Package: polymatching (via r-universe)

March 9, 2025

Type Package

Title A Matching Algorithm for Designs with Multiple Groups

Version 1.0.1

Description Includes functions implementing the conditionally optimal matching algorithm, which can be used to generate matched samples in designs with multiple groups. The algorithm is described in Nattino, Song and Lu (2022) <doi:10.1016/j.csda.2021.107364>.

License GPL-3

Encoding UTF-8

Imports optmatch, ggplot2, gridExtra, tidyr, utils, stats, dplyr, magrittr, rlang, methods

Suggests VGAM, knitr, rmarkdown

RoxygenNote 7.3.2

NeedsCompilation no

Author Giovanni Nattino [aut, cre], Bo Lu [aut], Chi Song [aut], Henry Xiang [aut]

Maintainer Giovanni Nattino <giovanni.nattino@marionegri.it>

Repository CRAN

Date/Publication 2025-02-07 18:00:02 UTC

Config/pak/sysreqs libicu-dev

Contents

`balance` *Evaluating the Balance of Covariates After Matching*

Description

The function `balance` computes the standardized mean differences and the ratio of the variances among treatment groups, before and after matching. The function computes the two measures of balance for each pair of treatment groups.

Usage

```
balance(
  formulaBalance,
  match_id,
  data,
  weights_before = NULL,
  weights_after = NULL
)
```

Arguments

`formulaBalance`	Formula with form `group ~ x_1 + ... + x_p`. `group` is the variable identifying the treatment groups/exposures. The balance is evaluated for the covariates `x_1`,...,`x_p`. Numeric and integer variables are treated as continuous. Factor variables are treated as categorical. Factor variables with two levels are treated as binary.
`match_id`	Vector identifying the matched sets—matched units must have the same identifier. It is generated by `polymatch`.
`data`	The `data.frame` object with the data.
`weights_before`	Optional vector of weights of the observations to be considered in the unmatched dataset. To compute the unweighted standardized mean differences, set `weights_before` to NULL (default).
`weights_after`	Vector of weights for the matched dataset. Set it to NULL (default) to compute the unweighted standardized mean differences.

Value

A `data.frame` containing the standardized differences and ratios of the variances (only for continuous variables) for each pair of treatment groups. A graphical representation of the results can be generated with `plotBalance`.

See Also

`polymatch` to generate matched samples and `plotBalance` to graphically represent the indicators of balance.

Examples

```
#Generate a datasets with group indicator and four variables:
#- var1, continuous, sampled from normal distributions;
#- var2, continuous, sampled from beta distributions;
#- var3, categorical with 4 levels;
#- var4, binary.
set.seed(1234567)
dat <- data.frame(group = c(rep("A",10),rep("B",20),rep("C",30)),
               var1 = c(rnorm(10,mean=0,sd=1),
                        rnorm(20,mean=1,sd=2),
                        rnorm(30,mean=-1,sd=2)),
               var2 = c(rbeta(10,shape1=1,shape2=1),
                        rbeta(20,shape1=2,shape2=1),
                        rbeta(30,shape1=1,shape2=2)),
               var3 = factor(c(rbinom(10,size=3,prob=.4),
                               rbinom(20,size=3,prob=.5),
                               rbinom(30,size=3,prob=.3))),
               var4 = factor(c(rbinom(10,size=1,prob=.5),
                               rbinom(20,size=1,prob=.3),
                               rbinom(30,size=1,prob=.7))))

#Match on propensity score
#--------------------------

#With multiple groups, need a multinomial model for the PS
library(VGAM)
psModel <- vglm(group ~ var1 + var2 + var3 + var4,
                family=multinomial, data=dat)
#Estimated logits - 2 for each unit: log(P(group=A)/P(group=C)), log(P(group=B)/P(group=C))
logitPS <- predict(psModel, type = "link")
dat$logit_AvsC <- logitPS[,1]
dat$logit_BvsC <- logitPS[,2]

#Match on logits of PS
resultPs <- polymatch(group ~ logit_AvsC + logit_BvsC, data = dat,
                    distance = "euclidean")
dat$match_id_ps <- resultPs$match_id

#Evaluate balance in covariates
tabBalancePs <- balance(group ~ var1 + var2 + var3 + var4,
                        match_id = dat$match_id_ps, data = dat)
tabBalancePs

#You can also represent the standardized mean differences with 'plotBalance'
#plotBalance(tabBalancePs, ratioVariances = TRUE)
```

`plotBalance` *Summary Plot of Balance in Covariates*

Description

The function generates a plot summarizing the balance of the covariates.

Usage

```
plotBalance(dataBalance, ratioVariances = FALSE, boxplots = TRUE)
```

Arguments

`dataBalance`	the output of `balance`.
`ratioVariances`	Boolean. If `TRUE`, the generated plot contains two panels: one for the standardized differences and one for the ratios of the variances. If `FALSE` (the default), only the standardized differences are represented.
`boxplots`	Boolean. If `TRUE` (default), boxplots are added to the plot, to show the distribution of the standardized differences and ratios of the variances.

Value

If at least one of the covariates is continuous and `ratioVariances=TRUE`, the function generates a plot with two panels: one for the standardized differences and one for the ratio of the variances (only for the continous variables). If either all the covariates are categorical/binary or `ratioVariances=FALSE` (or both), only the plot with the standardized differences is generated. The function also returns a list with the `ggplot2` objects corresponding to the generated plot(s).

See Also

`polymatch` to generate matched samples and `balance` to compute the indicators of balance.

Examples

```
#See examples of function 'balance'
```

`polymatch` *Polymatching*

Description

`polymatch` generates matched samples in designs with up to 10 groups.

Usage

```
polymatch(
  formulaMatch,
  start = "small.to.large",
  data,
  distance = "euclidean",
  exactMatch = NULL,
  vectorK = NULL,
  iterate = TRUE,
  niter_max = 50,
  withinGroupDist = TRUE,
  verbose = TRUE
)
```

Arguments

`formulaMatch`	Formula with form `group ~ x_1 + ... + x_p`, where `group` is the name of the variable identifying the treatment groups/exposures and `x_1`,...,`x_p` are the matching variables.
`start`	An object specifying the starting point of the iterative algorithm. Three types of input are accepted: • `start="small.to.large"` (default): the starting matched set is generated by matching groups from the smallest to the largest. • Users can specify the order to be used to match groups for the starting sample. For example, if there are four groups with labels "A","B","C" and "D", `start="D-B-A-C"` generates the starting sample by matching groups "D" and "B", then units from "A" to the "D"-"B"pairs, then units from "C" to the "D"-"B"-"A" triplets. • Users can provide the starting matched set and the algorithm will explore possible reductions in the total distance. In this case, `start` must be a vector with the IDs of the matched sets, i.e., a vector with length equal to the number of rows of `data` where matched subjects are flagged with the same value and non-matched subjects have value `NA`.
`data`	The `data.frame` object with the data.
`distance`	String specifying whether the distance between pairs of observations should be computed with the Euclidean (`"euclidean"`, default) or Mahalanobis (`"mahalanobis"`) distance. See section 'Details' for further information.
`exactMatch`	Formula with form `~ z_1 + ... + z_k`, where `z_1`,...,`z_k` must be factor variables. Subjects are exactly matched on `z_1`,...,`z_k`, i.e., matched within levels of these variables.
`vectorK`	A named vector with the number of subjects from each group in each matched set. The names of the vector must be the labels of the groups, i.e., the levels of the variable identifying the treatment groups/exposures. For example, in case of four groups with labels "A","B","C" and "D" and assuming that the desired design is 1:2:3:3 (1 subject from A, 2 from B, 3 from C and 3 from D in each matched set), the parameter should be set to `vectorK = c("A" = 1, "B"`

= 2, "C" = 3, "D" = 3). By default, the generated matched design includes 1 subject per group in each matched set, i.e, a 1:1: ... :1 matched design.

`iterate`	Boolean specifying whether iterations should be done (`iterate=TRUE`, default) or not (`iterate=FALSE`).
`niter_max`	Maximum number of iterations. Default is 50.
`withinGroupDist`	Boolean specifying whether the distances within the same treatment/exposure group should be considered in the total distance. For example, in a 1:2:3 matched design among the groups A, B and C, the parameters controls whether the distance between the two subjects in B and the three pairwise distances among the subjects in C should be counted in the total distance. The default value is `TRUE`.
`verbose`	Boolean: should text be printed in the console? Default is `TRUE`.

Details

The function implements the conditionally optimal matching algorithm, which iteratively uses two-group optimal matching steps to generate matched samples with small total distance. In the current implementation, it is possible to generate matched samples with multiple subjects per group, with the matching ratio being specified by the `vectorK` parameter.

The steps of the algorithm are described with the following example. Consider a 4-group design with groups labels "A", "B", "C" and "D" and a 1:1:1:1 matching ratio. The algorithm requires a set of quadruplets as starting point. The argument `start` defines the approach to be used to generate such a starting point. `polymatch` generates the starting point by sequentially using optimal two-group matching. In the default setting (`start="small.to.large"`), the steps are:

1. optimally match the two smallest groups;
2. optimally match the third smallest group to the pairs generated in the first step;
3. optimally match the last group to the triplets generated in the second step.

Notably, we can use the optimal two-group algorithm in steps 2) and 3) because they are two-dimensional problems: the elements of one group on one hand, fixed matched sets on the other hand. The order of the groups to be considered when generating the starting point can be user-specified (e.g., `start="D-B-A-C"`). In alternative, the user can provide a matched set that will be used as starting point.

Given the starting matched set, the algorithm iteratively explores possible reductions in the total distance (if `iterate="TRUE"`), by sequentially relaxing the connection to each group and rematching units of that group. In our example:

1. rematch "B-C-D" triplets within the starting quadruplets to units in group "A";
2. rematch "A-C-D" triplets within the starting quadruplets to units in group "B";
3. rematch "A-B-D" triplets within the starting quadruplets to units in group "C";
4. rematch "A-B-C" triplets within the starting quadruplets to units in group "D".

If none of the sets of quadruplets generated in 1)-4) has smaller total distance than the starting point, the algorihm stops. Otherwise, the set of quadruplets with smallest distance is seleceted and the process iterated, until no reduction in the total distance is found or the number of maximum iterations is reached (`niter_max=50` by default).

The total distance is defined as the sum of all the within-matched-set distances. The within-matched-set distance is defined as the sum of the pairwise distances between pairs of units in the matched set. The type of distance is specified with the `distance` argument. The current implementation supports Euclidean (`distance="euclidean"`) and Mahalanobis (`distance="mahalanobis"`) distances. In particular, for the Mahalanobis distance, the covariance matrix is defined only once on the full dataset.

Value

A list containing the following components:

match_id A numeric vector identifying the matched sets—matched units have the same identifier.

total_distance Total distance of the returned matched sample.

total_distance_start Total distance at the starting point.

See Also

`balance` and `plotBalance` to summarize the balance in the covariates.

Examples

```
#Generate a datasets with group indicator and four variables:
#- var1, continuous, sampled from normal distributions;
#- var2, continuous, sampled from beta distributions;
#- var3, categorical with 4 levels;
#- var4, binary.
set.seed(1234567)
dat <- data.frame(group = c(rep("A",10),rep("B",20),rep("C",30)),
               var1 = c(rnorm(10,mean=0,sd=1),
                        rnorm(20,mean=1,sd=2),
                        rnorm(30,mean=-1,sd=2)),
               var2 = c(rbeta(10,shape1=1,shape2=1),
                        rbeta(20,shape1=2,shape2=1),
                        rbeta(30,shape1=1,shape2=2)),
               var3 = factor(c(rbinom(10,size=3,prob=.4),
                               rbinom(20,size=3,prob=.5),
                               rbinom(30,size=3,prob=.3))),
               var4 = factor(c(rbinom(10,size=1,prob=.5),
                               rbinom(20,size=1,prob=.3),
                               rbinom(30,size=1,prob=.7))))

#Match on propensity score
#-------------------------

#With multiple groups, need a multinomial model for the PS
library(VGAM)
psModel <- vglm(group ~ var1 + var2 + var3 + var4,
                family=multinomial, data=dat)
#Estimated logits - 2 for each unit: log(P(group=A)/P(group=C)), log(P(group=B)/P(group=C))
logitPS <- predict(psModel, type = "link")
dat$logit_AvsC <- logitPS[,1]
```

```
dat$logit_BvsC <- logitPS[,2]

#Match on logits of PS
resultPs <- polymatch(group ~ logit_AvsC + logit_BvsC, data = dat,
                    distance = "euclidean")
dat$match_id_ps <- resultPs$match_id


#Match on covariates
#--------------------


#Match on continuous covariates with exact match on categorical/binary variables
resultCov <- polymatch(group ~ var1 + var2, data = dat,
                        distance = "mahalanobis",
                        exactMatch = ~var3+var4)
dat$match_id_cov <- resultCov$match_id
```

polymatching *Polymatching: Matching in Designs with Multiple Treatment Groups*

Description

The package implements the conditionally optimal matching algorithm, which can be used to generate matched samples in designs with multiple treatment groups.

Details

Currently, the algorithm can be applied to datasets with up to 10 groups and generates matched samples with one subject per group. The package provides functions to generate the matched sample and to evaluate the balance in key covariates.

Generating the Matched Sample

The function implementing the matching algorithm is `polymatch`. The algorithm is iterative and needs a matched sample with one subject per group as starting point. This matched sample can be automatically generated by `polymatch` or can be provided by the user. The algorithm iteratively explores possible reductions in the total distance of the matched sample.

Evaluating Balance in Covariates

Balance in key covariates can be evaluated with the function `balance`. Given a matched sample and a set of covariates of interest, the function computes the standardized differences and the ratio of the variances for each pair of treatment groups in the study design. For 3, 4, 5 and 6 groups, there are 3, 6, 10 and 15 pairs of groups and the balance is evaluated before and after matching. The result of `balance` can be graphically represented with `plotBalance`.

Author(s)

Maintainer: Giovanni Nattino <`giovanni.nattino@marionegri.it`>

Authors:

- Bo Lu
- Chi Song
- Henry Xiang

Index