Package 'kbal'

February 23, 2026

Type Package

Title Kernel Balancing

Version 0.1.4

Date 2026-02-23

Description Provides a weighting approach that employs kernels to make one group have a similar distribution to another group on covariates. This method matches not only means or marginal distributions but also higher-order transformations implied by the choice of kernel. 'kbal' is applicable to both treatment effect estimation and survey reweighting problems. Based on Hazlett, C. (2020) ``Kernel Balancing: A flexible non-parametric weighting procedure for estimating causal effects." Statistica Sinica. <https://www.researchgate.net/publication/299013953_Kernel_Balancing_A_flexible_non-parametric_weighting_procedure_for_estimating_causal_effects>.

URL https://github.com/chadhazlett/kbal

License GPL (>= 2)

LazyData TRUE

LazyLoad yes

Depends R (>= 3.5.0)

Imports Rcpp (>= 0.11.0), RcppParallel (>= 4.4.4), dplyr, RSpectra

LinkingTo Rcpp, RcppParallel

Maintainer Borna Bateni <borna@ucla.edu>

RoxygenNote 7.2.3

Encoding UTF-8

NeedsCompilation yes

Author Chad Hazlett [aut, cph],
Ciara Sterbenz [aut],
Erin Hartman [ctb],
Alex Kravetz [ctb],
Borna Bateni [aut, cre]

Repository CRAN

Date/Publication 2026-02-23 09:20:08 UTC

Contents

`biasbound` *Worst-Case Bias Bound due to Incomplete Balance*

Description

Calculate the upper bound on the bias induced by approximate balance with a given `hilbertnorm`. Approximate balance is conducted in `kbal()` and uses only the first `numdims` dimensions of the singular value decomposition of the kernel matrix to generate weights `w` which produce mean balance between control or sampled units and treated or population units. The following function calculates the worse-case bias induced by this approximate balancing with weights `w` and a given `hilbertnorm`.

Usage

```
biasbound(observed, target, svd.out, w, w.pop = NULL, hilbertnorm = 1)
```

Arguments

`observed`	a numeric vector of length equal to the total number of units where sampled/control units take a value of 1 and population/treated units take a value of 0.
`target`	a numeric vector of length equal to the total number of units where population/treated units take a value of 1 and sample/control units take a value of 0.
`svd.out`	the list object output from `svd()` performed on the kernel matrix. Requires a list object with left singular vectors in `svd.out$u` and singular values in `svd.out$d`
`w`	numeric vector containing the weight for every corresponding unit. Note that these weights should sum to the total number of units, not to one. They are divided by the number of control or sample and treated or population units internally.
`w.pop`	an optional vector input to specify population weights. Must be of length equal to the total number of units (rows in `svd.out`) with all sampled units receiving a weight of 1. The sum of the weights for population units must be either 1 or the number of population units.

`hilbertnorm` numeric value of the Hilbert norm. Default is 1.

Value

`biasbound` value of worst-case bias bound due to incomplete balance with inputted weights

Examples

```
#load and clean data a bit
set.seed(123)
data("lalonde")
# Select a random subset of 500 rows
lalonde_sample <- sample(1:nrow(lalonde), 500, replace = FALSE)
lalonde <- lalonde[lalonde_sample, ]

xvars=c("age","black","educ","hisp","married","re74","re75","nodegr","u74","u75")

#need a kernel matrix to run SVD on and pass in so get that first with makeK
#running makeK with the sampled units as the bases
K = makeK(allx = lalonde[,xvars], useasbases = 1-lalonde$nsw)

#svd on this kernel
svd_pass = svd(K)
#let's use the original weights of 1/number of sampled units, and 1/number of target units
#this is the default if we pass in w as all 1's
biasbound(observed=(1-lalonde$nsw),
          target=lalonde$nsw,
          svd.out = svd_pass,
          w = rep(1,nrow(lalonde)), hilbertnorm=1)
```

---

b_maxvarK *Maximum Variance of Gaussian Kernel Matrix*

Description

Searches for the argmax of the variance of the Kernel matrix.

Usage

```
b_maxvarK(data, useasbases, cat_data = TRUE, maxsearch_b = 2000)
```

Arguments

`data` a matrix of data where rows are all units and columns are covariates. Where all covariates are categorical, this matrix should be one-hot encoded (refer to `one_hot` to produce) with `cat_data` argument true.

useasbases	binary vector specifying what observations are to be used in forming bases (columns) of the kernel matrix. Suggested default is: if the number of observations is under 4000, use all observations; when the number of observations is over 4000, use the sampled (control) units only.
cat_data	logical for whether kernel contains only categorical data or not. Default is TRUE.
maxsearch_b	the maximum value of b, the denominator of the Gaussian, searched during maximization. Default is 2000.

Value

b_maxvar	numeric b value, the denominator of the Gaussian, which produces the maximum variance of K kernel matrix
var_K	numeric maximum variance of K kernel matrix found with b as b_maxvar

Examples

```
#lalonde with only categorical data
set.seed(123)
data("lalonde")
# Select a random subset of 500 rows
lalonde_sample <- sample(1:nrow(lalonde), 500, replace = FALSE)
lalonde <- lalonde[lalonde_sample, ]

cat_vars <- c("black","hisp","married","nodegr","u74","u75")
#Convert to one-hot encoded data matrix:
onehot_lalonde = one_hot(lalonde[, cat_vars])
colnames(onehot_lalonde)
best_b <- b_maxvarK(data = onehot_lalonde,
                    useasbases = 1-lalonde$nsw)
```

---

dimw *Difference in Means and Difference in Weighted Means*

Description

Calculates the simple difference in means or weighted difference in means between the control or sample population and the treated or target population.

Usage

```
dimw(X, w, target)
```

Arguments

X	matrix of data where rows are observations and columns are covariates.
w	numeric vector of weights for each observation.
target	numeric vector of length equal to the total number of units where population/treated units take a value of 1 and sample/control units take a value of 0.

Value

dim	the simple, unweighted difference in means.
dimw	the weighted difference in means.

Examples

```
#let's say we want to get the unweighted DIM and the weighted DIM using weights from the kbal
#function with the lalonde data:
#load and clean data a bit
set.seed(123)
data("lalonde")
# Select a random subset of 500 rows
lalonde_sample <- sample(1:nrow(lalonde), 500, replace = FALSE)
lalonde <- lalonde[lalonde_sample, ]

xvars=c("age","black","educ","hisp","married","re74","re75","nodegr","u74","u75")

#get the kbal weights
kbalout= kbal(allx=lalonde[,xvars],
               sampledinpop=FALSE,
               treatment=lalonde$nsw)
 #now use dimw to get the DIMs
 dimw(X = lalonde[,xvars], w = kbalout$w, target = lalonde$nsw)
```

---

drop_multicollin *Drop Multicollinear Columns*

Description

Drops multicollinear columns in order of highest correlation using the correlation matrix. This function uses the cor function from the stats package to calculate the correlations between columns.

Usage

```
drop_multicollin(allx, printprogress = TRUE)
```

Arguments

allx	a matrix of data to check for multicollinearity. All columns must be numeric.
printprogress	logical to indicate if progress should be printed out to the command line. Default is TRUE.

Value

A list containing:

allx_noMC	resulting data matrix of full rank after multicollinear columns have been dropped.
dropped_cols	column names of the dropped columns.

Examples

```
# Create data with multicollinearity
data <- data.frame(x = rnorm(100),
                   y = sample.int(100, 100),
                   z = runif(100, 3, 6))
test = data.frame(mc_1 = data$x,
                  mc_2 = data$x * 2 + data$y - data$z)
dat = cbind(test, data)
# Run function
mc_check = drop_multicollin(dat)
mc_check$dropped_cols
```

`ebalance_custom` *Modified version of ebalance (originally from Jens Hainmueller)*

Description

This is a custom version of the `ebal` (entropy balancing) package by Jens Hainmueller. Chooses weights on controls to make covariate means equal to those of treated. This version differs from `ebal` only in that it handles cases where there is only a single unit, which otherwise causes a problem in the original code.

Usage

```
ebalance_custom(
  Treatment,
  X,
  base.weight = NULL,
  norm.constant = NULL,
  coefs = NULL,
  max.iterations = 200,
  constraint.tolerance = 0.001,
  print.level = 0
)
```

Arguments

`Treatment`	a numeric vector of length equal to the total number of units where treated (population) units take a value of 1 and control (sampled) units take a value of 0.
`X`	matrix of data where rows are observations and columns are covariates.
`base.weight`	an optional numeric vector argument of length equal to the total number of control units to specify the base weight of each control unit within entropy balancing. Default is even weights (1) for all controls.
`norm.constant`	an optional numeric argument; users should leave unspecified in most cases.

coefs	an optional vector argument of length equal to one more than the number of covariates in X; users should leave unspecified in most cases.
max.iterations	numeric maximum number of iterations to use when searching for weights
constraint.tolerance	numeric tolerance level.
print.level	a numeric argument to specify the amount of information printed out. 0 is silent, 1 prints convergence status, 2 prints maximum deviance per iteration, 3 prints loss and step length.

Value

target.margins	Column sums of X among the treated units.
co.xdata	Covariate matrix for the controls only built from X with an additional appended column of ones.
w	weights found using ebalance. Note that treated units all receive flat weights of 1
maxdiff	absolute value of the largest component of the gradient in the last iteration.
norm.constant	norm constant used
constraint.tolerance	tolerance used to evaluate convergence
max.iterations	max iterations used
base.weight	base weights used
print.level	print level used
converged	Convergence status. If ebalance failed to find weights within the specified constraint.tolerance after max.iterations this is FALSE. Note that even if ebalance does not converge, the last iteration's weights w are returned.

---

getdist *L1 Distance*

Description

Calculates the L1 distance between the treated or population units and the kernel balanced control or sampled units.

Usage

```
getdist(
  target,
  observed,
  K,
  w.pop = NULL,
  w = NULL,
  numdims = NULL,
```

```
  base.weights = NULL,
  ebal.tol = 1e-06,
  ebal.maxit = 500,
  svd.U = NULL
)
```

Arguments

target	a numeric vector of length equal to the total number of units where population/treated units take a value of 1 and sample/control units take a value of 0.
observed	a numeric vector of length equal to the total number of units where sampled/control units take a value of 1 and population/treated units take a value of 0.
K	the kernel matrix
w.pop	an optional vector input to specify population weights. Must be of length equal to the total number of units (rows in svd.U) with all sampled units receiving a weight of 1. The sum of the weights for population units must be either 1 or the number of population units.
w	a optional numeric vector of weights for every observation. Note that these weights should sum to the total number of units, where treated or population units have a weight of 1 and control or sample units have appropriate weights derived from kernel balancing with mean 1, is consistent with the output of getw(). If unspecified, these weights are found internally using numdims dimensions of the SVD of the kernel matrix svd.U with ebalance_custom().
numdims	an optional numeric input specifying the number of columns of the singular value decomposition of the kernel matrix to use when finding weights when w is not specified.
base.weights	optional positive vector of base/design weights. These are only used for sample units (observed==1 & target==0); all other units are treated as having base weight 1 inside entropy balancing.
ebal.tol	an optional numeric input specifying the tolerance level used by custom entropy balancing function ebalance_custom() in the case that w is not specified. Default is 1e-6.
ebal.maxit	maximum number of iterations in optimization search used by ebalance_custom when w is not specified. Default is 500.
svd.U	an optional matrix of left singular vectors from performing svd() on the kernel matrix in the case that w is unspecified. If unspecified when w also not specified, internally computes the svd of K.

Value

L1	a numeric giving the L1 distance, the absolute difference between pX_D1 and pX_D0w
w	numeric vector of weights used
pX_D1	a numeric vector of length equal to the total number of observations where the nth entry is the sum of the kernel distances from the nth unit to every treated or population unit. If population units are specified, this sum is weighted by w.pop accordingly.

pX_D0 — a numeric vector of length equal to the total number of observations where the nth entry is the sum of the kernel distances from the nth unit to every control or sampled unit.

pX_D0w — a numeric vector of length equal to the total number of observations where the nth entry is the weighted sum of the kernel distances from the nth unit to every control or sampled unit. The weights are given by entropy balancing and produce mean balance on $\phi(X)$, the expanded features of X using a given kernel $\phi(.)$, for the control or sample group and treated group or target population.

Examples

```
#loading and cleaning lalonde data
set.seed(123)
data("lalonde")
# Select a random subset of 500 rows
lalonde_sample <- sample(1:nrow(lalonde), 500, replace = FALSE)
lalonde <- lalonde[lalonde_sample, ]

xvars=c("age","black","educ","hisp","married","re74","re75","nodegr","u74","u75")

#need to first build gaussian kernel matrix
K_pass <- makeK(allx = lalonde[,xvars])
#also need the SVD of this matrix
svd_pass <- svd(K_pass)

#running without passing weights in directly, using numdims=33
l1_lalonde <- getdist(target = lalonde$nsw,
                      observed = 1-lalonde$nsw,
                      K = K_pass,
                      svd.U = svd_pass$u,
                      numdims = 33)

 #alternatively, we can get the weights ourselves and pass them in directly
 #using the first 33 dims of svd_pass$u to match the above
w_opt <- getw(target= lalonde$nsw,
              observed = 1-lalonde$nsw,
              svd.U = svd_pass$u[,1:33])$w
l1_lalonde2 <- getdist(target = lalonde$nsw,
                 observed = 1-lalonde$nsw,
                 K = K_pass,
                 w = w_opt)
```

---

getw *Find Weights using Entropy Balancing.*

Description

Uses entropy balancing to find and return the weights that produce mean balance on $\phi(X_i)$, the expanded features of X_i using a given kernel $\phi(.)$, for the control or sample group and treated group or target population.

Usage

```
getw(
  target,
  observed,
  svd.U,
  base.weights = NULL,
  ebal.tol = 1e-06,
  ebal.maxit = 500
)
```

Arguments

target	binary length-N vector: 1 = target/population, 0 = not in target.
observed	binary length-N vector: 1 = observed/sample, 0 = not observed.
svd.U	a matrix of left singular vectors from performing svd() on the kernel matrix.
base.weights	optional positive length-N vector of base/design weights. These are only used for sample units (observed==1 & target==0); all other units are treated as having base weight 1 inside entropy balancing.
ebal.tol	tolerance level used by custom entropy balancing function ebalance_custom. Default is 1e-6.
ebal.maxit	maximum number of iterations in optimization search used by ebalance_custom. Default is 500.

Value

A list containing:

w	A numeric vector of weights.
converged	boolean indicating if ebalance_custom converged
ebal_error	returns error message if ebalance_custom encounters an error

Examples

```
#load and clean data
set.seed(123)
data("lalonde")
# Select a random subset of 500 rows
lalonde_sample <- sample(1:nrow(lalonde), 500, replace = FALSE)
lalonde <- lalonde[lalonde_sample, ]

xvars=c("age","black","educ","hisp","married","re74","re75","nodegr","u74","u75")

#need a kernel matrix to run SVD on then find weights with; so get that first with makeK.
#running makeK with the sampled units as the bases
K = makeK(allx = lalonde[,xvars], useasbases = 1-lalonde$nsw)

#SVD on this kernel and get matrix with left singular values
U = svd(K)$u
```

```
#Use the first 10 dimensions of U.
U2=U[,1:10]
getw.out=getw(target=lalonde$nsw,
              observed=1-lalonde$nsw,
              svd.U=U2)
```

---

kbal *Kernel Balancing*

Description

Kernel balancing (kbal) is non-parametric weighting tool to make two groups have a similar distribution of covariates, not only in terms of means or marginal distributions but also on (i) general smooth functions of the covariates, including on (ii) a smoothing estimator of the joint distribution of the covariates. It was originally designed (Hazlett, 2017) to make control and treated groups look alike, as desired when estimating causal effects under conditional ignorability. This package also facilitates use of this approach for more general distribution-alignment tasks, such as making a sampled group have a similar distribution of covariates as a target population, as in survey reweighting. The examples below provide an introduction to both settings.

To proceed in the causal effect setting, kbal assumes that the expectation of the non-treatment potential outcome conditional on the covariates falls in a large, flexible space of functions associated with a kernel. It then constructs linear bases for this function space and achieves approximate balance on these bases. The approximation is one that minimizes the worst-case bias that could persist due to remaining imbalances.

The kbal function implements kernel balancing using a gaussian kernel to expand the features of X_i to infinite dimensions. It finds approximate mean balance for the control or sample group and treated group or target population in this expanded feature space by using the first numdims dimensions of the singular value decomposition of the gaussian kernel matrix. It employs entropy balancing to find the weights for each unit which produce this approximate balance. When numdims is not user-specified, it searches through increasing dimensions of the SVD of the kernel matrix to find the number of dimensions which produce weights that minimizes the worst-case bias bound with a given hilbertnorm. It then returns these optimal weights, along with the minimized bias, the kernel matrix, a record of the number of dimensions used and the corresponding bias, as well as an original bias using naive group size weights for comparison. Note that while kernel balancing goes far beyond simple mean balancing, it may not result in perfect mean balance. Users who wish to require mean balancing can specify meanfirst = T to require mean balance on as many dimensions of the data as optimally feasible. Alternatively, users can manually specify constraint to append additional vector constraints to the kernel matrix in the bias bound optimization, requiring mean balance on these columns. Note further that kbal supports three types of input data: fully categorical, fully continuous, or mixed. When data is only categorical, as is common with demographic variables for survey reweighting, users should use argument cat_data = TRUE and can input their data as factors, numeric, or characters and kbal will internally transform the data to a more appropriate one-hot encoding and search for the value of b, the denominator of the exponent in the Gaussian, which maximizes the variance of the kernel matrix. When data is fully continuous, users should use default settings (cat_data = FALSE and cont_data = FAlSE, which will scale all columns and again conduct an internal search for the value of b which maximizes the variance

of K. Note that with continuous data, this search may take considerably more computational time than the categorical case. When data is a mix of continuous and categorical data, users should use argument mixed_data = TRUE, specify by name what columns are categorical with cat_columns, and also set the scaling of the continuous variables with cont_scale. This will result in a one-hot encoding of categorical columns concatenated with the continuous columns scaled in accordance with cont_scale and again an internal search for the value of b which maximizes the variance in the kernel matrix. Again note that compared to the categorical case, this search will take more computational time.

Usage

```
kbal(
  allx,
  useasbases = NULL,
  b = NULL,
  sampled = NULL,
  sampledinpop = NULL,
  treatment = NULL,
  population.w = NULL,
  base.weights = NULL,
  K = NULL,
  K.svd = NULL,
  cat_data = FALSE,
  mixed_data = FALSE,
  cat_columns = NULL,
  cont_scale = NULL,
  scale_data = NULL,
  drop_MC = NULL,
  linkernel = FALSE,
  meanfirst = FALSE,
  mf_columns = NULL,
  constraint = NULL,
  scale_constraint = TRUE,
  numdims = NULL,
  minnumdims = NULL,
  maxnumdims = NULL,
  fullSVD = FALSE,
  incrementby = 1,
  ebal.maxit = 500,
  ebal.tol = 1e-06,
  ebal.convergence = NULL,
  maxsearch_b = 2000,
  early.stopping = TRUE,
  printprogress = TRUE
)
```

Arguments

allx	a data matrix containing all observations where rows are units and columns are covariates. When using only continuous covariates (cat_data = F and mixed_data = F), all columns must be numeric. When using categorical data (either cat_data = T or mixed_data = T), categorical columns can be characters or numerics which will be treated as factors. Users should one-hot encoded categorical covariates as this transformation occurs internally.
useasbases	optional binary vector to specify what observations are to be used in forming bases (columns) of the kernel matrix to get balance on. If the number of observations is under 4000, the default is to use all observations. When the number of observations is over 4000, the default is to use the sampled (control) units only.
b	scaling factor in the calculation of Gaussian kernel distance equivalent to the entire denominator $2\sigma^2$ of the exponent. Default is to search for the value which maximizes the variance of the kernel matrix.
sampled	a numeric vector of length equal to the total number of units where sampled units take a value of 1 and population units take a value of 0.
sampledinpop	a logical to be used in combination with input sampled that, when TRUE, indicates that sampled units should also be included in the target population when searching for optimal weights.
treatment	an alternative input to sampled and sampledinpop that is a numeric vector of length equal to the total number of units. Current version supports the ATT estimand. Accordingly, the treated units are the target population, and the control are equivalent to the sampled. Weights play the role of making the control groups (sampled) look like the target population (treated). When specified, sampledinpop is forced to be FALSE.
population.w	optional vector of population weights length equal to the number of population units. Must sum to either 1 or the number of population units.
base.weights	optional positive length-N vector of base/design weights. These are only used for sampled/control units (e.g., sampled==1 or observed==1 & target==0, depending on which interface is used); all other units are treated as having base weight 1 internally.
K	optional matrix input that takes a user-specified kernel matrix and performs SVD on it internally in the search for weights which minimize the bias bound.
K.svd	optional list input that takes a user-specified singular value decomposition of the kernel matrix. This list must include three objects K.svd$u, a matrix of left-singular vectors, K.svd$v, a matrix of right-singular vectors, and their corresponding singular values K.svd$d.
cat_data	logical argument that when true indicates allx contains only categorical data. When true, the internal construction of the kernel matrix uses a one-hot encoding of allx (multiplied by a factor of $\sqrt{0.5}$ to compensate for double counting) and the value of b which maximizes the variance of this kernel matrix. When true, mixed_data, scale_data, linkernel, and drop_MC should be FALSE. Default is FALSE.
mixed_data	logical argument that when true indicates allx contains a combination of both continuous and categorical data. When true, the internal construction of the

kernel matrix uses a one-hot encoding of the categorical variables in allx as specified by cat_columns (multiplied by a factor of $\sqrt{0.5}$ to compensate for double counting) concatenated with the remaining continuous variables scaled to have default standard deviation of 1 or that specified in cont_scale. When both cat_data and cat_data are FALSE, the kernel matrix assumes all continuous data, does not one-hot encode any part of allx but still uses the value of b which produces maximal variance in K. Default is FALSE.

cat_columns — optional character argument that must be specified when mixed_data is TRUE and that indicates what columns of allx contain categorical variables.

cont_scale — optional numeric argument used when mixed_data is TRUE which specifies how to scale the standard deviation of continuous variables in allx. Can be either a a single value or a vector with length equal to the number of continuous variables in allx (columns not specified in cat_columns) and ordered accordingly.

scale_data — logical when true scales the columns of allx (demeans and scales variance to 1) before building the kernel matrix internally. This is appropriate when allx contains only continuous variables with different scales, but is not recommended when allx contains any categorical data. Default is TRUE when both cat_data and mixed_data are FALSE and FALSE otherwise.

drop_MC — logical for whether or not to drop multicollinear columns in allx before building K. When either cat_data or mixed_data is TRUE, forced to be FALSE. Otherwise, with continuous data only, default is TRUE.

linkernel — logical if true, uses the linear kernel $K = XX'$ which achieves balance on the first moments of X (mean balance). Note that for computational ease, the code employs $K = X$ and adjusts singular values accordingly. Default is FALSE.

meanfirst — logical if true, internally searches for the optimal number of dimensions of the svd of allx to append to K as additional constraints. This will produce mean balance on as many dimensions of allx as optimally feasible with specified ebalance convergence and a minimal bias bound on the remaining unbalances columns of the left singular vectors of K. Note that any scaling specified on allx will be also be applied in the meanfirst routine. Default is FALSE.

mf_columns — either character or numeric vector to specify what columns of allx to perform meanfirst with. If left unspecified, all columns will be used.

constraint — optional matrix argument of additional constraints which are appended to the front of the left singular vectors of K. When specified, the code conducts a constrained optimization requiring mean balance on the columns of this matrix throughout the search for the minimum bias bound over the dimensions of the left singular vectors of K.

scale_constraint — logical for whether constraints in constraint should be scaled before they are appended to the svd of K. Default is TRUE.

numdims — optional numeric argument specifying the number of dimensions of the left singular vectors of the kernel matrix to find balance bypassing the optimization search for the number of dimensions which minimize the biasbound.

minnumdims — numeric argument to specify the minimum number of the left singular vectors of the kernel matrix to seek balance on in the search for the number of dimensions which minimize the bias. Default minimum is 1.

`maxnumdims` numeric argument to specify the maximum number of the left singular vectors of the kernel matrix to seek balance on in the search for the number of dimensions which minimize the bias. For a Gaussian kernel, the default is the minimum between 500 and the number of bases given by `useasbases`. With a linear kernel, the default is the minimum between 500 and the number of columns in `allx`.

`fullSVD` logical argument for whether the full SVD should be conducted internally. When `FALSE`, the code uses truncated svd methods from the `Rspectra` package in the interest of improving run time. When `FALSE`, the code computes only the SVD up to the either 80 percent of the columns of `K` or `maxnumdims` singular vectors, whichever is larger. When the number of columns is less than 80 percent the number of rows, defaults to full svd. Default is `FALSE`.

`incrementby` numeric argument to specify the number of dimensions to increase by from `minnumdims` to `maxnumdims` in each iteration of the search for the number of dimensions which minimizes the bias. Default is 1.

`ebal.maxit` maximum number of iterations used by `ebalance_custom()` in optimization in the search for weights `w`. Default is `500`.

`ebal.tol` tolerance level used by `ebalance_custom()`. Default is `1e-6`.

`ebal.convergence` logical to require ebalance convergence when selecting the optimal `numdims` dimensions of `K` that minimize the biasbound. When constraints are appended to the left singular vectors of `K` via `meanfirst=TRUE` or `constraints`, forced to be `TRUE` and otherwise `FALSE`.

`maxsearch_b` optional argument to specify the maximum b in search for maximum variance of `K` in `b_maxvarK()`. Default is `2000`.

`early.stopping` logical argument indicating whether bias balance optimization should stop twenty rounds after finding a minimum. Default is `TRUE`.

`printprogress` logical argument to print updates throughout. Default is `TRUE`.

Value

`w` a vector of the weights found using entropy balancing on `numdims` dimensions of the SVD of the kernel matrix.

`biasbound_opt` a numeric giving the minimal bias bound found using `numdims` as the number of dimensions of the SVD of the kernel matrix. When `numdims` is user-specified, the bias bound using this number of dimensions of the kernel matrix.

`biasbound_orig` a numeric giving the bias bound found when all sampled (control) units have a weight equal to one over the number of sampled (control) units and all target units have a weight equal to one over the number of target units.

`biasbound_ratio` a numeric giving the ratio of `biasbound_orig` to`biasbound_opt`. Can be informative when comparing the performance of different `b` values.

`dist_record` a matrix recording the bias bound corresponding to balance on increasing dimensions of the SVD of the kernel matrix starting from `minnumdims` increasing by `incrementby` to `maxnumdims` or until the bias grows to be 1.25 times the minimal bias found.

numdims	a numeric giving the optimal number of dimensions of the SVD of the kernel matrix which minimizes the bias bound.
L1_orig	a numeric giving the L1 distance found when all sampled (control) units have a weight equal to one over the number of sampled (control) units and all target units have a weight equal to one over the number of target units.
L1_opt	a numeric giving the L1 distance at the minimum bias bound found using numdims as the number of dimensions of the SVD of the kernel matrix. When numdims is user-specified, the L1 distance using this number of dimensions of the kernel matrix.
K	the kernel matrix
onehot_dat	when categorical data is specified, the resulting one-hot encoded categorical data used in the construction of K. When mixed data is specified, returns concatenated one-hot encoded categorical data and scaled continuous data used to construct K.
linkernel	logical for whether linear kernel was used
svdK	a list giving the SVD of the kernel matrix with left singular vectors svdK$u, right singular vectors svdK$v, and singular values svdK$d
b	numeric scaling factor used in the the calculation of gaussian kernel equivalent to the denominator $2\sigma^2$ of the exponent.
maxvar_K	returns the resulting variance of the kernel matrix when the b determined internally as the argmax of the variance K
bases	numeric vector indicating what bases (rows in allx) were used to construct kernel matrix (columns of K)
truncatedSVD.var	when truncated SVD methods are used on symmetric kernel matrices, a numeric which gives the proportion of the total variance of K captured by the first maxnumdims singular values found by the truncated SVD. When the kernel matrix is non-symmetric, this is a worst case approximation of the percent variance explained, assuming the remaining unknown singular values are the same magnitude as the last calculated in the truncated SVD.
dropped_covariates	provides a vector of character column names for covariates dropped due to multicollinearity.
meanfirst_dims	when meanfirst=TRUE the optimal number of the singular vectors of allx selected and appended to the front of the left singular vectors of K
meanfirst_cols	when meanfirst=TRUE meanfirst_dims first left singular vectors of allx selected that are appended to the front of the left singular vectors of K and balanced on
ebal_error	when ebalance is unable to find convergent weights, the associated error message it reports
base.weights	The base/design weights supplied via base.weights.

References

Hazlett, C. (2017), "Kernel Balancing: A flexible non-parametric weighting procedure for estimating causal effects." Forthcoming in Statistica Sinica. https://doi.org/10.5705/ss.202017.0555

Examples

```
#-------------------------------------------------------------------
# Example 1: Reweight a control group to a treated to estimate ATT.
# Benchmark using Lalonde et al.
#-------------------------------------------------------------------
#1. Rerun Lalonde example with settings as in Hazlett, C (2017). Statistica Sinica paper:
set.seed(123)
data("lalonde")
# Select a random subset of 500 rows
lalonde_sample <- sample(1:nrow(lalonde), 500, replace = FALSE)
lalonde <- lalonde[lalonde_sample, ]

xvars=c("age","black","educ","hisp","married","re74","re75","nodegr","u74","u75")


kbalout.full= kbal(allx=lalonde[,xvars],
                   b=length(xvars),
                   treatment=lalonde$nsw,
                   fullSVD = TRUE)
summary(lm(re78~nsw,w=kbalout.full$w, data = lalonde))


 #2. Lalonde with categorical data only: u74, u75, nodegree, race, married
 cat_vars=c("race_ethnicity","married","nodegr","u74","u75")

 kbalout_cat_only = kbal(allx=lalonde[,cat_vars],
                         cat_data = TRUE,
                         treatment=lalonde$nsw,
                         fullSVD = TRUE)
 kbalout_cat_only$b
 summary(lm(re78~nsw,w=kbalout_cat_only$w, data = lalonde))


 #3. Lalonde with mixed categorical and continuous data
 cat_vars=c("race_ethnicity", "married")
 all_vars= c("age","educ","re74","re75","married", "race_ethnicity")

 kbalout_mixed = kbal(allx=lalonde[,all_vars],
                      mixed_data = TRUE,
                      cat_columns = cat_vars,
                      treatment=lalonde$nsw,
                      fullSVD = TRUE)
 kbalout_mixed$b
 summary(lm(re78~nsw,w=kbalout_mixed$w, data = lalonde))


#-------------------------------------------------------------------
# Example 1B: Reweight a control group to a treated to esimate ATT.
# Benchmark using Lalonde et al. -- but just mean balancing now
# via "linkernel".
#-------------------------------------------------------------------
```

```
# Rerun Lalonde example with settings as in Hazlett, C (2017). Statistica paper:
kbalout.lin= kbal(allx=lalonde[,xvars],
                  b=length(xvars),
                  treatment=lalonde$nsw,
                  linkernel=TRUE,
                  fullSVD=TRUE)

# Check balance with and without these weights:
dimw(X=lalonde[,xvars], w=kbalout.lin$w, target=lalonde$nsw)

summary(lm(re78~nsw,w=kbalout.lin$w, data = lalonde))

#-----------------------------------------------------------------
# Example 2: Reweight a sample to a target population.
#-----------------------------------------------------------------
# Suppose a population consists of four groups in equal shares:
# white republican, non-white republican, white non-republicans,
# and non-white non-republicans. A given policy happens to be supported
# by all white republicans, and nobody else. Thus the mean level of
# support in the population should be 25%.
#
# Further, the sample is surveyed in such a way that was careful
# to quota on party and race, obtaining 50% republican and 50% white.
# However, among republicans three-quarters are white and among non-republicans,
# three quarters are non-white. This biases the average level of support
# despite having a sample that matches the population on its marginal distributions. #'
# We'd like to reweight the sample so it resembles the population not
# just on the margins, but in the joint distribution of characteristics.

pop <- data.frame(
republican =  c(rep(0,400), rep(1,400)),
white = c(rep(1,200), rep(0,200), rep(1,200), rep(0,200)),
support = c(rep(1,200), rep(0,600)))

mean(pop$support)  # Target value

# Survey sample: correct margins/means, but wrong joint distribution
samp <- data.frame( republican = c(rep(1, 40), rep(0,40)),
   white = c(rep(1,30), rep(0,10), rep(1,10), rep(0,30)),
   support = c(rep(1,30), rep(0,50)))

mean(samp$support)  # Appears that support is 37.5% instead of 25%.

# Mean Balancing ----------------------------------------
# Sample is already mean-balanced to the population on each
# characteristic. However for illustrative purposes, use ebal()
dat <- rbind(pop,samp)

# Indicate which units are sampled (1) and which are population units(0)
sampled <- c(rep(0,800), rep(1,80))

# Run ebal (treatment = population units = 1-sampled)
ebal_out <- ebalance_custom(Treatment = 1-sampled,
```

```
                            X=dat[,1:2],
                            constraint.tolerance=1e-6,
                            print.level=-1)

# We can see everything gets even weights, since already mean balanced.
length(unique(ebal_out$w))

# And we end up with the same estimate we started with
weighted.mean(samp[,3], w = ebal_out$w)

# We see that, because the margins are correct, all weights are equal
unique(cbind(samp, e_bal_weight = ebal_out$w))

# Kernel balancing for weighting to a population (i.e. kpop) -------
kbalout = kbal(allx=dat[,1:2],
                useasbases=rep(1,nrow(dat)),
                sampled = sampled,
                b = 1,
                sampledinpop = FALSE)

# The weights now vary:
plot(kbalout$w[sampled ==1], pch=16)

# And produce correct estimate:
weighted.mean(samp$support, w = kbalout$w[sampled==1])

# kbal correctly downweights white republicans and non-white non-republicans
# and upweights the non-white republicans and white non-republicans
unique(round(cbind(samp[,-3], k_bal_weight = kbalout$w[sampled==1]),6))
```

`lalonde` *Data from National Supported Work program and Panel Study in Income Dynamics*

Description

Dehejia and Wahba (1999) sample of data from Lalonde (1986). This data set includes 185 treated units from the National Supported Work (NSW) program, paired with 2490 control units drawn from the Panel Study of Income Dynamics (PSID-1).

The treatment variable of interest is `nsw`, which indicates that an individual was in the job training program. The main outcome of interest is real earnings in 1978 (`re78`). The remaining variables are characteristics of the individuals, to be used as controls.

Usage

```
lalonde
```

Format

A data frame with 2675 rows and 14 columns.

nsw treatment indicator: participation in the National Supported Work program.

re78 real earnings in 1978 (outcome)

u78 unemployed in 1978; actually an indicator for zero income in 1978

age age in years

black indicator for identifying as black

hisp indicator for identifying as Hispanic

race_ethnicity factor for self-identified race/ethnicity; same information as `black` and `hisp` in character form.

married indicator for being married

re74 real income in 1974

re75 real income in 1975

u74 unemployment in 1974; actually an indicator for zero income in 1974

u75 unemployment in 1975; actually an indicator for zero income in 1975

educ Years of education of the individual

nodegr indicator for no high school degree; actually an indicator for years of education less than 12

References

Dehejia, Rajeev H., and Sadek Wahba. "Causal effects in non-experimental studies: Reevaluating the evaluation of training programs." Journal of the American statistical Association 94.448 (1999): 1053-1062.

LaLonde, Robert J. "Evaluating the econometric evaluations of training programs with experimental data." The American economic review (1986): 604-620.

`makeK` *Build the Gaussian Kernel Matrix*

Description

Builds the Gaussian kernel matrix using Rcpp.

Usage

```
makeK(allx, useasbases = NULL, b = NULL, linkernel = FALSE, scale = TRUE)
```

Arguments

allx	a data matrix containing all observations where rows are units and columns are covariates.
useasbases	a binary vector with length equal to the number of observations (rows in allx) to specify which bases to use when constructing the kernel matrix (columns of K). If not specified, the default is to use all observations.
b	Scaling factor in the calculation of Gaussian kernel distance equivalent to the entire denominator $2\sigma^2$ of the exponent. Default is twice the number of covariates or columns in allx.
linkernel	a logical value indicating whether to use a linear kernel, $K = XX'$, which in practice employs $K = X$. Default is FALSE.
scale	a logical value indicating whether to standardize allx (demeaned with sd=1) before constructing the kernel matrix. Default is TRUE.

Value

K	The kernel matrix

Examples

```
#load and clean data a bit

set.seed(123)
data("lalonde")
# Select a random subset of 500 rows
lalonde_sample <- sample(1:nrow(lalonde), 500, replace = FALSE)
lalonde <- lalonde[lalonde_sample, ]

xvars <- c("age","black","educ","hisp","married","re74","re75","nodegr","u74","u75")

#note that lalonde$nsw is the treatment vector, so the observed is 1-lalonde$nsw
#running makeK with the sampled/control units as the bases given
#the large size of the data
K <- makeK(allx = lalonde[,xvars], useasbases = 1-lalonde$nsw)
```

---

one_hot *One-Hot Encoding for Categorical Data*

Description

Converts raw categorical string/factor data matrix into numeric one-hot encoded data matrix. Intended to help prepare data to be passed to kbal argument allx when categorical data is used.

Usage

```
one_hot(data)
```

Arguments

data	a dataframe or matrix where columns are string or factor type covariates

Value

onehot_data	a matrix of combined sample and population data with rows corresponding to units and columns one-hot encoded categorical covariates

Examples

```
#Ex 1. Make up some categorical demographic data
dat = data.frame(pid = c(rep("Rep", 20),
                         rep("Dem", 20),
                         rep("Ind", 20)),
                 gender = c(rep("female", 35),
                            rep("male", 25)))
#Convert to one-hot encoded data matrix:
onehot_dat = one_hot(dat)

#Ex 2. lalonde data
set.seed(123)
data("lalonde")
# Select a random subset of 500 rows
lalonde_sample <- sample(1:nrow(lalonde), 500, replace = FALSE)
lalonde <- lalonde[lalonde_sample, ]

cat_vars=c("black","hisp","married","nodegr","u74","u75")
onehot_lalonde = one_hot(lalonde[, cat_vars])
```

Index