...)

## S3 method for class 'ebalance'
bal.tab(ebal, ...)

## S3 method for class 'designmatch'
bal.tab(dm, ...)
```

Arguments

M a Match object; the output of a call to Match() from the **Matching** package.

`optmatch` an `optmatch` object; the output of a call to `pairmatch()` or `fullmatch()` from the **optmatch** package. This should be a factor vector containing the matching stratum membership for each unit.

`ebal` an `ebalance` object; the output of a call to `ebalance()` or `ebalance.trim()` from the **ebal** package.

`dm` the output of a call to `bmatch()`, `nmatch()`, or related wrapper functions from the **designmatch** package. This should be a list containing the IDs of the matched treated and control units.

`formula` a `formula` with the treatment variable as the response and the covariates for which balance is to be assessed as the predictors. All named variables must be in `data`. See Details.

`data` a data frame containing all the variables named in `formula`. See Details.

`treat` a vector of treatment statuses. See Details.

`covs` a data frame of covariate values for which to check balance. See Details.

`int` `logical` or `numeric`; whether or not to include 2-way interactions of covariates included in `covs` and in `addl`. If `numeric`, will be passed to `poly` as well. In older versions of **cobalt**, setting `int = TRUE` displayed squares of covariates; to replicate this behavior, set `int = 2`.

`poly` `numeric`; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If `int` is numeric, `poly` will take on the value of `int`.

`distance` optional; either a vector or data.frame containing distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in `data`.

`addl` a data frame of additional covariates for which to present balance. These may be covariates included in the original dataset but not included in `formula` or `covs`. In general, it makes more sense to include all desired variables in `formula` or `covs` than in `addl`. See note in Details for using `addl`.

`continuous` whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using `set.cobalt.options`.

`binary` whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using `set.cobalt.options`.

`s.d.denom` whether the denominator for standardized differences (if any are calculated) should be the standard deviation of the treated group ("treated"), the standard deviation of the control group ("control"), or the pooled standard deviation ("pooled"), computed as the square root of the mean of the group variances. Abbreviations allowed. If not specified, for `Match` objects, `bal.tab()` will use "treated" if the estimand of the call to `Match()` is the ATT, "pooled" if the estimand is the ATE, and "control" if the estimand is the ATC; for `optmatch`, `ebal`, and `designmatch` objects, `bal.tab()` will use "treated".

m.threshold	a numeric value for the threshold for mean differences. .1 is recommended.
v.threshold	a numeric value for the threshold for variance ratios. Will automatically convert to the inverse if less than 1.
ks.threshold	a numeric value for the threshold for Kolmogorov-Smirnov statistics. Must be between 0 and 1.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the CBPS object. See bal.tab.cluster for details.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical vector denoting whether each observation should be included. It should be the same length as the variables in the original call to the conditioning function. NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Leave FALSE if computed values not displayed will be used later.
...	for bal.tab.optmatch(), bal.tab.ebalance(), and bal.tab.designmatch(), the same arguments as those passed to bal.tab.Match(). Otherwise, further arguments to control display of output. See display options for details.

Details

bal.tab() generates a list of balance summaries for the object given, and function similarly to MatchBalance() in **Matching** and meantab() in **designmatch**. Note that output objects from **designmatch** do not have their own class; bal.tab() first checks whether the object meets the criteria to be treated as a designmatch object before dispatching the correct method. In particular, renaming or removing items from the output object can create unintended consequences.

The input to bal.tab.Match(), bal.tab.optmatch(), bal.tab.ebalance(), and bal.tab.designmatch() must include either both formula and data or both covs and treat. Using the formula + data inputs mirrors how MatchBalance() is used in **Matching**, and using the covs + treat input mirrors how meantab() is used in **designmatch**. (Note that to see identical results to meantb(), s.d.denom must be set to "pooled", though this is not recommended.)

All balance statistics are calculated whether they are displayed by print or not, unless quick = TRUE. The threshold values (m.threshold, v.threshold, and ks.threshold) control whether extra columns should be inserted into the Balance table describing whether the balance statistics in question exceeded or were within the threshold. Including these thresholds also creates summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure.

The inputs (if any) to covs must be a data frame; if more than one variable is included, this is straightforward (i.e., because data[,c("v1", "v2")] is already a data frame), but if only one variable is used (e.g., data[,"v1"]), R will coerce it to a vector, thus making it unfit for input. To avoid this, simply wrap the input to covs in data.frame() or use subset() if only one variable is to be added. Again, when more than one variable is included, the input is general already a data frame and nothing needs to be done.

Value

For point treatments, if clusters and imputations are not specified, an object of class "bal.tab" containing balance summaries for the given object. See bal.tab for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See bal.tab.cluster for details.

Author(s)

Noah Greifer

See Also

bal.tab for details of calculations.

Examples

```
########## Matching ##########
library(Matching); data("lalonde", package = "cobalt")

p.score <- glm(treat ~ age + educ + race +
            married + nodegree + re74 + re75,
            data = lalonde, family = "binomial")$fitted.values
Match.out <- Match(Tr = lalonde$treat, X = p.score)

## Using formula and data
bal.tab(Match.out, treat ~ age + educ + race +
        married + nodegree + re74 + re75, data = lalonde)

########## optmatch ##########
library("optmatch"); data("lalonde", package = "cobalt")

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

## Using formula and data
bal.tab(pm, treat ~ age + educ + race +
        married + nodegree + re74 + re75, data = lalonde,
        distance = "prop.score")

########## ebal ##########
library("ebal"); data("lalonde", package = "cobalt")

covariates <- subset(lalonde, select = -c(re78, treat, race))
e.out <- ebalance(lalonde$treat, covariates)

## Using treat and covs
bal.tab(e.out, treat = lalonde$treat, covs = covariates)

## Not run:
```

```
########## designmatch ##########
library("designmatch"); data("lalonde", package = "cobalt")

covariates <- as.matrix(lalonde[c("age", "educ", "re74", "re75")])
dmout <- bmatch(lalonde$treat,
                total_groups = sum(lalonde$treat == 1),
                mom = list(covs = covariates,
                           tols = absstddif(covs, treat, .05))
                )

## Using treat and covs
bal.tab(dmout, treat = lalonde$treat, covs = covariates)

## End(Not run)
```

`bal.tab.matchit` *Balance Statistics for MatchIt Objects*

Description

Generates balance statistics for `matchit` objects from **MatchIt**. Note that several arguments that used to be documented here are now documented in display options. They are still available.

Usage

```
## S3 method for class 'matchit'
bal.tab(m,
    int = FALSE,
    poly = 1,
    distance = NULL,
    addl = NULL,
    data = NULL,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
    v.threshold = NULL,
    ks.threshold = NULL,
    cluster = NULL,
    abs = FALSE,
    subset = NULL,
    quick = FALSE,
    ...)
```

Arguments

`m`	a `matchit` object; the output of a call to `matchit()` from the **MatchIt** package.

int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well. In older versions of **cobalt**, setting int = TRUE displayed squares of covariates; to replicate this behavior, set int = 2.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	Optional; either a vector or data.frame containing distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in data. Note that the distance measure generated by matchit() is automatically included.
addl	an optional data frame or the quoted names of additional covariates for which to present balance. These may be covariates included in the original dataset but not included in the call to matchit(). If variable names are specified, bal.tab() will look first in the argument to data, if specified, and next in the matchit object.
data	an optional data frame containing variables that might be named in arguments to distance, addl, and cluster.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options.
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options.
s.d.denom	whether the denominator for standardized differences (if any are calculated) should be the standard deviation of the treated group ("treated"), the standard deviation of the control group ("control"), or the pooled standard deviation ("pooled"), computed as the square root of the mean of the group variances. Abbreviations allowed. The default is "treated".
m.threshold	a numeric value for the threshold for mean differences. .1 is recommended.
v.threshold	a numeric value for the threshold for variance ratios. Will automatically convert to the inverse if less than 1.
ks.threshold	a numeric value for the threshold for Kolmogorov-Smirnov statistics. Must be between 0 and 1.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the CBPS object. See bal.tab.cluster for details.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical vector denoting whether each observation should be included. It should be the same length as the variables in the original call to MatchIt(). NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.

quick	logical; if TRUE, will not compute any values that will not be displayed. Leave FALSE if computed values not displayed will be used later.
...	further arguments to control display of output. See display options for details.

Details

bal.tab.matchit() generates a list of balance summaries for the matchit object given, and functions similarly to summary.matchit() in **MatchIt**. bal.tab() behaves differently depending on whether subclasses are used in conditioning or not. If they are used, bal.tab() creates balance statistics for each subclass and for the sample in aggregate; see bal.tab.subclass for more information.

All balance statistics are calculated whether they are displayed by print or not, unless quick = TRUE. The threshold values (m.threshold, v.threshold, and ks.threshold) control whether extra columns should be inserted into the Balance table describing whether the balance statistics in question exceeded or were within the threshold. Including these thresholds also creates summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure. When subclassification is used, the extra threshold columns are placed within the balance tables for each subclass as well as in the aggregate balance table, and the summary tables display balance for each subclass.

Value

If subclassification is used, an object of class "bal.tab.subclass" containing balance summaries within and across subclasses. See bal.tab.subclass for details.

If matching is used and clusters are not specified, an object of class "bal.tab" containing balance summaries for the matchit object. See bal.tab for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See bal.tab.cluster for details.

Author(s)

Noah Greifer

See Also

bal.tab for details of calculations.

Examples

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

## Nearest Neighbor matching
m.out1 <- matchit(treat ~ age + educ + race +
                  married + nodegree + re74 + re75,
                  data = lalonde, method = "nearest")
bal.tab(m.out1, un = TRUE, m.threshold = .1,
        v.threshold = 2)

## Subclassification
```

```
m.out2 <- matchit(treat ~ age + educ + race +
                  married + nodegree + re74 + re75,
                  data = lalonde, method = "subclass")
bal.tab(m.out2, disp.subclass = TRUE)
```

`bal.tab.ps` *Balance Statistics for twang Objects*

Description

Generates balance statistics for `ps`, `mnps`, and `iptw` objects from **twang** and for `ps.cont` objects from **WeightIt**. Note that several arguments that used to be documented here are now documented in display options. They are still available.

Usage

```
## S3 method for class 'ps'
bal.tab(ps,
    stop.method,
    int = FALSE,
    poly = 1,
    distance = NULL,
    addl = NULL,
    data = NULL,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
    v.threshold = NULL,
    ks.threshold = NULL,
    cluster = NULL,
    abs = FALSE,
    subset = NULL,
    quick = FALSE, ...)

## S3 method for class 'mnps'
bal.tab(mnps,
    stop.method,
    int = FALSE,
    poly = 1,
    distance = NULL,
    addl = NULL,
    data = NULL,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
```

```
    v.threshold = NULL,
    ks.threshold = NULL,
    cluster = NULL,
    pairwise = TRUE,
    focal = NULL,
    abs = FALSE,
    subset = NULL,
    quick = FALSE, ...)

## S3 method for class 'iptw'
bal.tab(iptw,
    stop.method,
    int = FALSE,
    poly = 1,
    distance.list = NULL,
    addl.list = NULL,
    data = NULL,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
    v.threshold = NULL,
    ks.threshold = NULL,
    pairwise = TRUE,
    abs = FALSE,
    subset = NULL,
    quick = FALSE, ...)

## S3 method for class 'ps.cont'
bal.tab(ps.cont,
    stop.method,
    int = FALSE,
    poly = 1,
    distance = NULL,
    addl = NULL,
    data = NULL,
    r.threshold = NULL,
    cluster = NULL,
    abs = FALSE,
    subset = NULL,
    quick = FALSE, ...)
```

Arguments

`ps, mnps, iptw, ps.cont`
: a `ps`, `mnps`, `iptw`, or `ps.cont` object; the output of a call to `ps()`, `mnps()`, or `iptw()` from **twang** or from a call to `ps.cont()` from **WeightIt**.

`stop.method`
: a string containing the names of the stopping methods used in the original call to `ps()`, `mnps()`, `iptw()`, or `ps.cont()`. Examples include "es.max" or

"ks.mean" for ps and mnps objects and "p.mean" or "s.max" for ps.cont objects. bal.tab will assess balance for the weights created by those stopping methods. The names can be abbreviated as long as the abbreviations are specific enough. If no stopping methods are provided, bal.tab will default to displaying balance for all available stopping methods. This argument used to be called full.stop.method, and that name still works.

int — logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well. In older versions of **cobalt**, setting int = TRUE displayed squares of covariates; to replicate this behavior, set int = 2.

poly — numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.

distance, distance.list — optional; either a vector or data.frame containing distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in data. Note that the propensity scores generated by ps() and iptw() (but not mnps() or ps.cont()) are automatically included.

addl, addl.list — an optional data frame or the quoted names of additional covariates for which to present balance. These may be covariates included in the original dataset but not included in the call to ps(), mnps(), iptw(), or ps.cont(). If variable names are specified, bal.tab() will look first in the argument to data, if specified, and next in the input object. For iptw objects, must be a list of additional covariate values described above, with one list entry per time period.

data — an optional data frame containing variables that might be named in arguments to distance, addl, and cluster.

continuous — whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options.

binary — whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options.

s.d.denom — whether the denominator for standardized differences (if any are calculated) should be the standard deviation of the treated group ("treated"), the standard deviation of the control group ("control"), or the pooled standard deviation ("pooled"), computed as the square root of the mean of the group variances. Abbreviations allowed. If not specified, bal.tab() will use "treated" if the estimand of the call to ps() is the ATT and "pooled" if the estimand is the ATE.

m.threshold — a numeric value for the threshold for mean differences. .1 is recommended.

v.threshold — a numeric value for the threshold for variance ratios. Will automatically convert to the inverse if less than 1.

ks.threshold — a numeric value for the threshold for Kolmogorov-Smirnov statistics. Must be between 0 and 1.

r.threshold	a numeric value for the threshold for correlations between covariates and treatment when treatment is continuous.
cluster	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in data or the CBPS object. See bal.tab.cluster for details.
pairwise	whether balance should be computed for pairs of treatments or for each treatment against all others combined. See bal.tab.multi for details.
focal	ignored. With multiple categorical treatments, bal.tab() automatically assigns a focal treatment based on the mnps object. See bal.tab.multi for details.
abs	logical; whether displayed balance statistics should be in absolute value or not.
subset	a logical vector denoting whether each observation should be included. It should be the same length as the variables in the original call to ps(), mnps(), or ps.cont(). NAs will be treated as FALSE. This can be used as an alternative to cluster to examine balance on subsets of the data.
quick	logical; if TRUE, will not compute any values that will not be displayed. Leave FALSE if computed values not displayed will be used later.
...	further arguments to control display of output. See display options for details.

Details

bal.tab.ps() generates a list of balance summaries for the ps object given, and functions similarly to bal.table() in **twang**.

All balance statistics are calculated whether they are displayed by print or not, unless quick = TRUE. The threshold values (m.threshold, v.threshold, ks.threshold, and r.threshold) control whether extra columns should be inserted into the Balance table describing whether the balance statistics in question exceeded or were within the threshold. Including these thresholds also creates summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure.

Value

For binary point treatments, if clusters are not specified, an object of class "bal.tab" containing balance summaries for the ps object. See bal.tab for details.

If clusters are specified, an object of class "bal.tab.cluster" containing balance summaries within each cluster and a summary of balance across clusters. See bal.tab.cluster for details.

If mnps() is used with multiple categorical treatments, an object of class "bal.tab.multi" containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See bal.tab.multi for details.

If ps.cont() is used with continuous treatments, means, mean differences, and variance ratios are replaced by (weighted) Pearson correlations between each covariate and treatment. The r.threshold argument works the same as m.threshold, v.threshold, or ks.threshold, adding an extra column to the balance table output and creating additional summaries for balance tallies and maximum imbalances. All arguments related to the calculation or display of mean differences or variance ratios are ignored. The int, addl, un, distance, and cluster arguments are still used as described above.

Author(s)

Noah Greifer

See Also

`bal.tab` for details of calculations. `bal.tab.cluster` for more information on clustered data. `bal.tab.multi` for more information on multiple categorical treatments. `bal.tab.msm` for more information on longitudinal treatments.

Examples

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

## Not run:
## Using ps() for generalized boosted modeling
ps.out <- ps(treat ~ age + educ + married + race +
             nodegree + re74 + re75, data = lalonde,
             stop.method = c("ks.mean", "es.mean"),
             estimand = "ATT", verbose = FALSE)
bal.tab(ps.out, stop.method = "es.max", un = TRUE,
        m.threshold = .1, disp.ks = TRUE)

## End(Not run)
```

---

bal.tab.weightit *Balance Statistics for WeightIt Objects*

Description

Generates balance statistics for `weightit` and `weightitMSM` objects from **WeightIt**. Note that several arguments that used to be documented here are now documented in display options. They are still available.

Usage

```
## S3 method for class 'weightit'
bal.tab(weightit,
    int = FALSE,
    poly = 1,
    distance = NULL,
    addl = NULL,
    data = NULL,
    continuous,
    binary,
    s.d.denom,
    m.threshold = NULL,
    v.threshold = NULL,
    ks.threshold = NULL,
```

```
    r.threshold = NULL,
    cluster = NULL,
    imp = NULL,
    pairwise = TRUE,
    focal = NULL,
    abs = FALSE,
    subset = NULL,
    quick = FALSE, ...)
```

Arguments

weightit	a weightit object; the output of a call to weightit() or weightitMSM() from the **WeightIt** package.
int	logical or numeric; whether or not to include 2-way interactions of covariates included in covs and in addl. If numeric, will be passed to poly as well. In older versions of **cobalt**, setting int = TRUE displayed squares of covariates; to replicate this behavior, set int = 2.
poly	numeric; the highest polynomial of each continuous covariate to display. For example, if 2, squares of each continuous covariate will be displayed (in addition to the covariate itself); if 3, squares and cubes of each continuous covariate will be displayed, etc. If 1, the default, only the base covariate will be displayed. If int is numeric, poly will take on the value of int.
distance	optional; either a vector or data.frame containing distance values (e.g., propensity scores) for each unit or a string containing the name of the distance variable in data. Note that the distance measure generated by weightit() is automatically included. For weightitMSM objects, must be a list of distance values described above, with one list entry per time period.
addl	an optional data frame or the quoted names of additional covariates for which to present balance. These may be covariates included in the original dataset but not included in the call to weightit(). If variable names are specified, bal.tab() will look first in the argument to data, if specified, and next in the weightit object. For weightitMSM objects, must be a list of additional covariate values described above, with one list entry per time period.
data	an optional data frame containing variables that might be named in arguments to distance, addl, and cluster.
continuous	whether mean differences for continuous variables should be standardized ("std") or raw ("raw"). Default "std". Abbreviations allowed. This option can be set globally using set.cobalt.options.
binary	whether mean differences for binary variables (i.e., difference in proportion) should be standardized ("std") or raw ("raw"). Default "raw". Abbreviations allowed. This option can be set globally using set.cobalt.options.
s.d.denom	whether the denominator for standardized differences (if any are calculated) should be the standard deviation of the treated group ("treated"), the standard deviation of the control group ("control"), or the pooled standard deviation ("pooled"), computed as the square root of the mean of the group variances. Abbreviations allowed. If not specified, bal.tab() will figure out which one is best based

	on the estimand of the `weightit` object: if ATT, "treated"; if ATC, "control", otherwise "pooled".
`m.threshold`	a numeric value for the threshold for mean differences. .1 is recommended.
`v.threshold`	a numeric value for the threshold for variance ratios. Will automatically convert to the inverse if less than 1.
`ks.threshold`	a numeric value for the threshold for Kolmogorov-Smirnov statistics. Must be between 0 and 1.
`r.threshold`	a numeric value for the threshold for correlations between covariates and treatment when treatment is continuous.
`cluster`	either a vector containing cluster membership for each unit or a string containing the name of the cluster membership variable in `data` or the CBPS object. See `bal.tab.cluster` for details.
`imp`	either a vector containing imputation indices for each unit or a string containing the name of the imputation index variable in `data` or the original data set used in the call to `weightit()`. See `bal.tab.imp` for details.
`pairwise`	whether balance should be computed for pairs of treatments or for each treatment against all others combined. See `bal.tab.multi` for details.
`focal`	Currently ignored.
`abs`	`logical`; whether displayed balance statistics should be in absolute value or not.
`subset`	a `logical` vector denoting whether each observation should be included. It should be the same length as the variables in the original call to `weightit()` or `weightitMSM()`. `NA`s will be treated as `FALSE`. This can be used as an alternative to `cluster` to examine balance on subsets of the data.
`quick`	`logical`; if `TRUE`, will not compute any values that will not be displayed. Leave `FALSE` if computed values not displayed will be used later.
`...`	further arguments to control display of output. See display options for details.

Details

`bal.tab.weightit()` generates a list of balance summaries for the `weightit` object given.

All balance statistics are calculated whether they are displayed by `print` or not, unless `quick = TRUE`. The threshold values (`m.threshold`, `v.threshold`, `ks.threshold`, and `r.threshold`) control whether extra columns should be inserted into the Balance table describing whether the balance statistics in question exceeded or were within the threshold. Including these thresholds also creates summary tables tallying the number of variables that exceeded and were within the threshold and displaying the variables with the greatest imbalance on that balance measure.

Value

For point treatments, if clusters and imputations are not specified, an object of class `"bal.tab"` containing balance summaries for the `weightit` object. See `bal.tab` for details.

If clusters are specified, an object of class `"bal.tab.cluster"` containing balance summaries within each cluster and a summary of balance across clusters. See `bal.tab.cluster` for details.

If imputations are specified, an object of class "bal.tab.imp" containing balance summaries for each imputation and a summary of balance across imputations, just as with clusters. See bal.tab.imp for details.

If both clusters and imputations are specified, an object of class "bal.tab.imp.cluster" containing summaries between and across all clusters and imputations.

If weightit() is used with multiple categorical treatments, an object of class "bal.tab.multi" containing balance summaries for each pairwise treatment comparison and a summary of balance across pairwise comparisons. See bal.tab.multi for details.

If weightitMSM() is used for longitudinal treatments, an object of class "bal.tab.msm" containing balance summaries for each time period and a summary of balance across time periods. See bal.tab.msm for details.

Author(s)

Noah Greifer

See Also

bal.tab for details of calculations. bal.tab.cluster for more information on clustered data. bal.tab.msm for more information on longitudinal treatments.

Examples

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

## Basic propensity score weighting
w.out1 <- weightit(treat ~ age + educ + race +
                   married + nodegree + re74 + re75,
                   data = lalonde, method = "ps")
bal.tab(w.out1, un = TRUE, m.threshold = .1,
        v.threshold = 2)

## Entropy balancing for the ATE
w.out2 <- weightit(treat ~ age + educ + race +
                   married + nodegree + re74 + re75,
                   data = lalonde, method = "entropy",
                   estimand = "ATE")
bal.tab(w.out2)
```

---

class-bal.tab.cluster *Using* bal.tab() *with Clustered Data*

Description

When using bal.tab() with clustered data, the output will be different from the case with single-level data, and there are some options that are common across all bal.tab methods. This page outlines the outputs and options in this case.

There are two main components of the output of bal.tab() with clustered data: the within-cluster balance summaries and the across-cluster balance summary. The within-cluster balance summaries display balance for units within each cluster separately.

The across-cluster balance summary pools information across the within-cluster balance summaries to simplify balance assessment. It provides a combination (e.g., mean or maximum) of each balance statistic for each covariate across all clusters. This allows you to see how bad the worst imbalance is and what balance looks like on average.

Arguments

There are four arguments for each bal.tab() method that can handle clustered data: cluster, which.cluster, cluster.summary, and cluster.fun.

A vector of cluster membership. This can be factor, character, or numeric vector. This argument is required to let bal.tab() know that the data is clustered. If a data argument is specified, this can also be the name of a variable in data that contains cluster membership.

which.cluster This is a display option that does not affect computation. If NULL (the default), all clusters in cluster will be displayed. If NA, no clusters will be displayed. Otherwise, can be a vector of cluster names or numerical indices for which to display balance. Indices correspond to the alphabetical order of cluster names (or the order of cluster levels if a factor).

cluster.summary
This is a display option that does not affect computation. If TRUE, the balance summary across clusters will be displayed. The default is TRUE, and if which.cluster is NA, it will automatically be set to TRUE.

cluster.fun This is a display option that does not affect computation. Can be "min", "mean", or "max" and corresponds to which function is used in the across-cluster summary to combine results across clusters. For example, if cluster.fun = "mean" the mean balance statistic across clusters will be displayed. The default when abs = FALSE in the bal.tab() call is to display all three. The default when abs = FALSE in the bal.tab() call is to display just the mean and max balance statistic.

Value

The output is a bal.tab.cluster object, which inherits from bal.tab. It has the following elements:

Cluster.Balance
For each cluster, a regular bal.tab object containing a balance table, a sample size summary, and other balance assessment tools, depending on which options are specified.

Cluster.Summary
The balance summary across clusters. This will include the combination of absolute mean differences for each covariate across all clusters according to the value of cluster.fun, and the same for variance ratios and KS statistics if requested.

`Observations` A table of sample sizes or effective sample sizes for each cluster before and after adjustment.

As with other methods, multiple weights can be specified, and values for all weights will appear in all tables.

See Also

`bal.tab`, `bal.tab.data.frame`, `print.bal.tab.cluster`

`class-bal.tab.imp` *Using* `bal.tab()` *with Multiply Imputed Data*

Description

When using `bal.tab()` with multiply imputed data, the output will be different from the case with a single data set. Multiply imputed data can only be used with the data.frame and formula `bal.tab()` methods. This page outlines the outputs and options available with multiply imputed data.

There are two main components of the output of `bal.tab()` with multiply imputed data: the within-imputation balance summaries and the across-imputation balance summary. The within-imputation balance summaries display balance for units within each imputed data set separately. In general, this will not be very useful because interest rarely lies in the qualities of any individual imputed data set.

The across-imputation balance summary pools information across the within-imputation balance summaries to simplify balance assessment. It provides the average (mean and median) and greatest (maximum) balance statistic for each covariate across all imputations. This allows you to see how bad the worst imbalance is and what balance looks like on average across the imputations.

Arguments

There are four arguments for each `bal.tab()` method that can handle multiply imputed data: `imp`, `which.imp`, `imp.summary`, and `imp.fun`.

`imp` A vector of imputation membership. This can be factor, character, or numeric vector. This argument is required to let `bal.tab()` know that the data is multiply imputed. If a `data` argument is specified, this can also be the name of a variable in `data` that contains imputation membership.

`which.imp` This is a display option that does not affect computation. If `NULL`, all imputations in `imp` will be displayed. If `NA` (the default), no imputations will be displayed. Otherwise, can be a vector of imputation indices for which to display balance.

`imp.summary` This is a display option that does not affect computation. If `TRUE`, the balance summary across imputations will be displayed. The default is `TRUE`, and if `which.imp` is `NA`, it will automatically be set to `TRUE`.

`imp.fun` This is a display option that does not affect computation. Can be "min", "mean", or "max" and corresponds to which function is used in the across-imputation summary to combine results across imputations. For example, if `imp.fun = "mean"` the mean balance statistic across imputations will be displayed. The default when `abs = FALSE` in the `bal.tab()` call is to display all three. The default when `abs = FALSE` in the `bal.tab()` call is to display just the mean and max balance statistic.

Value

The output is a `bal.tab.imp` object, which inherits from `bal.tab`. It has the following elements:

`Imputation.Balance`
: For each imputation, a regular `bal.tab` object containing a balance table, a sample size summary, and other balance assessment tools, depending on which options are specified.

`Balance.Across.Imputations`
: The balance summary across imputations. This will include the combination of absolute mean differences for each covariate across all imputations according to the value of `imp.fun`, and the same for variance ratios and KS statistics if requested.

`Observations` A table of sample sizes or effective sample sizes averaged across imputations before and after adjustment.

As with other methods, multiple weights can be specified, and values for all weights will appear in all tables.

Author(s)

Noah Greifer

See Also

`bal.tab`, `bal.tab.data.frame`, `print.bal.tab.imp`

---

class-bal.tab.msm *Using* `bal.tab()` *with Longitudinal Treatments*

Description

When using `bal.tab()` with longitudinal treatments, the output will be different from the case with point treatments, and there are some options that are common across all `bal.tab` methods for dealing with longitudinal data. This page outlines the outputs and options in this case.

There are two main components of the output of `bal.tab()` with longitudinal treatments: the time-point-specific balance summary and across-time-points balance summary. The time-point-specific balance summaries are standard point treatment balance summaries at each time point.

The across-time-points balance summary is, for each variable, the greatest imbalance across all time-point-specific balance summaries. If the greatest observed imbalance is tolerable, then all other imbalances for that variable will be tolerable too, so focusing on reducing the greatest imbalance is sufficient for reducing imbalance overall.

Arguments

There are two additional arguments for each `bal.tab()` method that can handle longitudinal treatments: `which.time` and `msm.summary`.

This is a display option that does not affect computation. If `NULL` (the default), all time points will be displayed. If `NA`, no time points will be displayed. Otherwise, can be a vector of treatment names or indices for which to display balance.

`which.time`, `msm.summary` This is a display option that does not affect computation. If `TRUE`, the balance summary across time points will be displayed. The default is `TRUE`, and if `which.time` is `NA`, it will automatically be set to `TRUE`.

Value

The output is a `bal.tab.msm` object, which inherits from `bal.tab`. It has the following elements:

`Time.Balance` For each time point, a regular `bal.tab` object containing a balance table, a sample size summary, and other balance assessment tools, depending on which options are specified.

`Balance.Across.Times`
The balance summary across time points. This will include the maximum absolute mean difference for each covariate across all time points, and the same for variance ratios and KS statistics if requested.

`Observations` A table of sample sizes or effective sample sizes for each time point before and after adjustment.

As with other methods, multiple weights can be specified, and values for all weights will appear in all tables.

Author(s)

Noah Greifer

See Also

`bal.tab`, `bal.tab.list`, `print.bal.tab.msm`

class-bal.tab.multi *Using* bal.tab() *with Multiple Categorical Treatments*

Description

When using bal.tab() with multiple categorical treatments, the output will be different from the case with binary or continuous treatments, and there are some options that are common across all bal.tab methods. This page outlines the outputs and options in this case.

There are two main components of the output of bal.tab() with multiple categorical treatments: the two-group treatment comparisons and the balance summary. The two-group treatment comparisons are standard binary treatment comparison either for pairs of groups (e.g., for treatments A, B, and C, "A vs. B", "A vs. C", and "B vs. C") or each group against the others.

The balance summary is, for each variable, the greatest imbalance across all two-group comparisons. So, for variable X1, if "A vs. B" had a standardized mean difference of 0.52, "A vs. C" had a standardized mean difference of .17, and "B vs. C" had a standardized mean difference of .35, the balance summary would have 0.52 for the value of the standardized mean difference for X1. The same goes for other variables and other measures of balance. If the greatest observed imbalance is tolerable, then all other imbalances for that variable will be tolerable too, so focusing on reducing the greatest imbalance is sufficient for reducing imbalance overall. (Note that when s.d.denom = "pooled", i.e., when the estimand is the ATE, the pooled standard deviation in the denominator will be the average of the standard deviations across all treatment groups, not just those used in the pairwise comparison.)

Arguments

There are four arguments for each bal.tab() method that can handle multiple categorical treatments: pairwise, focal, which.treat, and multi.summary.

pairwise

Whether to compute the two-group comparisons pairwise or not. If TRUE, bal.tab() will compute comparisons for each pair of treatments. This can be valuable if treatments are to be compared with one another (which is often the case). If FALSE, bal.tab() will compute balance for each treatment group against all other groups combined. This only makes sense if the ATE is desired. When focal is specified, pairwise is automatically set to TRUE.

focal

When one group is to be compared to multiple control groups in an ATT analysis, the group considered "treated" is the focal group. Only comparisons between other groups and the focal group are of interest. By specifying the name or index of the treatment condition considered focal, bal.tab() will only compute and display pairwise balance for treatment comparisons that include the focal group. For example, if "A" was to be compared with "B" and "C", "A" would be considered focal, and the comparison between groups "B" and "C" would not be computed. In general it is only a good idea to specify focal if an only if the ATT is sought.

which.treat

This is a display option that does not affect computation. When displaying the bal.tab output, which treatments should be displayed? If a vector of length 1 is entered, all comparisons involving that treatment group will be displayed.

If a vector of length 2 or more is entered, all comparisons involving treatments that both appear in the input will be displayed. For example, inputting `"A"` will display "A vs. B" and "A vs. C", while entering `c("A", "B")` will only display "A vs. B". `NA` indicates no treatment comparisons will be displayed, and `NULL` indicates all treatment comparisons will be displayed. `NA` is the default.

`multi.summary` This is a display option that does not affect computation. If `TRUE`, the balance summary across all comparisons will be displayed. This includes one row for each covariate with maximum balance statistic across all pairwise comparisons. Note that, if variance ratios or KS statistics are requested, the displayed values may not come from the same pairwise comparisons; that is, the greatest standardized mean difference and the greatest variance ratio may not come from the same comparison. The default is `TRUE`, and if `which.treat` is `NA`, it will automatically be set to `TRUE`.

Value

The output is a `bal.tab.multi` object, which inherits from `bal.tab`. It has the following elements:

`Pair.Balance` For each pair of treatment groups, a regular `bal.tab` object containing a balance table, a sample size summary, and other balance assessment tools, depending on which options are specified. If `focal` is specified, only the comparisons involving the focal group are computed. If `pairwise` is `FALSE`, the comparisons will be between each group and the other groups combined (labeled "Others").

`Balance.Across.Pairs`
The balance summary across two-group comparisons. This will include the greatest (i.e., maximum) absolute mean difference for each covariate across all comparisons computed, and the same for variance ratios and KS statistics if requested. Thresholds can be requested for each balance measure as with binary treatments.

`Observations` A table of sample sizes or effective sample sizes for each treatment group before and after adjustment.

As with other methods, multiple weights can be specified, and values for all weights will appear in all tables.

Author(s)

Noah Greifer

See Also

`bal.tab`, `bal.tab.data.frame`, `print.bal.tab.multi`

`class-bal.tab.subclass`

Using `bal.tab()` *with Subclassified Data*

Description

When using `bal.tab()` with subclassified data, i.e., data split into subclasses where balance may hold, the output will be different from the standard, non-subclassified case, and there is an additional option for controlling display. This page outlines the outputs and options in this case.

There are two main components of the output of `bal.tab()` with subclassified data: the balance within subclasses and the balance summary across subclasses. The within-subclass balance displays essentially are standard balance displays for each subclass, except that only "adjusted" values are available, because the subclassification itself is the adjustment.

The balance summary is, for each variable, a weighted average of the means and mean differences across subclasses. Each subclass statistic is weighed by the proportion of units corresponding to the target population (i.e., treated, control, or total sample) that are in that subclass. Variance ratios and KS statistics do not appear in this summary. Although this summary can be useful as a heuristic, it is important to ensure balance within each subclass as well.

Arguments

There are two arguments for `bal.tab()` that relate to subclasses: `subclass` and `disp.subclass`.

For the data.frame and formula methods of `bal.tab`, a vector of subclass membership or the name of the variable in `data` containing subclass membership. When using subclassification with a function compatible with **cobalt**, such as `matchit()` in **MatchIt**, this argument can be omitted because the subclass are in the output object.

`disp.subclass` This is a display option that does not affect computation. If `TRUE`, balance for each subclass will be displayed. This can be cumbersome with many subclasses or covariates. The default is `TRUE`.

Value

The output is a `bal.tab.subclass` object, which inherits from `bal.tab`. It has the following elements:

`Subclass.Balance`

A list of data frames containing balance information for each covariate in each subclass. Each data frame contains the following columns:

- `Type`: Whether the covariate is binary, continuous, or a measure of distance (e.g., the propensity score).
- `M.0.Adj`: The mean of the control group in the subclass.
- `M.1.Adj`: The mean of the treated group in the subclass.

- `Diff.Adj`: The (standardized) difference in means between the two groups in the subclass.
- `M.Threshold`: Whether or not the calculated mean difference exceeds or is within the threshold given by `m.threshold`. If `m.threshold` is `NULL`, this column will be `NA`.
- `V.Ratio.Adj`: The ratio of the variances of the two groups in the subclass. `NA` for binary variables. If less than 1, the reciprocal is reported.
- `V.Threshold`: Whether or not the calculated variance ratio exceeds or is within the threshold given by `v.threshold` for continuous variables. If `v.threshold` is `NULL`, this column will be `NA`.
- `KS.Adj`: The KS statistic for the two groups in the subclass. `NA` for binary variables.
- `KS.Threshold`: Whether or not the calculated KS statistic exceeds or is within the threshold given by `ks.threshold` for continuous variables. If `ks.threshold` is `NULL`, this column will be `NA`.

`Balance.Across.Subclass`
: A data frame containing balance statistics for each covariate aggregated across subclasses and for the original sample (i.e., unadjusted). Variance ratios and KS statistics are not reported here.

`Balanced.Means.Subclass`
: If `m.threshold` is specified, a table tallying the number of variables in each subclass that exceed or are within the threshold for mean differences.

`Max.Imbalance.Means.Subclass`
: If `m.threshold` is specified, a table displaying the variable in each subclass with the greatest absolute mean difference.

`Balanced.Variances.Subclass`
: If `v.threhsold` is specified, a table tallying the number of variables in each subclass that exceed or are within the threshold for variance ratios.

`Max.Imbalance.Variance.Subclass`
: If `v.threshold` is specified, a table displaying the variable in each subclass with the greatest variance ratio.

`Balanced.KS.Subclass`
: If `ks.threhsold` is specified, a table tallying the number of variables in each subclass that exceed or are within the threshold for KS statistics.

`Max.Imbalance.KS.Subclass`
: If `ks.threshold` is specified, a table displaying the variable in each subclass with the greatest KS statistics.

`Subclass.Observations`
: A table displaying the sample sizes in each subclass.

Author(s)

Noah Greifer

See Also

`link{bal.tab}`, `bal.tab.data.frame`, `print.bal.tab.subclass`

f.build *Convenient Formula Generation*

Description

`f.build()` returns a formula of the form `y ~ x1 + x2 + ...` from a data frame input. It can be much quicker to use `f.build()` than to hand-write the precise formula, which may contain errors. It can be used in place of a formula in, for example, `glm()`, `matchit()`, or `bal.tab()`.

Usage

```
f.build(y, rhs)
```

Arguments

`y`	the quoted name of the response (left hand side) variable in the formula. Only one variable is supported. If missing, `NULL`, or the empty string (`""`), the formula will have no response variable.
`rhs`	a data frame whose variable names will be the terms on the right hand side of the formula, or a character vector whose values will be the terms on the right hand side of the formula.

Value

an object of class `"formula"`.

Examples

```
data(lalonde)
covs <- subset(lalonde, select = -c(treat, re78))
lm(f.build("treat", covs), data = lalonde)
```

get.w *Extract Weights from Preprocessing Objects*

Description

Extracts weights from the outputs of preprocessing functions.

Usage

```
get.w(x, ...)

## S3 method for class 'matchit'
get.w(x, ...)

## S3 method for class 'ps'
get.w(x, stop.method = NULL, estimand = NULL, s.weights = FALSE, ...)

## S3 method for class 'mnps'
get.w(x, stop.method = NULL, s.weights = FALSE, ...)

## S3 method for class 'iptw'
get.w(x, stop.method = NULL, s.weights = FALSE, ...)

## S3 method for class 'Match'
get.w(x, ...)

## S3 method for class 'CBPS'
get.w(x, estimand = NULL, ...)

## S3 method for class 'ebalance'
get.w(x, treat, ...)

## S3 method for class 'optmatch'
get.w(x, ...)

## S3 method for class 'weightit'
get.w(x, s.weights = FALSE, ...)

## S3 method for class 'designmatch'
get.w(x, treat, ...)
```

Arguments

x	output from the corresponding preprocessing packages.
stop.method	the name of the stop method used in the original call to ps() or mnps() in **twang**, e.g., "es.mean". If empty, will return weights from all stop method available into a data.frame. Abbreviations allowed.
estimand	if weights are computed using the propensity score (i.e., for the ps and CBPS methods), which estimand to use to compute the weights. If "ATE", weights will be computed as 1/ps for the treated group and 1/(1-ps) for the control group. If "ATT", weights will be computed as 1 for the treated group and ps/(1-ps) for the control group. If NULL, get.w() will try to figure out which estimand is desired based on the object.
treat	a vector of treatment status for each unit. This is required for methods that

include `treat` as an argument. The treatment variable that was used in the original preprocessing function call should be used.

`s.weights` whether the sampling weights included in the original call to the fitting function should be included in the weights. If `TRUE`, the returned weights will be the product of the balancing weights estimated by the fitting function and the sampling weights. If `FALSE`, only the balancing weights will be returned.

`...` further arguments passed to or from other methods.

Details

The output of `get.w()` can be used in calls to the formula and data frame methods of `bal.tab()` (see example below). In this way, the output of multiple preprocessing packages can be viewed simultaneously and compared. The weights can also be used in `weights` statements in regression methods to compute weighted effects.

twang has a function called `get.weights()` that performs the same function on `ps` objects but offers slightly finer control. Note that the weights generated by `get.w()` for `ps` objects do not include sampling weights.

When sampling weights are used with `CBPS()`, the returned weights will already have the sampling weights incorporated. To retrieve the balancing weights on their own, divide the returned weights by the original sampling weights. For other packages, the balancing weights are returned separately unless `s.weights = TRUE`, which means they must be multiplied by the sampling weights for effect estimation.

Value

A vector or data frame of weights for each unit. These may be matching weights or balancing weights.

Author(s)

Noah Greifer

See Also

`get.weights` in **twang**.

Examples

```
## Not run:
data("lalonde", package = "cobalt")
library("MatchIt"); library("WeightIt")

m.out <- matchit(treat ~ age + educ + race, data = lalonde)

w.out <- weightit(treat ~ age + educ + race, data = lalonde,
                  estimand = "ATT")

bal.tab(treat ~ age + educ + race, data = lalonde,
        weights = data.frame(matched = get.w(m.out),
```

```
                            weighted = get.w(w.out)),
          method = c("matching", "weighting"),
          estimand = "ATT")

## End(Not run)
```

`lalonde` *Lalonde's National Supported Work Demonstration data*

Description

One of the datasets used by Dehejia and Wahba in their paper "Causal Effects in Non-Experimental Studies: Reevaluating the Evaluation of Training Programs." Versions of this data set have been used as an example data set in **MatchIt**, **twang**, **Matching**, and **CBPS**. The data set `lalonde_mis` is the same but with some values missing (set to `NA`).

Usage

```
data("lalonde")
data("lalonde_mis")
```

Format

A data frame with 614 observations on the following 9 variables.

`treat` 1 if treated in the National Supported Work Demonstration, 0 if from the Current Population Survey

`age` age

`educ` years of education

`race` factor; black, Hispanic (hispan), or white

`married` 1 if married, 0 otherwise

`nodegree` 1 if no degree, 0 otherwise

`re74` earnings in 1974 (pretreatment)

`re75` earnings in 1975 (pretreatment)

`re78` earnings in 1978 (outcome)

Source

http://users.nber.org/~rdehejia/data/nswdata2.html

References

Lalonde, R. (1986). Evaluating the econometric evaluations of training programs with experimental data. American Economic Review 76: 604-620.

Dehejia, R.H. and Wahba, S. (1999). Causal Effects in Nonexperimental Studies: Re-Evaluating the Evaluation of Training Programs. Journal of the American Statistical Association 94: 1053-1062.

love.plot *Generate Balance Plots for Publication*

Description

Generates a "Love" plot graphically displaying covariate balance before and after adjusting.

Usage

```
love.plot(x,
    stat = c("mean.diffs", "variance.ratios",
             "ks.statistics"),
    threshold = NULL,
    abs = TRUE,
    var.order = NULL,
    no.missing = TRUE,
    var.names = NULL,
    drop.distance = FALSE,
    agg.fun = c("mean", "max", "range"),
    colors = NULL,
    shapes = NULL,
    line = FALSE,
    ...)
```

Arguments

x	a bal.tab object; the output of a call to bal.tab. m.threshold, v.threshold, and r.threshold can be used in bal.tab instead of love.plot's threshold parameter.
stat	character; which statistic should be reported if treatment is binary. The options are "mean.diffs" for mean differences (standardized or not according the options selected in bal.tab object), "variance.ratios" for variance ratios, and "ks.statistics" for Kolmogorov-Smirnov statistics. "mean.diffs" is the default. Abbreviations allowed.
threshold	numeric; an optional value to be used as a threshold marker in the plot. Overrides the threshold set in the bal.tab object.
abs	logical; whether to present the statistic in absolute value or not if stat = "mean.diffs" or the treatment variable is continuous. See Details.
var.order	character; how to order the variables in the plot. If NULL, they will be displayed in the same order as in the call to bal.tab, which is the order of the underlying data set. If "alphabetical", they will be displayed in alphabetical order. If "adjusted", they will be ordered by the balance statistic of the adjusted sample. if "unadjusted", they will be ordered by the balance statistic of the unadjusted sample. "unadjusted" looks the nicest, but NULL or "alphabetical" should be used if comparing variables across data sets to maintain variable order. If multiple plots are produced simultaneously (i.e., for individual clusters or imputations), var.order cannot be "unadjusted" or "adjusted".

`no.missing` `logical`; whether to drop rows for variables for which the statistic has a value of `NA`, for example, variance ratios for binary variables. If `FALSE`, there will be rows for these variables but no points representing their value. Default is `TRUE`, so that variables with missing values are absent.

`var.names` an optional object providing alternate names for the variables in the plot, which will otherwise be the variable names as they are stored. This may be useful when variables have ugly names. See Details on how to specify `var.names`. `link{var.names}` can be a useful tool for extracting the names from the `bal.tab` object.

`drop.distance` `logical`; whether to ignore the distance measure (if there are any) in plotting.

`agg.fun` if balance is to be displayed across clusters or imputations rather than within a single cluster or imputation, which summarizing function (mean, max, or range) of the balance statistics should be used. If "range" is entered, `love.plot` will display a line from the min to the max with a point at the mean for each covariate; it can only be used if `quick = FALSE` in the `bal.tab` call. Abbreviations allowed; "mean" is default.

`colors` the colors of the points on the plot. See 'Color Specification' at `par` or the `ggplot2` aesthetic specifications page. The first value corresponds to the color for the unadjusted sample, and the second color to the adjusted sample. If only one is specified, it will apply to both. Defaults to the default **ggplot2** colors.

`shapes` the shapes of the points on the plot. Must be one or two numbers between 1 and 25 or the name of a valid shape. See the `ggplot2` aesthetic specifications page for valid options. Values 21 to 25 and "filled" shapes are recommended. The first value corresponds to the shape for the unadjusted sample, and the second color to the adjusted sample. If only one is specified, it will apply to both. Defaults to 21 (`"circle filled"`).

`line` `logical`; whether to display a line connecting the points for each sample.

`...` Options for display of the plot. The following arguments are currently accepted:

`size` `numeric`; the size of the points on the plot. Defaults to 1.

`title` `character`; the title of the plot.

`subtitle` `character`; the subtitle of the plot.

`sample.names` `character`; new names to be given to the samples (i.e., in place of "Unadjusted" and "Adjusted").

`limits` `numeric`; the bounds for the x-axis of the plot. Must a vector of length 2 in ascending order.

`which.cluster` which clusters to display. Overrides the `which.cluster` option in the original `bal.tab` object.

`which.imp` which imputations to display. Overrides the `which.imp` option in the original `bal.tab` object.

`which.treat` which treatment groups to display. Overrides the `which.treat` option in the original `bal.tab` object.

`disp.subclass` whether to display individual subclasses. Overrides the `disp.subclass` option in the original `bal.tab` object.

Details

love.plot uses ggplot from the **ggplot2** package, and (invisibly) returns a "ggplot" object. This means that users can edit aspects of the plot using ggplot2 syntax.

The default in love.plot is to present variables as they are named in the output of the call to bal.tab, so it is important to know this output before specifying alternate variable names when using var.names, as the displayed variable names may differ from those in the original data.

There are several ways to specify alternate names for presentation in the displayed plot using the var.names argument by specifying a list of old and new variable names, pairing the old name with the new name. You can do this in three ways: 1) use a vector or list of new variable names, with the names of the values the old variable names; 2) use a data frame with exactly one column containing the new variable names and the row names containing the old variable names; or 3) use a data frame with two columns, the first (or the one named "old") containing the old variable names and the second (or the one named "new") containing the new variable names. If a variable in the output from bal.tab is not provided in the list of old variable names, love.plot will use the original old variable name.

love.plot can replace old variables names with new ones based on exact matching for the name strings or matching using the variable name components. For example, if a factor variable "X" with levels "a", "b", and "c" is displayed with love.plot, the variables "X_a", "X_b", and "X_c" will be displayed. You can enter replacement names for all three variables individually with var.names, or you can simply specify a replacement name for "X", and "X" will be replaced by the given name in all instances it appears, including not just factor expansions, but also polynomials and interactions in int = TRUE in the original bal.tab call. In an interaction with another variable, say "Y", there are several ways to replace the name of the interaction term "X_a * Y". If the entire string ("X_a * Y") is included in var.names, the entire string will be replaced. If "X_a" is included in var.name, only it will be replaced (and it will be replaced everywhere else it appears). If "X" is included in var.name, only it will be replaced (and it will be replaced everywhere else it appears). See example at var.names.

love.plot can be used with clusters, imputations, or both as well. The cluster or imputation arguments must be specified in the call to bal.tab. Several types of plots can be requested: a plot aggregating over all imputations across all clusters, a plot displaying individual clusters aggregating over imputations (if any), a plot displaying individual imputations across clusters, or a plot displaying individual clusters within one imputation (if any) or individual imputations for one cluster. The choice of these displays is controlled by the arguments to which.cluster and which.imp. If either of them are NA, the plot will aggregate over that collection. If either are individual values, the plot will display the values for those specific clusters or imputations. If either of them are NULL, the plot will display the values for all clusters or imputations. If both clusters and imputations are specified, at least one of which.cluster or which.imp must be a single value, or NA. When aggregating, an argument should be specified to agg.fun referring to whether the mean, minimum ("min"), or maximum ("max") balance statistic or range ("range") of balance statistics for each covariate should be presented in the plot.

With subclasses, balance will be displayed for the unadjusted sample and the aggregated subclassified sample. If disp.subclass is TRUE, each subclass will be displayed additionally as a number on the plot.

If no aggregation (e.g., over clusters or imputations) is to take place, abs will be TRUE if it is TRUE in the call to bal.tab or in the call to love.plot. Otherwise, if balance is requested with multinomial

or longitudinal treatments, abs is set to TRUE, and if balance is requested with clustered or multiply imputed data, abs will take on the value of abs set in the original call to bal.tab.

Value

A "ggplot" object, returned invisibly.

Note

love.plot can also be called by using plot on a bal.tab object. That is, the "love." prefix is optional.

Author(s)

Noah Greifer

See Also

bal.tab

Examples

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

## Nearest Neighbor matching
m.out1 <- matchit(treat ~ age + educ + race +
                  married + nodegree + re74 + re75,
                  data = lalonde)

love.plot(bal.tab(m.out1), stat = "mean.diffs", threshold = .1,
          var.order = "unadjusted")

## Using alternate variable names
v <- data.frame(old = c("age", "educ", "race_black", "race_hispan",
                        "race_white", "married", "nodegree", "re74",
                        "re75"),
                new = c("Age", "Years of Education", "Black",
                        "Hispanic", "White", "Married", "No Degree",
                        "Earnings 1974", "Earnings 1975"))

love.plot(bal.tab(m.out1), stat = "mean.diffs", threshold = .1,
          var.order = "unadjusted", var.names = v)

#Changing colors and shapes and adding lines
love.plot(bal.tab(m.out1), stat = "mean.diffs", threshold = .1,
          var.order = "unadjusted", var.names = v, abs = TRUE,
          shapes = c(22, 25), colors = c("darkblue", "lightblue"),
          line = TRUE)
```

options-display *Options for Displaying* bal.tab *Output*

Description

Several additional arguments can be passed to bal.tab() that control the display of the output but don't affect computations. These arguments are documented here.

Not all arguments are applicable to all uses of bal.tab(); for example, disp.subclass, which controls whether individual subclasses are displayed when subclassification is used, won't do anything when subclassification is not used.

Note that when quick = TRUE is set in the call to bal.tab, setting any of these arguments to FALSE can prevent some values from being computed, which can have unintended effects. For example, setting quick = TRUE and un = FALSE in bal.tab and then using love.plot on the output will yield an error because love.plot depends on the computation of the unadjusted balance statistics, which are not computed when quick = TRUE and un = FALSE.

Arguments

imbalanced.only
: logical; whether to display only the covariates that failed to meet at least one of balance thresholds. Default is FALSE, so all covariates are displayed.

un
: logical; whether to print statistics for the unadjusted sample as well as for the adjusted sample. Default is FALSE, so only the statistics for the adjusted sample are displayed.

disp.means
: logical; whether to print the group means in balance output. Default is FALSE, so means are not displayed.

disp.sds
: logical; whether to print the group standard deviations in balance output. Default is FALSE, so standard deviations are not displayed.

disp.v.ratio
: logical; whether to display variance ratios in balance output for binary and multinomial treatments. Default is FALSE, so variance ratios are not displayed.

disp.ks
: logical; whether to display Kolmogorov-Smirnov (KS) statistics in balance output for binary and multinomial treatments. Default is FALSE, so KS statistics are not displayed.

disp.bal.tab
: logical; whether to display the table of balance statistics. Default is TRUE, so the balance table is displayed.

factor_sep
: character; the string used to separate factor variables from their levels when variable names are printed. Default is "_".

int_sep
: character; the string used to separate two variables involved in an interaction when variable names are printed. Default is " * ". Older versions of **cobalt** used "_".

When subclassification is used

`disp.subclass` `logical`; whether to display balance information for individual subclasses if subclassification is used in conditioning. See `bal.tab.subclass` for details. Default is `FALSE`, so individual subclasses are not displayed.

When the treatment is multinomial

`which.treat` For which treatments or treatment combinations balance tables should be displayed. If a vector of length 1 is entered, all comparisons involving that treatment group will be displayed. If a vector of length 2 or more is entered, all comparisons involving treatments that both appear in the input will be displayed. For example, setting `which.treat = "A"` will display "A vs. B" and "A vs. C", while setting `which.treat = c("A", "B")` will only display "A vs. B". `NA` indicates no treatment comparisons will be displayed, and `NULL` indicates all treatment comparisons will be displayed. Default is `NA`. See `bal.tab.multi`.

`multi.summary` `logical`; whether to display the balance summary across all treatment pairs. This includes one row for each covariate with maximum balance statistic across all pairwise comparisons. Note that, if variance ratios or KS statistics are requested, the displayed values may not come from the same pairwise comparisons; that is, the greatest standardized mean difference and the greatest variance ratio may not come from the same comparison. Default is `TRUE`, and if `which.treat` is `NA`, it will automatically be set to `TRUE`. See `bal.tab.multi`.

When clusters are present

`which.cluster` For which clusters balance tables should be displayed. If `NULL` (the default), all clusters in `cluster` will be displayed. If `NA`, no clusters will be displayed. Otherwise, can be a vector of cluster names or numerical indices for which to display balance. Indices correspond to the alphabetical order of cluster names (or the order of cluster levels if a factor). See `bal.tab.cluster`.

`cluster.summary`

`logical`; whether to display the balance summary across clusters. Default is `TRUE`, and if `which.cluster` is `NA`, it will automatically be set to `TRUE`. See `bal.tab.cluster`.

`cluster.fun` Which function is used in the across-cluster summary to combine results across clusters. Can be "min", "mean", or "max". For example, if `cluster.fun = "mean"` the mean balance statistic across clusters will be displayed. The default when `abs = FALSE` in the `bal.tab()` call is to display all three. The default when `abs = FALSE` in the `bal.tab()` call is to display just the mean and max balance statistic. See `bal.tab.cluster`.

When multiple imputations are present

`which.imp` For which imputations balance tables should be displayed. If `NULL`, all imputations in `imp` will be displayed. If `NA` (the default), no imputations will be displayed. Otherwise, can be a vector of imputation indices for which to display balance. See `bal.tab.imp`.

`imp.summary` `logical`; whether to display the balance summary across imputations. Default is `TRUE`, and if `which.cluster` is `NA`, it will automatically be set to `TRUE`. See `bal.tab.imp`.

`imp.fun` Which function is used in the across-imputation summary to combine results across imputations. Can be "min", "mean", or "max". For example, if `imp.fun = "mean"` the mean balance statistic across imputations will be displayed. The default

when abs = FALSE in the bal.tab() call is to display all three. The default when abs = FALSE in the bal.tab() call is to display just the mean and max balance statistic. See bal.tab.imp.

When the treatment is longitudinal

which.time — For which time points balance tables should be displayed. If NULL, all time points will be displayed. If NA, no time points will be displayed. Otherwise, can be a vector of treatment names or indices for which to display balance. Default is NULL. See bal.tab.msm.

msm.summary — logical; whether to display the balance summary across time points. Default is TRUE, and if which.time is NA, it will automatically be set to TRUE. See bal.tab.msm.

Details

These arguments used to be named arguments to bal.tab, but were relegated to arguments passed through ... to consolidate documentation and make it easier to add new features and options. For users of previous versions of **cobalt**, nothing should change as far as how the arguments are used, but new options may be added.

In addition to being able to be specified as arguments, if you find you frequently set a display option to something other than its default, you can set that as a global option (for the present R session) using set.cobalt.options and retrieve it using get.cobalt.options. Note that global options cannot be set for which.cluster, which.imp, which.treat, or which.time.

See Also

bal.tab, print.bal.tab

---

print.bal.tab *Print Results of a Call to* bal.tab()

Description

Prints bal.tab() output in a clean way. Provides options for printing.

Usage

```
## S3 method for class 'bal.tab'
print(x,
    disp.m.threshold = "as.is",
    disp.v.threshold = "as.is",
    disp.ks.threshold = "as.is",
    disp.r.threshold = "as.is",
    imbalanced.only = "as.is",
    un = "as.is",
    disp.bal.tab = "as.is",
    disp.means = "as.is",
```

```
    disp.sds = "as.is",
    disp.v.ratio = "as.is",
    disp.ks = "as.is",
    digits = max(3, getOption("digits") - 3),
    ...)

## S3 method for class 'bal.tab.subclass'
print(x,
    disp.m.threshold = "as.is",
    disp.v.threshold = "as.is",
    disp.ks.threshold = "as.is",
    disp.r.threshold = "as.is",
    imbalanced.only = "as.is",
    un = "as.is",
    disp.bal.tab = "as.is",
    disp.means = "as.is",
    disp.sds = "as.is",
    disp.v.ratio = "as.is",
    disp.ks = "as.is",
    disp.subclass = "as.is",
    digits = max(3, getOption("digits") - 3),
    ...)

## S3 method for class 'bal.tab.cluster'
print(x,
    disp.m.threshold = "as.is",
    disp.v.threshold = "as.is",
    disp.ks.threshold = "as.is",
    disp.r.threshold = "as.is",
    imbalanced.only = "as.is",
    un = "as.is",
    disp.bal.tab = "as.is",
    disp.means = "as.is",
    disp.sds = "as.is",
    disp.v.ratio = "as.is",
    disp.ks = "as.is",
    which.cluster,
    cluster.summary = "as.is",
    cluster.fun = "as.is",
    digits = max(3, getOption("digits") - 3),
    ...)

## S3 method for class 'bal.tab.imp'
print(x,
    disp.m.threshold = "as.is",
    disp.v.threshold = "as.is",
    disp.ks.threshold = "as.is",
    disp.r.threshold = "as.is",
```

```
    imbalanced.only = "as.is",
    un = "as.is",
    disp.bal.tab = "as.is",
    disp.means = "as.is",
    disp.sds = "as.is",
    disp.v.ratio = "as.is",
    disp.ks = "as.is",
    which.imp,
    imp.summary = "as.is",
    imp.fun = "as.is",
    digits = max(3, getOption("digits") - 3),
    ...)

## S3 method for class 'bal.tab.imp.cluster'
print(x,
    disp.m.threshold = "as.is",
    disp.v.threshold = "as.is",
    disp.ks.threshold = "as.is",
    disp.r.threshold = "as.is",
    imbalanced.only = "as.is",
    un = "as.is",
    disp.bal.tab = "as.is",
    disp.means = "as.is",
    disp.sds = "as.is",
    disp.v.ratio = "as.is",
    disp.ks = "as.is",
    which.cluster,
    cluster.summary = "as.is",
    cluster.fun = "as.is",
    which.imp,
    imp.summary = "as.is",
    imp.fun = "as.is",
    digits = max(3, getOption("digits") - 3),
    ...)

    ## S3 method for class 'bal.tab.multi'
print(x,
    disp.m.threshold = "as.is",
    disp.v.threshold = "as.is",
    disp.ks.threshold = "as.is",
    imbalanced.only = "as.is",
    un = "as.is",
    disp.bal.tab = "as.is",
    disp.means = "as.is",
    disp.sds = "as.is",
    disp.v.ratio = "as.is",
    disp.ks = "as.is",
    which.treat,
```

```
    multi.summary = "as.is",
    digits = max(3, getOption("digits") - 3),
    ...)

    ## S3 method for class 'bal.tab.msm'
print(x,
    disp.m.threshold = "as.is",
    disp.v.threshold = "as.is",
    disp.ks.threshold = "as.is",
    disp.r.threshold = "as.is",
    imbalanced.only = "as.is",
    un = "as.is",
    disp.bal.tab = "as.is",
    disp.means = "as.is",
    disp.sds = "as.is",
    disp.v.ratio = "as.is",
    disp.ks = "as.is",
    which.cluster,
    cluster.summary = "as.is",
    cluster.fun = "as.is",
    which.time,
    msm.summary = "as.is",
    digits = max(3, getOption("digits") - 3),
    ...)
```

Arguments

`x`	a `bal.tab` object; the output of a call to `bal.tab()`.
`disp.m.threshold`	whether to display output related to specifying `m.threshold` in the call to `bal.tab()`, which includes the Mean Difference Threshold column in the Balance table, the Mean Difference Balance Tally, and the variable with the Maximum Mean Difference Imbalance. Either `FALSE` or `"as.is"`.
`disp.v.threshold`	whether to display output related to specifying `v.threshold` in the call to `bal.tab()`, which includes the Variance Ratio Threshold column in the Balance table, the Variance Ratio Balance Tally, and the variable with the Maximum Variance Ratio Imbalance. Either `FALSE` or `"as.is"`.
`disp.ks.threshold`	whether to display output related to specifying `ks.threshold` in the call to `bal.tab()`, which includes the KS Threshold column in the Balance table, the KS Balance Tally, and the variable with the Maximum KS Statistic. Either `FALSE` or `"as.is"`.
`disp.r.threshold`	whether to display output related to specifying `r.threshold` in the call to `bal.tab()` with a continuous treatment, which includes the Correlation Threshold column in the Balance table, the Correlation Balance Tally, and the variable with the Maximum Correlation Imbalance. Either `FALSE` or `"as.is"`.

`imbalanced.only`
: whether to display only the covariates that failed to meet at least one of balance thresholds. Depends only on whether threshold were initial set in the call to `bal.tab()` and not on any arguments to `print()` (except `disp.bal.tab`).

`un`
: whether to display balance values for the unadjusted sample. Ignored (and set to `TRUE`) if no conditioning was performed.

`disp.bal.tab`
: whether to display the table of balance statistics. If `FALSE`, only other values (e.g., the call, sample sizes, balance tallies, and maximum imbalances) will be presented.

`disp.means`
: whether to print the group means in balance output.

`disp.sds`
: whether to print the group standard deviations in balance output.

`disp.v.ratio`
: whether to display variance ratios in balance output.

`disp.ks`
: whether to display KS statistics in balance output.

`digits`
: the number of digits to display.

`disp.subclass`
: whether to display balance information for individual subclasses if subclassification is used in conditioning.

`which.cluster`
: which cluster(s) to display. If `NULL`, all clusters will be displayed. If `NA`, no clusters will be displayed. Otherwise, can be a vector of cluster names or numerical indices for which to display balance. Indices correspond to the alphabetical order of cluster names. To display the clusters requested in the original call to `bal.tab()`, omit this argument, as specifying `"as.is"` will request a cluster called "as.is.".

`cluster.summary`
: whether to display the cluster summary table. If `which.cluster` is `NA`, `cluster.summary` will be set to `TRUE`.

`cluster.fun`
: a character vector of functions of balance statistics to display when displaying balance across clusters. Can be "mean", "min", or "max". More than one are allowed.

`which.imp`
: which imputation(s) to display. If `NULL`, all imputations will be displayed. If `NA`, no imputations will be displayed. Otherwise, can be a vector of imputations numbers for which to display balance. To display the imputations requested in the original call to `bal.tab()`, omit this argument, or enter `"as.is"`.

`imp.summary`
: whether to display the imputation summary table. If `which.imp` is `NA`, `cluster.summary` will be set to `TRUE`.

`imp.fun`
: a character vector of functions of balance statistics to display when displaying balance across imputations. Can be "mean", "min", or "max". More than one are allowed.

`which.treat`
: which treatments to display when multiple categorical treatments are used. See `bal.tab.multi` for details.

`multi.summary`
: `logical`; whether to display the balance summary table across pairwise comparisons when multiple categorical treatments are used. See `bal.tab.multi` for details.

`which.time`
: which time periods to display if longitudinal treatments are used. See `bal.tab.msm` for details.

msm.summary	logical; whether to display the balance summary table across time periods when longitudinal treatments are used. See bal.tab.msm for details.
...	further arguments passed to or from other methods.

Details

Simply calling bal.tab() will print its results, but it can be useful to store the results into an object and print them again later, possibly with different print options specified. The print() function automatically dispatches the correct method for the bal.tab object given. For balance tables generated from using weighting, matching, or no adjustment, print.bal.tab() will be used. For balance tables generated from using weighting, matching, or no adjustment with clusters, print.bal.tab.cluster() will be used. For balance tables generated from using subclassification, print.bal.tab.subclass() will be used. For balance tables generated from using weighting, matching, or no adjustment with multiply imputed data, print.bal.tab.imp() will be used. For balance tables generated from using weighting, matching, or no adjustment with clusters and multiply imputed data, print.bal.tab.imp.cluster() will be used.

For all parameters except disp.m.threshold, disp.v.threshold, disp.ks.threshold, disp.r.threshold, and which.cluster, either omitting the argument or setting it to "as.is" will use the corresponding print option stored in the bal.tab object, which results from the original call to bal.tab().

For disp.m.threshold, disp.v.threshold, disp.ks.threshold, and disp.r.threshold, setting the argument to FALSE will display the results as if the corresponding threshold value had been omitted or set to NULL in the original call to bal.tab(). If the original threshold was omitted or set to NULL, a new threshold cannot be set without a new call to bal.tab(), so TRUE is not an acceptable option here.

For which.cluster, to retain the display option of the original call to bal.tab(), the argument must omitted, as using "as.is" would cause print() to attempt to display balance for a cluster called "as.is". If such a cluster existed and it was desired, it would otherwise be impossible to display it.

Any parameter used in bal.tab() for calculations, such as int, addl, or distance, cannot be used with print(); only those parameters listed above, those that solely determine printing options, can be used. To change computation options, a new call to bal.tab() must be performed.

Note

If quick = TRUE in the original call to bal.tab(), some values may not be calculated, in which case using print() will not display these values even when requested. For example, if disp.means = FALSE and quick = TRUE in the original call to bal.tab(), setting disp.means = TRUE in print() will not print the covariate means because they were not calculated.

Author(s)

Noah Greifer

See Also

print, bal.tab display options for further information on some of these options.

Examples

```
## Not run:
b <- bal.tab(x, un = TRUE, v.threshold = 2)
print(b, un = FALSE, disp.v.threshold = FALSE)

## End(Not run)
```

`set.cobalt.options` *Set options in cobalt*

Description

Makes it easier to set **cobalt** options. `set.cobalt.options` is essentially a wrapper for `options` but performs several checks, and `get.cobalt.options` is essentially a wrapper for `getOption`.

Usage

```
set.cobalt.options(..., default = FALSE)

get.cobalt.options(...)
```

Arguments

`...`	For `set.cobalt.options`, `bal.tab` parameters and the values they should take. These should be the name of the parameter in `bal.tab` without `"cobalt_"` preceding them. See examples. If any values are `NULL`, the corresponding options will be set back to their defaults. For `get.cobalt.options`, one or more strings containing the name of a parameter option to be retrieved. See examples. If empty, all available options and their values will be returned.
`default`	if `TRUE`, sets all **cobalt** options not named in `...` to their default values.

Details

When an option is set to `NULL`, it is set to its default value. The defaults are not displayed but are listed on the help pages where they appear. Most options correspond to display options, which can be accessed here. Some others (e.g., `continous` and `binary`) are described on the `bal.tab` help page.

See Also

`options`

`display_options` for some arguments that can be set via options.

Examples

```
# Set un to be TRUE to always display unadjusted
# balance measures and set binary to "std" to
# produce standardized mean differences for
# binary variables.

set.cobalt.options(un = TRUE, binary = "std")

# Note: the above is equivalent to:
# options(cobalt_un = TRUE, cobalt_binary = "std")
# but performs some additional checks

get.cobalt.options("un", "binary")

# Note: the above is equivalent to:
# getOption("cobalt_un")
# getOption("cobalt_binary")

# Return all cobalt options to their defaults

set.cobalt.options(default = TRUE)

# View all available options
get.cobalt.options()
```

---

splitfactor *Split and Unsplit Factors into Dummy Variables*

Description

`splitfactor()` splits factor variables into dummy (0/1) variables. This can be useful when functions do not process factor variables well or require numeric matrices to operate. `unsplitfactor()` combines dummy variables into factor variables, undoing the operation of `splitfactor()`.

Usage

```
splitfactor(data, var.name, replace = TRUE, sep = "_",
            drop.level = NULL, drop.first = TRUE,
            drop.singleton = FALSE,
            drop.na = TRUE, check = TRUE)

unsplitfactor(data, var.name, replace = TRUE, sep = "_",
                dropped.level = NULL, dropped.na = TRUE)
```

Arguments

`data`	A data.frame containing the variables to be split or unsplit. In `splitfactor()`, can be a factor variable to be split.

var.name	For splitfactor(), the names of the factor variables to split. If not specified, will split all factor variables in data. If data is a factor, the stem for each of the new variables to be created. For unsplitfactor(), the name of the previously split factor.
replace	Whether to replace the original variable(s) with the new variable(s) (TRUE) or the append the newly created variable(s) to the end of the data set (FALSE).
sep	A character separating the the stem from the value of the variable for each dummy. For example, for "race_black", sep = "_".
drop.level	The name of a level of var.name for which to drop the dummy variable. Only works if there is only one variable to be split.
drop.first	Whether to drop the first dummy created for each factor. If "if2", will only drop the first category if the factor has exactly two levels. The default is to always drop the first dummy (TRUE).
drop.singleton	Whether to drop a factor variable if it only has one level.
drop.na	If NAs are present in the variable, how to handle them. If TRUE, no new dummy will be created for NA values, but all created dummies will have NA where the original variable was NA. If FALSE, NA will be treated like any other factor level, given its own column, and the other dummies will have a value of 0 where the original variable is NA.
check	Whether to make sure the variables specified in var.name are actually factor (or character) variables. If splitting non-factor (or non-character) variables into dummies, set check = FALSE. If check = FALSE and data is a data frame, an argument to var.name must be specified.
dropped.level	The value of the original factor variable whose dummy was dropped when the variable was split. If left empty and a dummy was dropped, the resulting factor will have the value NA instead of the dropped value. Can only be specified when one factor is to be unsplit.
dropped.na	If TRUE, will assume that NAs in the variables to be unsplit correspond to NA in the unsplit factor (i.e., that drop.na = TRUE was specified in split.factor()). If FALSE, will assume there is a dummy called "var.name_stem_NA" (e.g., "x_NA") that contains 1s where the unsplit factor should be NA (i.e., that drop.na = FALSE was specified in split.factor(). If NAs are stored in a different column with the same stem, e.g., "x_miss", that name (e.g., "miss") can be entered instead.

Details

If there are NAs in the variable to be split, the new variables created by splitfactor() will have NA where the original variable is NA.

Value

For splitfactor(), a data frame containing the original data set with the newly created dummies. For unsplitfactor(). a data frame containing the original data set with the newly created factor variables.

Examples

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

lalonde.split <- splitfactor(lalonde, "race",
                             replace = TRUE,
                             drop.first = TRUE)
# A data set with "race_hispan" and "race_white" instead
# of "race".

lalonde.unsplit <- unsplitfactor(lalonde.split, "race",
                         replace = TRUE,
                         dropped.level = "black")

all.equal(lalonde, lalonde.unsplit) #TRUE
```

---

var.names *Extract Variable Names from* bal.tab *Objects*

Description

This function extracts variable names from a bal.tab object for use in specifying alternate variable names in love.plot. Optionally, a file can be written for easy editing of names.

Usage

```
var.names(b,
          type,
          file = NULL,
          minimal = FALSE)
```

Arguments

b	a bal.tab object; the output of a call to bal.tab.
type	the type of output desired. Can either be "df" for a data.frame or "vec" for a named vector. See "Value". The default is "vec" unless file is not NULL.
file	optional; a file name to save the output if type = "df". See write.csv, which var.name calls. Must end in .csv.
minimal	whether the output should contain all variables names (i.e., all rows that appear the output of bal.tab) or just the unique base variables. See "Details".

Details

The goal of the function is to make supplying new variable names to the var.names argument in love.plot easier. Rather than manually creating a vector or data.frame with all the variable names that one desires to change, one can use var.names to extract variable names from a bak.tab object and edit the output. Importantly, the output can be saved to a CSV file, which can be easily edited and read back into R for use in love.plot, as demonstrated in the Example.

When `minimal = TRUE`, only a minimal set of variables will be output. For example, if the variables analyzed in `bal.tab` are age, race, and married, and `int = TRUE` in `bal.tab`, many variables will appear in the output, including expansions of the factor variables, the polynomial terms, and the interactions. Rather than renaming all of these variables individually, one can rename just the three base variables, and all variables that arise from them will be accordingly renamed. Setting `minimal = TRUE` requests only these base variables.

Value

If `type = "vec"`, a character vector the the variable names both as the names and the entries. If `type = "df"`, a `data.frame` with two columns called `"old"` and `"new"`, each with the variables as the entries. If `file` is not `NULL`, the output will be returned invisibly.

Note

Not all programs can properly read the Unicode characters for the polynomial terms when requested. These may appear strange in, e.g., Excel, but R will process the characters correctly.

Examples

```
data(lalonde)

b1 <- bal.tab(treat ~ age + race + married, data = lalonde,
              int = TRUE)
v1 <- var.names(b1, type = "vec", minimal = TRUE)
v1["age"] <- "Age (Years)"
v1["race"] <- "Race/Eth"
v1["married"] <- "Married"
love.plot(b1, var.names = v1)

## Not run:
b2 <- bal.tab(treat ~ age + race + married + educ + nodegree +
               re74 + re75 + I(re74==0) + I(re75==0),
               data = lalonde)
var.names(b2, file = "varnames.csv")

##Manually edit the CSV (e.g., in Excel), then save it.
##
v2 <- read.csv("varnames.csv")
love.plot(b2, var.names = v2)

## End(Not run)
```

Index