Package ‘quickblock’

July 22, 2025

Type Package

Title Quick Threshold Blocking

Version 0.2.2

Date 2024-07-31

Description Provides functions for assigning treatments in randomized experiments using near-optimal threshold blocking. The package is made with large data sets in mind and derives blocks more than an order of magnitude quicker than other methods.

Depends R (>= 3.4.0), distances

Imports scclust (>= 0.2.0)

Suggests testthat

NeedsCompilation yes

License GPL (>= 3)

URL `https://github.com/fsavje/quickblock`

BugReports `https://github.com/fsavje/quickblock/issues`

Encoding UTF-8

RoxygenNote 7.2.3

Author Fredrik Savje [aut, cre],
Jasjeet Sekhon [aut],
Michael Higgins [aut]

Maintainer Fredrik Savje <`rpackages@fredriksavje.com`>

Repository CRAN

Date/Publication 2024-07-31 22:30:27 UTC

Contents

quickblock-package 2
assign_treatment 2
blocking_estimator 3

is.qb_blocking 5
qb_blocking 6
quickblock 7

Index **10**

`quickblock-package` *quickblock: Quick Threshold Blocking*

Description

Provides functions for assigning treatments in randomized experiments using near-optimal threshold blocking. The package is made with large data sets in mind and derives blocks more than an order of magnitude quicker than other methods.

Details

See `quickblock` for the main blocking function.

See the package's website for more information: `https://github.com/fsavje/quickblock`.

Bug reports and suggestions are greatly appreciated. They are best reported here: `https://github.com/fsavje/quickblock/issues`.

References

Higgins, Michael J., Fredrik Sävje and Jasjeet S. Sekhon (2016), 'Improving massive experiments with threshold blocking', *Proceedings of the National Academy of Sciences*, **113:27**, 7369–7376.

`assign_treatment` *Random treatment assignment with blocks*

Description

`assign_treatment` randomly assigns treatments to the units in the sample so to best maintain the equal proportions of conditions within each block. The function expects the user to provide a blocking object and treatment conditions.

Usage

```
assign_treatment(blocking, treatments = c("Treated", "Control"))
```

Arguments

`blocking`	`qb_blocking` or `scclust` object with the blocked units.
`treatments`	character vector with treatment conditions.

Details

When the number of treatment conditions evenly divides the size of a block, the conditions will be repeated that many times within the block. For example, with three conditions, `c("T1", "T2", "C")`, and a block with six units, two units will be assigned to each condition.

When the number of treatment conditions does not evenly divide the block size, the conditions are repeated up to the closest multiple lower than the block size and the remaining conditions are chosen at random. For example, with the three conditions from above and a block with four units, each condition will be repeated once (since `floor(4/3) == 1`). One additional condition is needed to assign all units in the block, and that condition is selected at random from `c("T1", "T2", "C")` with equal probability. In a block with 8 units, each condition will be repeated twice (`floor(8/3) == 2`). Two additional conditions are now needed, and they are chosen from `c("T1", "T2", "C")` without replacement.

In all cases, the treatment conditions within a block are shuffled so that all units have the same probability of being assigned to each condition. Units not assigned to blocks will not be assigned treatments (indicated by `NA`).

Value

Returns a factor with the assigned treatments.

Examples

```
# Example blocking
my_blocking <- qb_blocking(c("A", "A", "B", "C", "B",
                             "C", "C", "A", "B", "B"))

# Two treatment conditions
assign_treatment(my_blocking)

# Three treatment conditions
assign_treatment(my_blocking, c("T1", "T2", "C"))

# Four treatment conditions
# (This throws warning because some blocks contain less than four units)
## Not run: assign_treatment(my_blocking, c("T1", "T2", "T3", "C"))
```

`blocking_estimator` *Estimator for treatment effects in blocked experiments*

Description

`blocking_estimator` estimates treatment effects in blocked experiments. The function expects the user to provide the outcomes, a blocking object and treatment assignments. It returns point estimates of sample average treatment effects and variance estimates.

Usage

```
blocking_estimator(outcomes, blocking, treatments)
```

Arguments

`outcomes`	numeric vector with observed outcomes.
`blocking`	`qb_blocking` or `scclust` object with the block assignments.
`treatments`	factor specifying the units' treatment assignments.

Details

To produce point estimates, `blocking_estimator` requires that each block contains at least one unit assigned to each treatment condition. For variance estimation, it requires that each block contains at least two units assigned to each condition. When treatments have been assigned with the `assign_treatment` function (or an equivalent procedure), the variance estimators are conservative in expectation (see the referenced note below for details). If treatment is assigned with another method, the estimator might not be valid.

The function estimates treatment effects by aggregating block-level effect estimates. It estimates effects within each block by taking the difference in mean outcomes in the block. The sample-level estimate is then derived as the weighted average of the block-level effects using the size of the blocks as weights. In detail, let n_b be the number of units assigned to block b, and n be the total number of units in the sample. Let $Y(t, b)$ be the average outcome for units assigned to treatment t in block b. The effect of treatment t versus treatment s is then estimated as:

$$\sum \frac{n_b}{n}[Y(t,b) - Y(s,b)],$$

where the sum is taken over the blocks in the experiment. See the referenced note for more details.

Value

A list with two numeric matrices with estimated treatment effects and their estimated variances is returned. The first matrix (`effects`) contains estimated treatment effects. Rows in this matrix indicate minuends in the treatment effect contrast and columns indicate subtrahends. For example, in the matrix:

	a	b	c
a	0.0	4.5	5.5
b	-4.5	0.0	1.0
c	-5.5	-1.0	0.0

the estimated treatment effect between conditions a and b is 4.5, and the estimated treatment effect between conditions c and b is -1.0.

The second matrix (`effect_variances`) contains estimates of variances of the corresponding effect estimators.

References

Higgins, Michael J., Fredrik Sävje and Jasjeet S. Sekhon (2015), ‘Blocking estimators and inference under the Neyman-Rubin model’, arXiv 1510.01103. https://arxiv.org/abs/1510.01103

Examples

```
# Example blocking
my_blocking <- qb_blocking(c("A", "A", "B", "C", "B",
                             "C", "B", "C", "B", "A",
                             "C", "C", "A", "B", "B",
                             "B", "B", "A", "A", "C"))

# Two treatment conditions
my_treatments <- assign_treatment(my_blocking)
my_outcomes <- rnorm(20)
blocking_estimator(my_outcomes, my_blocking, my_treatments)

# Three treatment conditions
my_treatments <- assign_treatment(my_blocking, c("T1", "T2", "C"))
my_outcomes <- rnorm(20)
blocking_estimator(my_outcomes, my_blocking, my_treatments)

# Four treatment conditions
# (This will throw an error because variances cannot be estimated)
my_treatments <- assign_treatment(my_blocking, c("T1", "T2", "T3", "C"))
my_outcomes <- rnorm(20)
## Not run: blocking_estimator(my_outcomes, my_blocking, my_treatments)
```

---

is.qb_blocking *Check qb_blocking object*

Description

`is.qb_blocking` checks whether the provided object is a valid instance of the `qb_blocking` class.

Usage

```
is.qb_blocking(x)
```

Arguments

x	object to check.

Details

`is.qb_blocking` does not check whether the blocking itself is sensible or whether it satisfies some set of constraints. See `check_clustering` for that functionality.

Value

Returns TRUE if x is a valid qb_blocking object, otherwise FALSE.

qb_blocking *Constructor for qb_blocking objects*

Description

The qb_blocking function constructs a qb_blocking object from existing block labels. The function does not derive blockings from sets of data points; see quickblock for that functionality.

Usage

```
qb_blocking(block_labels, unassigned_labels = NULL, ids = NULL)
```

Arguments

block_labels	a vector containing each unit's block label.
unassigned_labels	labels that denote unassigned units. If NULL, NA values in block_labels are used to denote unassigned units.
ids	IDs of the units. Should be a vector of the same length as block_labels or NULL. If NULL, the IDs are set to 1:length(group_labels).

Details

qb_blocking objects are based on integer vectors, and it indexes the blocks starting with zero. The qb_blocking class inherits from the scclust class.

Value

Returns a qb_blocking object with the blocking described by the provided labels.

Examples

```
# 10 units in 3 blocks
blocking1 <- qb_blocking(c("A", "A", "B", "C", "B",
                           "C", "C", "A", "B", "B"))

# 8 units in 3 blocks, 2 units unassigned
blocking2 <- qb_blocking(c(1, 1, 2, 3, 2,
                           NA, 3, 1, NA, 2))

# Custom labels indicating unassigned units
blocking3 <- qb_blocking(c("A", "A", "B", "C", "NONE",
                           "C", "C", "NONE", "B", "B"),
                         unassigned_labels = "NONE")
```

```
# Two different labels indicating unassigned units
blocking4 <- qb_blocking(c("A", "A", "B", "C", "NONE",
                           "C", "C", "0", "B", "B"),
                         unassigned_labels = c("NONE", "0"))

# Custom unit IDs
blocking5 <- qb_blocking(c("A", "A", "B", "C", "B",
                           "C", "C", "A", "B", "B"),
                         ids = letters[1:10])
```

quickblock *Construct threshold blockings*

Description

quickblock constructs near-optimal threshold blockings. The function expects the user to provide distances measuring the similarity of units and a required minimum block size. It then constructs a blocking so that units assigned to the same block are as similar as possible while satisfying the minimum block size.

Usage

```
quickblock(
  distances,
  size_constraint = 2L,
  caliper = NULL,
  break_large_blocks = FALSE,
  ...
)
```

Arguments

distances	distances object or a numeric vector, matrix or data frame. The parameter describes the similarity of the units to be blocked. It can either be preprocessed distance information using a distances object, or raw covariate data. When called with covariate data, Euclidean distances are calculated unless otherwise specified.
size_constraint	integer with the required minimum number of units in each block.
caliper	restrict the maximum within-block distance.
break_large_blocks	logical indicating whether large blocks should be broken up into smaller blocks.
...	additional parameters to be sent either to the distances function when the distances parameter contains covariate data, or to the underlying sc_clustering function.

Details

The caliper parameter constrains the maximum distance between units assigned to the same block. This is implemented by restricting the edge weight in the graph used to construct the blocks (see sc_clustering for details). As a result, the caliper will affect all blocks and, in general, make it harder for the function to find good matches even for blocks where the caliper is not binding. In particular, a too tight caliper can lead to discarded units that otherwise would be assigned to a block satisfying both the matching constraints and the caliper. For this reason, it is recommended to set the caliper value quite high and only use it to avoid particularly poor blocks. It strongly recommended to use the caliper parameter only when primary_unassigned_method = "closest_seed" in the underlying sc_clustering function (which is the default behavior).

The main algorithm used to construct the blocking may produce some blocks that are much larger than the minimum size constraint. If break_large_blocks is TRUE, all blocks twice as large as size_constraint will be broken into two or more smaller blocks. Block are broken so to ensure that the new blocks satisfy the size constraint. In general, large blocks are produced when units are highly clustered, so breaking up large blocks will often only lead to small improvements. The blocks are broken using the hierarchical_clustering function.

quickblock calls sc_clustering with seed_method = "inwards_updating". The seed_method parameter governs how the seeds are selected in the nearest neighborhood graph that is used to construct the blocks (see sc_clustering for details). The "inwards_updating" option generally works well and is safe with most datasets. Using seed_method = "exclusion_updating" often leads to better performance (in the sense of blocks with more similar units), but it may increase run time. Discrete data (or more generally when units tend to be at equal distance to many other units) will lead to particularly poor run time with this option. If the dataset has at least one continuous covariate, "exclusion_updating" is typically quick. A third option is seed_method = "lexical", which decreases the run time relative to "inwards_updating" (sometimes considerably) at the cost of performance. quickblock passes parameters on to sc_clustering, so to change seed_method, call quickblock with the parameter specified as usual: quickblock(..., seed_method = "exclusion_updating").

Value

Returns a qb_blocking object with the constructed blocks.

References

Higgins, Michael J., Fredrik Sävje and Jasjeet S. Sekhon (2016), ‘Improving massive experiments with threshold blocking’, *Proceedings of the National Academy of Sciences*, **113:27**, 7369–7376.

See Also

See sc_clustering for the underlying function used to construct the blocks.

Examples

```
# Construct example data
my_data <- data.frame(x1 = runif(100),
                      x2 = runif(100))


# Make distances
```

```
my_distances <- distances(my_data, dist_variables = c("x1", "x2"))

# Make blocking with at least two units in each block
quickblock(my_distances)

# Require at least three units in each block
quickblock(my_distances, size_constraint = 3)

# Impose caliper
quickblock(my_distances, caliper = 0.2)

# Break large block
quickblock(my_distances, break_large_blocks = TRUE)

# Call `quickblock` directly with covariate data (ie., not pre-calculating distances)
quickblock(my_data[c("x1", "x2")])

# Call `quickblock` directly with covariate data using Mahalanobis distances
quickblock(my_data[c("x1", "x2")], normalize = "mahalanobize")
```

Index

assign_treatment, 2, 4

blocking_estimator, 3

check_clustering, 5

distances, 7

hierarchical_clustering, 8

is.qb_blocking, 5

qb_blocking, 2, 4–6, 6, 8
quickblock, 2, 6, 7
quickblock-package, 2

sc_clustering, 7, 8
scclust, 2, 4, 6