Package: minMSE (via r-universe)

September 8, 2024

Type Package

Title Implementation of the minMSE Treatment Assignment Method for One or Multiple Treatment Groups

Version 0.5.1

Author Sebastian O. Schneider, Giulia Baldini

Maintainer Sebastian O. Schneider <sschneider@coll.mpg.de>

Description Performs treatment assignment for (field) experiments considering available, possibly multivariate and continuous, information (covariates, observable characteristics), that is: forms balanced treatment groups, according to the minMSE-method as proposed by Schneider and Schlather (2017) <DOI:10.1017/pan.2017.16>.

URL https://www.sebastianoschneider.com

License GNU General Public License

Encoding UTF-8

Imports MASS, Rcpp

RoxygenNote 7.1.0

LinkingTo Rcpp

NeedsCompilation yes

Date/Publication 2021-11-26 22:50:08 UTC

Repository https://sebastianoschneider.r-universe.dev

RemoteUrl https://github.com/cran/minMSE

RemoteRef HEAD

RemoteSha 343ca090a16cb8147079bd73e4866187afc170f9

Contents

`assign_minMSE_treatment`

minMSE Treatment Assignment for One or Multiple Treatment Groups – Wrapper

---

Description

Calling assign_treatment, this user-friendly wrapper function computes a given number of treatment assignment vectors that will contain at most a certain percentage of duplicates, specified in percentage_equal_treatments. This is useful if non-parametric inference (randomization inference, sometimes called Fisher's exact test or permutation test) is desired (which is often advised), for analysis of significance of the treatment effect. The main function, assign_treatment, computes the treatment assignment vector according to available data (observable characteristics, covariate vectors) given about the units (individuals or clusters, such as schools, hospitals, ...)

Usage

```
assign_minMSE_treatment(data,
                        prev_treatment = NULL,
                        n_treatments = 1,
                        n_per_group = NULL,
                        mse_weights = NULL,
                        iterations = 50,
                        change = 3,
                        cooling = 1,
                        t0 = 10,
                        tmax = 10,
                        built_in = 0,
                        desired_test_vectors = 100,
                        percentage_equal_treatments = 1,
                        plot = 0,
                        trace_output = 1,
                        filename = NULL)
```

Arguments

`data`	a dataframe or a matrix containing the covariate vectors for each attribute. The values might be missing or on different scales as the software deals with missing values and scaling automatically.
`prev_treatment`	takes a numerical vector of partial treatment assignment as argument, and assigns the missing units (where the value is NA) to a treatment group while minimizing the objective function. Non-missing values are copied to the new vector, i.e., treatment group assignment of these observations is unaffected, but taken into consideration for achieving balanced treatment groups.
`n_treatments`	specifies the number of treatment groups desired (in addition to the control group); minimum and default value is n_treatments = 1.
`n_per_group`	specifies a vector containing uneven sizes for the treatment groups. Default value is NULL, which yields even sized groups. The sum of the elements in the vector should be equal to the total number of observations.
`mse_weights`	a vector containing the mse_weights for each treatment, or a matrix containing the mse_weights for treatments and outcomes and scaling factors.
`iterations`	specifies the number of iterations the algorithm performs; the default value is iterations = 50. Depending on the number of units and the number of covariates to consider for group assignment, a high value could result in a long run-time.
`change`	sets the number of units to exchange treatment in each iteration; the default value is change = 3. In case of big datasets (e.g. with more than 100 units), one might consider increasing the default value.
`cooling`	specifies the cooling scheme for the simulated annealing algorithm to use. cooling = 1, which is the default scheme, sets the temperature to $$t0/log(floor((k-1)/tmax) * tmax + exp(1)),$$ whereas cooling = 2 sets the temperature to the faster decreasing sequence $$t0/(floor((k-1)/tmax) * tmax + 1).$$ In praxis, cooling schemes are mostly of one of these forms. One might want to change the cooling scheme if the plot indicates a too slow decrease of objective values. For a theoretical discussion of cooling schemes Belisle (see 1992, p. 890).
`t0`	sets the starting temperature for the simulated annealing algorithm, see Belisle (1992) for theoretical convergence considerations. In praxis, a lower starting temperature t0 decreases the acceptance rate of a worse solution more rapidly. Specifying a negative number allows values proportional to the objective function, i.e. t0 = -5 sets the starting temperature to 1/5 of the objective function for the starting point, and thus - for the first tmax iterations of the algorithm - the difference of the old and the proposed solution is scaled by 1/5. When changing the default value, it should be considered that also worse solutions have to be accepted in order for the algorithm to escape a local minimum, so it should be chosen high enough. The default value is t0 = 10.

`tmax` specifies the number of function evaluations at each temperature: For instance, tmax = 10 makes the algorithm evaluate 10 treatment assignments that are found based on the current solution, before the temperature is decreased and thus the probability of accepting a worse solution is decreased. The default value is tmax = 10.

`built_in` if built_in = 1 the R built-in function optim with method 'SANN' (Simulated ANNealing) will be used to optimize the function. Otherwise, if built_in = 0, our implementation of the simulated annealing will be used. The function built_in = 0 uses our first cooling function and this cannot be changed. To used the second cooling function, set built_in = 0. All the other parameters, such as iterations, change, t0, tmax are taken into account.

`desired_test_vectors` specifies the number of treatment assignment vectors that will be produced to perform Fischer's exact test (sometimes also called permutation test) for assessment of significance of the treatment effect (this, of course, will be done after treatment has been conducted and measurement of the outcome of interest has occurred). The number of possible treatment vectors will not exceed this number. The default value is desired_test_vectors = 100. For small datasets, one might consider increasing it without affecting performance. Note that this will affect your significance level: If desired_test_vectors = 100 and all of them are unique (see 'percentage_equal_treatments' below), you can achieve a significance level of at most $p < 0.01\%$.If desired_test_vectors = 1, then the program returns a single vector that can be used for treatment assignment. Note that Fischer's exact test might still be possible and that alternative treatment vectors might also be produced after treatment has been conducted; yet, it is not sure how many *different* vectors can be produced with a given number of iterations. It is, therefore, good practice to produce the desired number of vectors with treatment assignment. For testing purposes, however, one might want to produce just one vector.

`percentage_equal_treatments` the percentage of non-unique treatment vectors that we allow. The default value is percentage_equal_treatments = 1. Note that this will affect your significance level: If desired_test_vectors = 100 and percentage_equal_treatments = 1, you can achieve a significance level of at most $p < 0.01\%$.

`plot` can be used to draw a plot showing the value of the objective function for the a percentage of the iterations by setting plot = 1. The default setting is plot = 0, which suppresses the plot.

`trace_output` trace_output = 1 prints helpful output such as the current iteration. To avoid the program output to be too cumbersome, a more detailed output is saved in a txt file called program_output.txt.

`filename` takes a string that represents the name of the csv file where the possible treatment assignments will be stored. If filename = NULL, then the file will not be saved.

Value

The program returns a dataframe containing all the unique treatments generated by the program. It also outputs the maximum number of iterations that were reached before finding a non-unique vector.

Note

With the default setting of plotting and using the trace output, the program writes to different files. To avoid this, set plot = 0 and trace_output = 0. For the built-in function optim, the trace output is necessary for printing, because we pipe the output of the program to file to obtain the intermediate values of the optimization function.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

References

Schneider and Schlather (2017), Belisle (1992)

See Also

ginv, optim, assign_treatment

Examples

```
input <- data.frame(c(10, 20, 30, 40, 130, 40, 120, 5, 10, 80),
                    c(2, 6, 2, 8, 1, 10, 9, 8, 7, 5),
                    c(1, 0, 2, 1, 0, 1, 0, 2, 1, 0))
colnames(input) <- c("IQ", "grade_maths", "both_parents")

assign_minMSE_treatment(input,
                        prev_treatment = c(0, NA, NA, NA, 1, NA, NA, NA, NA, NA),
                        n_treatments = 2,
                        mse_weights = c(1, 2),
                        iterations = 100,
                        trace_output = 1,
                        built_in = 0,
                        desired_test_vectors = 100,
                        plot = 0,
                        filename = NULL)
```

---

assign_treatment *Min MSE Treatment Assignment*

Description

Computes the treatment assignment vector according to available data (observable characteristics, covariate vectors) given about the units (individuals or clusters, such as schools, hospitals, ...). Consider using the user-friendly wrapper function assign_minMSE_treatment.

Usage

```
assign_treatment(current_data,
                 prev_treatment = NULL,
                 evaluation_function = evaluate_solution,
                 swap_treatment_function = swap_treatment,
                 n_treatments = 1,
                 n_per_group = NULL,
                 mse_weights = NULL,
                 iterations = 50,
                 change = 3,
                 cooling = 1,
                 t0 = 10,
                 tmax = 10,
                 built_in = 0,
                 plot = 0,
                 create_plot_file = 1)
```

Arguments

current_data	a matrix containing the covariate vectors for each attribute. If the values are missing or on different scales, please use assign_minMSE_treatment, which automatically scales the data.
prev_treatment	takes a numerical vector of partial treatment assignment as argument, and assigns the missing units (where the value is NA) to a treatment group while minimizing the objective function. Non-missing values are copied to the new vector, i.e., treatment group assignment of these observations is unaffected, but taken into consideration for achieving balanced treatment groups.
evaluation_function	the function used to evaluate the MSE treatment. Default is evaluate_solution, which does not take into account outcome or treatment weights. Other options are evaluate_solution_vector and evaluate_solution_matrix.
swap_treatment_function	the function used to create new treatments. Default is swap_treatment. Other options are swap_treatment_prev which, given a previous treatment, creates a new treatment assignment that takes the previous one into account.
n_treatments	specifies the number of treatment groups desired (in addition to the control group); minimum and default value is n_treatments = 1.
n_per_group	specifies a vector containing uneven sizes for the treatment groups. Default value is NULL, which yields even sized groups. The sum of the elements in the vector should be equal to the total number of observations.
mse_weights	a vector containing the mse_weights for each treatment, or a matrix containing the mse_weights for treatments and outcomes and scaling factors.
iterations	specifies the number of iterations the algorithm performs; the default value is iterations = 50. Depending on the number of units and the number of covariates to consider for group assignment, a high value could result in a long run-time.

`change` sets the number of units to exchange treatment in each iteration; the default value is change = 3. In case of big datasets (e.g., with more than 100 units), one might consider increasing the default value.

`cooling` specifies the cooling scheme for the simulated annealing algorithm to use. cooling = 1, which is the default scheme, sets the temperature to

$$t0/log(floor((k-1)/tmax) * tmax + exp(1)),$$

whereas cooling = 2 sets the temperature to the faster decreasing sequence

$$t0/(floor((k-1)/tmax) * tmax + 1).$$

In praxis, cooling schemes are mostly of one of these forms. One might want to change the cooling scheme if the plot indicates a too slow decrease of objective values. For a theoretical discussion of cooling schemes, see Belisle (see 1992, p. 890).

`t0` sets the starting temperature for the simulated annealing algorithm, see Belisle (1992) for theoretical convergence considerations. In praxis, a lower starting temperature t0 decreases the acceptance rate of a worse solution more rapidly. Specifying a negative number allows values proportional to the objective function, i.e. t0 = -5 sets the starting temperature to 1/5 of the objective function for the starting point, and thus - for the first tmax iterations of the algorithm - the difference of the old and the proposed solution is scaled by 1/5. When changing the default value, it should be considered that also worse solutions have to be accepted in order for the algorithm to escape a local minimum, so it should be chosen high enough. The default value is t0 = 10.

`tmax` specifies the number of function evaluations at each temperature: For instance, tmax = 10 makes the algorithm evaluate 10 treatment assignments that are found based on the current solution, before the temperature is decreased and thus the probability of accepting a worse solution is decreased. The default value is tmax = 10.

`built_in` if built_in = 1 the R built-in function optim with method 'SANN' (Simulated ANNealing) will be used to optimize the function. Otherwise, if built_in = 0, our implementation of the simulated annealing will be used. The function built_in = 0 uses our first cooling function and this cannot be changed. To use the second cooling function, set built_in = 0. All the other parameters, such as iterations, change, t0, tmax are taken into account.

`plot` can be used to draw a plot showing the value of the objective function for the a percentage of the iterations by setting plot = 1. The default setting is plot = 0, which suppresses the plot.

`create_plot_file`
Used to overwrite the plot file, in case there already exists one. It should only be 1 (true) when this method is called without the wrapper assign_minMSE_treatment. This method alone is not capable of plotting, but it will create an auxiliary file that contains the information for plotting. To include plotting, use assign_minMSE_treatment with desired_test_vectors = 1.

Value

Returns the current assignment and the mean squared error value for that assignment.

Note

With the default setting of plotting and using the trace output, the program writes to different files. To avoid this, set plot = 0 and trace_output = 0. For the built-in function optim, the trace output is necessary for printing, because we pipe the output of the program to a file to obtain the intermediate values of the optimization function.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

References

Schneider and Schlather (2017), Belisle (1992)

See Also

ginv, optim

Examples

```
input <- matrix(1:30, nrow = 10, ncol = 3)

assign_treatment(input,
                 evaluation_function = evaluate_solution_vector,
                 swap_treatment_function = swap_treatment_prev,
                 prev_treatment = c(0, NA, NA, NA, 1, NA, NA, NA, NA, NA),
                 n_treatments = 2,
                 mse_weights = c(1, 2),
                 iterations = 100,
                 built_in = 0,
                 plot = 0)
```

---

count_occurrences *Count of Equal Treatment Vectors*

Description

Checks if the treatment vector given as argument already exists in the dataframe, i.e., has been produced by one or more earlier iteration(s).

Usage

```
count_occurrences(df_treatments, curr_treatment)
```

Arguments

df_treatments	dataframe containing all the discovered treatment vectors.
curr_treatment	treatment vector to be investigated.

Value

Returns the number of treatment assignment vectors which are equal to the one being investigated.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

Examples

```
df_treatments <- data.frame(c(0, 2, 0, 1, 1, 0, 2, 1, 1, 0),
                            c(0, 2, 0, 1, 2, 0, 1, 2, 1, 0),
                            c(0, 2, 1, 1, 2, 0, 0, 2, 0, 0))
colnames(df_treatments) <- c("treatment_iter_1", "treatment_iter_2", "treatment_iter_3")

count_occurrences(df_treatments,
                  c(0, 2, 1, 1, 2, 0, 0, 2, 0, 0))
```

---

evaluate_solution *Evaluate MSE Equation*

Description

The function computes the mean squared error for a given treatment assignment. More precisely: it computes the mean squared error of the treatment effect estimator resulting from the treatment groups as specified by the argument, the treatment assignment vector. The function uses matrix multiplication and the Moore-Penrose generalized inverse.

Usage

```
evaluate_solution(treatment, data, mse_weights = NULL)
```

Arguments

treatment	a treatment assignment. The treatment and the data must have the same number of observations (rows).
data	a matrix containing the covariate vectors for each attribute.
mse_weights	not used, needed for compatibility.

Value

Returns the mean squared error value for the current treatment assignment.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

References

Schneider and Schlather (2017),

See Also

ginv

Examples

```
input <- matrix(1:30, nrow = 10, ncol = 3)

evaluate_solution(treatment = c(0, 1, 1, 1, 1, 0, 0, 0, 0, 0),
                  input)
```

`evaluate_solution.optim`

Evaluate MSE Equation (using optim)

Description

This function calls evaluate_solution, but since optim requires fn and gr to have the same parameters, it has two additional ones.

Usage

```
evaluate_solution.optim(par,
                        data,
                        evaluation_function = evaluate_solution,
                        swap_treatment_function = NULL,
                        mse_weights = NULL,
                        change = NULL,
                        prev_index_list = NULL)
```

Arguments

`par`	a treatment assignment. The treatment and the data must have the same number of observations (rows).
`data`	a matrix containing the covariate vectors for each attribute.
`evaluation_function`	the function used to evaluate the MSE treatment. Default is evaluate_solution, which does not take into account outcome or treatment weights. Other options are evaluate_solution_vector and evaluate_solution_matrix.
`swap_treatment_function`	the parameter is only needed for optim, it does not play any role.
`mse_weights`	a vector containing the mse_weights for each treatment, or a matrix containing the mse_weights for treatments and outcomes and scaling factors.

change the parameter is only needed for optim, it does not play any role.

prev_index_list
the parameter is only needed for optim, it does not play any role.

Value

Returns the mean square error value for the current treatment assignment.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

References

Schneider and Schlather (2017),

See Also

ginv, optim

Examples

```
input <- matrix(1:30, nrow = 10, ncol = 3)

evaluate_solution.optim(par = c(0, 1, 1, 1, 1, 0, 0, 0, 0, 0),
                        input)
```

---

evaluate_solution_matrix

Evaluate MSE Equation

Description

The function computes the mean squared error for a given treatment assignment. More precisely: it computes the mean squared error of the treatment effect estimator resulting from the treatment groups as specified by the argument, the treatment assignment vector. The function uses matrix multiplication and the Moore-Penrose generalized inverse.

Usage

```
evaluate_solution_matrix(treatment, data, mse_weights)
```

Arguments

treatment	a treatment assignment. The treatment and the data must have the same number of observations (rows).
data	a matrix containing the covariate vectors for each attribute.
mse_weights	a matrix containing the mse_weights for treatments and outcomes and scaling factors.

Value

Returns the mean squared error value for the current treatment assignment.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

References

Schneider and Schlather (2017),

See Also

ginv

Examples

```
input <- matrix(1:30, nrow = 10, ncol = 3)

evaluate_solution_matrix(treatment = c(0, 1, 1, 1, 1, 0, 0, 0, 0, 0),
                         input,
                         mse_weights = matrix(1:20, nrow = 10, ncol = 2))
```

---

evaluate_solution_vector

Evaluate MSE Equation

Description

The function computes the mean squared error for a given treatment assignment. More precisely: it computes the mean squared error of the treatment effect estimator resulting from the treatment groups as specified by the argument, the treatment assignment vector. The function uses matrix multiplication and the Moore-Penrose generalized inverse.

Usage

```
evaluate_solution_vector(treatment, data, mse_weights)
```

Arguments

treatment	a treatment assignment. The treatment and the data must have the same number of observations (rows).
data	a matrix containing the covariate vectors for each attribute.
mse_weights	a vector containing the mse_weights for each treatment.

Value

Returns the mean squared error value for the current treatment assignment.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

References

Schneider and Schlather (2017),

See Also

ginv

Examples

```
input <- matrix(1:30, nrow = 10, ncol = 3)

evaluate_solution_vector(treatment = c(0, 1, 1, 1, 1, 0, 0, 0, 0, 0),
                         input,
                         mse_weights = c(1, 2))
```

---

output_file *TXT File That Contains Additional Output Information*

Description

File .txt that contains additional output information.

---

plotting_file *CSV File for Saving the Data to Plot*

Description

CSV file for saving the data to plot.

`sample_with_prev_treatment` *Sample Under Consideration of an Already Treated Subset of Units*

Description

Given a previous treatment assignment vector for a subset of all observations that treatment assigment is desired for, the function computes a treatment assignment vector for which the previously assigned units are not changed. At a later step, the previously assigned units are also taken into consideration for computation of the score value, the min MSE function, to achieve balanced treatment groups.

Usage

```
sample_with_prev_treatment(prev_treatment, n_treatments, n_per_group)
```

Arguments

`prev_treatment`	takes a numerical vector of partial treatment assignment as argument, and – for a start – assigns the missing units (where the value is NA) to a random treatment group, while maintaining the same proportions in the groups.
`n_treatments`	specifies the number of treatment groups desired (in addition to the control group). They might be more than the ones already defined in prev_treatment.
`n_per_group`	specifies the distribution of participants per experimental group. It is either an integer, which produces even-sized groups, or a vector which has the same length as the number of experimental groups.

Value

Returns a treatment assignment vector where the observations given by prev_treatment are unmodified, and the others are assigned to a group.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

Examples

```
sample_with_prev_treatment(prev_treatment = c(0, NA, NA, NA, 1, NA, NA, NA, NA, NA),
                           n_treatments = 2,
                           n_per_group = c(2, 4, 4))
```

scale_vars *Covariate Vectors Scaling*

Description

Scales the data such that the empty fiels (NA) are the mean of the column and all variables are scaled to have variance 1. In case a variable has cero variance, the variable internally is treated as if it was 0, that way it is not taken into account for treatment assignment.

Usage

```
scale_vars(data)
```

Arguments

data	a dataframe containing the covariate vectors for each attribute.

Value

Returns a dataframe where the empty fields are filled with the mean of the column, and for all variables the variance is 1.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

Examples

```
input <- data.frame(c(10, 20, 30, 40, 130, 40, 120, 5, 10, 80),
                    c(2, 6, 2, 8, 1, 10, 9, 8, 7, 5),
                    c(1, 0, 2, 1, 0, 1, 0, 2, 1, 0))
colnames(input) <- c("IQ", "grade_maths", "both_parents")

scale_vars(input)
```

swap_treatment *Swap Treatment*

Description

Scrambles the elements of the vector and swaps a predefined number of elements. Afterwards, the vector is ordered according to the original ordering and returned.

Usage

```
swap_treatment(current_treatment,
               change,
               prev_index_list = NULL)
```

Arguments

current_treatment	a treatment vector to be changed.
change	number of elements that will be changed in the treatment vector.
prev_index_list	not used, needed for compatibility.

Value

Returns a new treatment vector.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

Examples

```
swap_treatment(current_treatment = c(0, 2, 0, 1, 1, 0, 2, 1, 1, 0),
               change = 2)
```

---

swap_treatment.optim *Swap Treatment (using optim)*

Description

This function calls swap_treatment, but since optim requires fn and gr to have the same parameters, it has an additional one.

Usage

```
swap_treatment.optim(current_treatment,
                     data = NULL,
                     evaluation_function = NULL,
                     swap_treatment_function = swap_treatment,
                     mse_weights = NULL,
                     change,
                     prev_index_list = NULL)
```

Arguments

current_treatment	a treatment vector to be changed.
data	the parameter is only needed for optim, it does not play any role.
evaluation_function	the parameter is only needed for optim, it does not play any role.
swap_treatment_function	the function used to create new treatments. Default is swap_treatment. Other options are swap_treatment_prev which, given a previous treatment, creates a new treatment assignment that takes the previous one into account.
change	number of elements that will be changed in the treatment vector.
prev_index_list	index list of the elements that can be changed. The current treatment vector may belong to a previous, unchangeable assignment.
mse_weights	the parameter is only needed for optim, it does not play any role.

Value

Returns a new treatment vector.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

See Also

optim

Examples

```
swap_treatment.optim(current_treatment = c(0, 2, 0, 1, 1, 0, 2, 1, 1, 0),
                     change = 2)

swap_treatment.optim(current_treatment = c(0, 2, 0, 1, 1, 0, 2, 1, 1, 0),
                     change = 2,
                     prev_index_list = c(1, 2, 3, 4))
```

---

swap_treatment_prev *Swap Treatment*

Description

Scrambles the elements of the vector and swaps a predefined number of elements. Afterwards, the vector is ordered according to the original ordering and returned.

Usage

```
swap_treatment_prev(current_treatment,
                    change,
                    prev_index_list)
```

Arguments

current_treatment	a treatment vector to be changed.
change	number of elements that will be changed in the treatment vector.
prev_index_list	index list of the elements that can be changed. The current treatment vector may belong to a previous, unchangeable assignment.

Value

Returns a new treatment vector.

Author(s)

Sebastian Schneider <sschneider@coll.mpg.de>; <sebastian@sebastianschneider.eu>, Giulia Baldini <giulia.baldini@uni-bonn.de>

Examples

```
swap_treatment_prev(current_treatment = c(0, 2, 0, 1, 1, 0, 2, 1, 1, 0),
                    change = 2,
                    prev_index_list = c(1, 2, 3, 4))
```

---

vector_gcd *Vector Greatest Common Divisor*

Description

Computes the GCD of a vector.

Usage

```
vector_gcd(vec)
```

Arguments

vec	a vector, for which we want to compute the GCD.

Value

The GCD of the elements in the given vector.

Examples

```
input = c(3,6,9)
vector_gcd(input)
```

Index