Package 'ebal'

May 8, 2026

Type Package

Title Entropy Reweighting to Create Balanced Samples

Version 0.2.1

Date 2026-04-28

Description Implements entropy balancing, a data preprocessing procedure described in Hainmueller (2012, <doi:10.1093/pan/mpr025>) that allows users to reweight a dataset such that the covariate distributions in the reweighted data satisfy a set of user-specified moment conditions. Useful for creating balanced samples in observational studies with a binary treatment where the control group is reweighted to match the covariate moments of the treatment group, and for reweighting a survey sample to known characteristics from a target population.

License GPL (>= 2)

Imports graphics, methods, stats

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://web.stanford.edu/~jhain/, https://github.com/j-hai/ebal

BugReports https://github.com/j-hai/ebal/issues

Encoding UTF-8

NeedsCompilation no

Author Jens Hainmueller [aut, cre]

Maintainer Jens Hainmueller <jhain@stanford.edu>

Repository CRAN

Date/Publication 2026-04-29 07:30:08 UTC

Contents

`baltest.collect` *Collect Covariate Balance Statistics*

Description

A function that summarizes the covariate balance statistics that are computed by `MatchBalance(Matching)` in a balance table.

Usage

```
baltest.collect(matchbal.out, var.names, after = TRUE)
```

Arguments

`matchbal.out`	An object from a call to `MatchBalance(Matching)`
`var.names`	A vector of covariate names.
`after`	A logical flag for whether the results from before or after Matching should be summarized. If TRUE `baltest.collect` summarizes the results from the covariate balance checks that `MatchBalance` computes in the matched data. If FALSE the results from the balance checks in the unmatched data are used.

Details

See `MatchBalance(Matching)` for details.

Value

A matrix that contains the covariate balance statistics in tabular format.

Author(s)

Jens Hainmueller

See Also

`MatchBalance` in the `Matching` package.

Examples

```
## load(Matching) to run this example
## create toy data: one treatment indicator and three covariates X1-3
#dat <- data.frame(treatment=rbinom(50,size=1,prob=.5),replicate(3,rnorm(50)))
#covarsname <- colnames(dat)[-1]

## run balance checks
#mout <- MatchBalance(treatment~X1+X2+X3,data=dat)

## summarize in balance table
#baltest.collect(matchbal.out=mout,var.names=covarsname,after=FALSE)
```

---

eb *Function for Entropy Balancing*

Description

This function is called internally by `ebalance` and `ebalance.trim` to implement entropy balancing. This function would normally not be called manually by a user.

Usage

```
eb(tr.total = tr.total, co.x = co.x,
   coefs = coefs, base.weight = base.weight,
   max.iterations = max.iterations,
   constraint.tolerance = constraint.tolerance,
   print.level = print.level)
```

Arguments

`tr.total`	NA
`co.x`	NA
`coefs`	NA
`base.weight`	NA
`max.iterations`	NA
`constraint.tolerance`	NA
`print.level`	NA

Value

A list containing the results from the algorithm.

Author(s)

Jens Hainmueller

See Also

ebalance, ebalance.trim

Examples

```
##---- NA -----
```

---

ebalance *Entropy balancing*

Description

This function implements entropy balancing, a data preprocessing procedure that allows users to reweight a dataset. The preprocessing is based on a maximum entropy reweighting scheme that assigns weights to each unit such that the covariate distributions in the reweighted data satisfy a set of moment conditions specified by the researcher. This can be useful to balance covariate distributions in observational studies with a binary treatment where the control group data can be reweighted to match the covariate moments in the treatment group. Entropy balancing can also be used to reweight a survey sample to known characteristics from a target population. The weights that result from entropy balancing can be passed to regression or other models to subsequently analyze the reweighted data.

By default, ebalance reweights the covariate distributions from a control group to match target moments computed from a treatment group such that the reweighted data can be used to analyze the average treatment effect on the treated.

Two interfaces are supported. With Treatment as a numeric or logical vector, supply the covariate matrix X directly. With Treatment as a two-sided formula, supply a data frame; the formula's left-hand side is used as the treatment indicator and the right-hand side as the covariate matrix (the intercept column is dropped automatically).

Usage

```
ebalance(Treatment, X = NULL, base.weight = NULL,
         norm.constant = NULL, coefs = NULL,
         max.iterations = 200, constraint.tolerance = 1,
         print.level = 0, data = NULL, ...)
```

Arguments

Treatment	For the default method: a vector indicating the observations to reweight (controls) and those used to compute target moments (treatment). This can be a logical vector or a numeric vector where 0 denotes control observations and 1 denotes treatment observations. For the formula method: a two-sided formula of the form treat ~ x1 + x2 + ..., with the treatment indicator on the left-hand side and the covariates on the right.

X	A matrix containing the covariates to include in the reweighting. To adjust the means of the covariates, include the raw covariates. To adjust the variances, include squared terms; for co-moments, include interaction terms. All columns must have positive variance and the matrix must be invertible. No missing data is allowed.
data	For the formula method: a data frame containing the variables in Treatment.
base.weight	An optional vector of base weights for the maximum entropy reweighting (one weight per control unit). Default: uniform base weights.
norm.constant	An optional normalizing constant. By default the weights are normalized such that their sum equals the number of treated observations.
coefs	An optional vector of starting coefficients.
max.iterations	Maximum number of iterations.
constraint.tolerance	Tolerance for declaring the moments in the reweighted data equal to the target moments.
print.level	Controls the level of printing: 0 (silent, the default), 1 (normal printing), 2 (detailed), and 3 (very detailed).
...	Additional arguments. For the formula method, passed through to the default method.

Value

A list of class ebalance with the following elements:

target.margins	Target moments computed from the treatment group.
co.xdata	Covariate data from the control group (with intercept column).
w	Control-group weights assigned by entropy balancing (length = number of controls).
coefs	Coefficients from the reweighting algorithm.
maxdiff	Maximum deviation between reweighted moments and targets.
norm.constant	Normalizing constant used.
constraint.tolerance	Tolerance level used for the balance constraints.
max.iterations	Maximum number of iterations used.
base.weight	Base weight used.
print.level	Print level used.
converged	Logical flag indicating convergence within tolerance.
Treatment	The treatment indicator vector as supplied (length = number of observations).
X	The covariate matrix as supplied.

Author(s)

Jens Hainmueller

References

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

Zaslavsky, A. (1988), 'Representing local reweighting area adjustments by of households', Survey Methodology 14(2), 265–288.

Ireland, C. and Kullback, S. (1968), 'Contingency tables with given marginals', Biometrika 55, 179–188.

Kullback, S. (1959), Information Theory and Statistics, Wiley, NY.

See Also

`ebalance.trim` for trimming weights, and the `summary`, `plot`, and `weights` methods for inspection and downstream use.

Examples

```
# Toy observational-study data: treatment is associated with older,
# more educated, higher-income units; the true effect on the outcome
# is 5, but a naive comparison is biased upward by the confounders.
set.seed(42)
n_t <- 75; n_c <- 250
df  <- data.frame(
  treat  = c(rep(1, n_t), rep(0, n_c)),
  age    = c(rnorm(n_t, 45,  8), rnorm(n_c, 38, 10)),
  educ   = c(rnorm(n_t, 16,  2.5), rnorm(n_c, 13, 3)),
  income = c(rnorm(n_t, 65, 12), rnorm(n_c, 50, 15))
)
df$y <- 0.1 * df$age + 0.3 * df$educ + 0.05 * df$income +
        5 * df$treat + rnorm(nrow(df), 0, 3)

# ---- Naive (biased) regression -----------------------------------
coef(lm(y ~ treat, data = df))["treat"]   # ATT estimate; pulled up by confounders

# ---- Entropy balancing: formula interface ------------------------
fit <- ebalance(treat ~ age + educ + income, data = df)
fit                       # one-screen overview via print()
summary(fit)              # balance table: pre/post means and std diffs

# ---- Equivalent matrix interface ---------------------------------
X <- as.matrix(df[, c("age", "educ", "income")])
fit2 <- ebalance(Treatment = df$treat, X = X)
all.equal(fit$w, fit2$w)  # identical results

# ---- Use the weights downstream ----------------------------------
df$w <- weights(fit)              # length = nrow(df); treated get 1
coef(lm(y ~ treat, data = df,     # weighted regression, ATT
        weights = w))["treat"]

# ---- Visualize balance -------------------------------------------
```

```
## Not run:
plot(fit)                          # base-R Love plot, no dependencies

## End(Not run)
```

---

ebalance-methods *Methods for ebalance and ebalance.trim objects*

Description

Convenience methods for inspecting and using objects returned by ebalance and ebalance.trim.

Usage

```
## S3 method for class 'ebalance'
print(x, ...)
## S3 method for class 'ebalance.trim'
print(x, ...)
## S3 method for class 'ebalance'
summary(object, ...)
## S3 method for class 'ebalance.trim'
summary(object, ...)
## S3 method for class 'summary.ebalance'
print(x, digits = 4, ...)
## S3 method for class 'summary.ebalance.trim'
print(x, digits = 4, ...)
## S3 method for class 'ebalance'
plot(x, abs.values = TRUE,
     xlab = NULL, main = NULL, ...)
## S3 method for class 'ebalance.trim'
plot(x, ...)
## S3 method for class 'ebalance'
weights(object, ...)
## S3 method for class 'ebalance.trim'
weights(object, ...)
```

Arguments

x, object	An object of class ebalance or ebalance.trim.
abs.values	Logical. If TRUE (default) the Love plot shows absolute standardized differences; if FALSE signed.
xlab, main	Standard graphical arguments passed to plot().
digits	Number of digits used when printing the summary table.
...	Additional arguments. Currently unused for print, summary, weights; passed to plot() for plot methods.

Details

`print` gives a one-screen overview: counts of treated/control units, number of moments balanced, convergence status, and (for trimmed objects) whether the trim target was met.

`summary` returns a list of class `summary.ebalance` (or `summary.ebalance.trim`) containing a balance table that compares treated and control covariate means before and after weighting along with the corresponding standardized differences.

`plot` produces a Love plot of the standardized differences before and after weighting, one row per covariate.

`weights` returns a length-n numeric vector aligned to the original `Treatment`: treated observations receive weight 1 and control observations receive their entropy-balancing weight. This is suitable for use with `lm(..., weights = w)` and other model fitters that accept case weights.

Value

`print` and the print methods for summary objects return their input invisibly. `summary` returns an object of class `summary.ebalance` or `summary.ebalance.trim` containing `$call.info` and `$balance`. `plot` returns the balance table invisibly. `weights` returns a numeric vector of length equal to the original `Treatment` vector.

See Also

`ebalance`, `ebalance.trim`.

Examples

```
set.seed(1)
df <- data.frame(
  treat = c(rep(1, 30), rep(0, 50)),
  x1    = c(rnorm(30, 0.5), rnorm(50, 0)),
  x2    = c(rnorm(30, 0.5), rnorm(50, 0)),
  x3    = c(rnorm(30, 0.5), rnorm(50, 0))
)

fit <- ebalance(treat ~ x1 + x2 + x3, data = df)

# print(): one-screen overview of the fit
print(fit)

# summary(): pre/post means and standardized differences for each
# covariate; the post-weighting std diffs should be near zero.
summary(fit)

# weights(): length-n vector aligned to the original treatment.
# Treated observations get weight 1; control observations get the
# entropy-balancing weight. Drop-in for lm(..., weights = w).
w <- weights(fit)
length(w) == nrow(df)
all(w[df$treat == 1] == 1)

# Same methods on a trimmed object
```

```
trimmed <- ebalance.trim(fit)
print(trimmed)              # also shows trim.feasible
summary(trimmed)
weights(trimmed)[1:5]

## Not run:
# Love plot of standardized differences before vs. after
plot(fit)
plot(trimmed)

## End(Not run)
```

`ebalance.trim` *Trimming of Weights for Entropy Balancing*

Description

Trim weights obtained from entropy balancing. Takes the output from a call to `ebalance` and trims the weights (subject to the moment conditions) so that the ratio of the maximum (or minimum) weight to the mean weight is reduced to satisfy a user-specified target. If no target is specified, the maximum weight ratio is automatically trimmed as far as is feasible given the data.

Usage

```
ebalance.trim(ebalanceobj, max.weight = NULL,
              min.weight = 0, max.trim.iterations = 200,
              max.weight.increment = 0.92,
              min.weight.increment = 1.08,
              print.level = 0)
```

Arguments

`ebalanceobj`	An object from a call to `ebalance`.
`max.weight`	Optional target for the ratio of the maximum to mean weight.
`min.weight`	Optional target for the ratio of the minimum to mean weight.
`max.trim.iterations`	Maximum number of trimming iterations.
`max.weight.increment`	Increment for iterative trimming of the ratio of the maximum to mean weight (a scalar between 0-1, .92 indicates that the attempted reduction in the max ratio is 8 percent).
`min.weight.increment`	Increment for iterative trimming of the ratio of the minimum to mean weight (a scalar > 1, 1.08 indicates that the attempted reduction in the max ratio is 8 percent).
`print.level`	Controls the level of printing: 0 (silent, the default), 1 (normal printing), 2 (detailed), and 3 (very detailed).

Value

An list object of class `ebalance.trim` with the following elements:

`target.margins`	A vector that contains the target moments coded from the covariate distributions of the treatment group.
`co.xdata`	A matrix that contains the covariate data from the control group.
`w`	A vector that contains the control group weights assigned by trimming entropy balancing algorithm.
`coefs`	A vector that contains coefficients from the reweighting algorithm.
`maxdiff`	A scalar that contains the maximum deviation between the moments of the reweighted data and the target moments.
`norm.constant`	Normalizing constant used.
`constraint.tolerance`	The tolerance level used for the balance constraints.
`max.iterations`	Maximum number of trimming iterations used.
`base.weight`	The base weight used.
`converged`	Logical flag if the inner entropy-balancing algorithm converged within tolerance on the last successful iteration.
`trim.feasible`	Logical flag indicating whether the requested trimming target was achieved. `TRUE` when (a) an explicit `max.weight` (and optional `min.weight`) target was met, or (b) automated minimization mode finished. `FALSE` when an explicit target was not met because the maximum number of iterations was exceeded or the inner solve became numerically singular; in that case a warning is emitted and the most recent feasible fit is returned.

Author(s)

Jens Hainmueller

References

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

Zaslavsky, A. (1988), 'Representing local reweighting area adjustments by of households', Survey Methodology 14(2), 265–288.

Ireland, C. and Kullback, S. (1968), 'Contingency tables with given marginals', Biometrika 55, 179–188.

Kullback, S. (1959), Information Theory and Statistics, Wiley, NY.

See Also

Also see `ebalance`.

Examples

```
# Toy data with substantial covariate imbalance
set.seed(20260427)
n_t <- 50; n_c <- 100
df  <- data.frame(
  treat = c(rep(1, n_t), rep(0, n_c)),
  x1    = c(rnorm(n_t, 0.6), rnorm(n_c, 0)),
  x2    = c(rnorm(n_t, 0.6), rnorm(n_c, 0)),
  x3    = c(rnorm(n_t, 0.6), rnorm(n_c, 0))
)
fit <- ebalance(treat ~ x1 + x2 + x3, data = df)

# ---- Auto-minimization mode ----------------------------------------
# Without a target, ebalance.trim() iteratively reduces the maximum
# weight ratio as far as the data allows. trim.feasible is TRUE by
# definition for auto mode.
trimmed <- ebalance.trim(fit)
trimmed                       # print method shows trim.feasible + max ratio
summary(trimmed)              # balance table for the trimmed weights

# Compare untrimmed vs. trimmed weight distributions
round(summary(fit$w),     2)
round(summary(trimmed$w), 2)

# ---- Explicit max.weight target ------------------------------------
# Pick a target above the natural minimum ratio so it's achievable.
target <- max(fit$w / mean(fit$w)) * 1.5
trimmed2 <- ebalance.trim(fit, max.weight = target)
trimmed2$trim.feasible        # TRUE — target was met

# ---- Infeasible target: graceful fallback (new in 0.2.0) ----------
# Asking for something the data cannot support no longer crashes.
# A warning is emitted and the most recent feasible fit is returned
# with trim.feasible = FALSE.
trimmed3 <- suppressWarnings(ebalance.trim(fit, max.weight = 1.2))
trimmed3$trim.feasible        # FALSE — target was infeasible
max(trimmed3$w) / mean(trimmed3$w)   # the best we could do

# ---- Use the trimmed weights downstream ----------------------------
df$y <- df$treat * 5 + df$x1 + df$x2 + df$x3 + rnorm(nrow(df))
df$w <- weights(trimmed)      # length = nrow(df), treated = 1
coef(lm(y ~ treat, data = df, weights = w))["treat"]
```

`getsquares` *Generate Matrix of Squared Terms*

Description

Takes a matrix of covariates and generates a new matrix that contains the original covariates and all squared terms. Squared terms for binary covariates are omitted.

Usage

```
getsquares(mat)
```

Arguments

mat	n by k numeric matrix of covariates.

Value

n by k*2 numeric matrix that contains the original covariates plus all squared terms.

Author(s)

Jens Hainmueller

See Also

See matrixmaker

Examples

```
# create toy matrix
mold <- replicate(3,rnorm(50))
colnames(mold) <- paste("x",1:3,sep="")
head(mold)
# create new matrix
mnew <- getsquares(mold)
head(mnew)
```

---

line.searcher *Optimal step length search for entropy balancing algorithm*

Description

Function called internally by ebalance and ebalance.trim to compute optimal step length for entropy balancing algorithm. This function would normally not be called manually by a user.

Usage

```
line.searcher(Base.weight, Co.x,
Tr.total, coefs, Newton, ss)
```

Arguments

Base.weight	NA
Co.x	NA
Tr.total	NA
coefs	NA
Newton	NA
ss	NA

Value

A list with the results from the search.

Author(s)

Jens Hainmueller

See Also

ebalance, ebalance.trim

Examples

```
##---- NA -----
```

---

matrixmaker *Generate Matrix of One-way Interactions and Squared Terms*

Description

Takes a matrix of covariates and generates a new matrix that contains the original covariates, all one-way interaction terms, and all squared terms.

Usage

```
matrixmaker(mat)
```

Arguments

mat	n by k numeric matrix of covariates.

Value

n by (k*(k+1))/2 +1) matrix of covariates with original covariates, all one-way interaction terms, and all squared terms.

Author(s)

Jens Hainmueller

See Also

See getsquares

Examples

```
# create toy matrix
mold <- replicate(3,rnorm(50))
colnames(mold) <- paste("x",1:3,sep="")
head(mold)
# create new matrix
mnew <- matrixmaker(mold)
head(mnew)
```

Index