

Package ‘vecmatch’

September 8, 2026

Title Generalized Propensity Score Estimation and Matching for Multiple Groups

Version 1.4.0

Description Implements the Vector Matching algorithm to match multiple treatment groups based on previously estimated generalized propensity scores. The package includes tools for visualizing initial confounder imbalances, estimating treatment assignment probabilities using various methods, defining the common support region, performing matching across multiple groups, and evaluating matching quality. For more details, see Lopez and Gutman (2017) <[doi:10.1214/17-STS612](https://doi.org/10.1214/17-STS612)>.

License GPL (>= 3)

URL <https://github.com/Polymerase3/vecmatch>,
<https://polymerase3.github.io/vecmatch/>

BugReports <https://github.com/Polymerase3/vecmatch/issues>

Depends R (>= 3.5)

Imports chk, cli, foreach, ggplot2, ggpp, ggpubr, Matching, optmatch, productplots, progressr, rlang, rstatix, stats, utils, withr

Suggests brglm2, doFuture, doRNG, future, knitr, MASS, mclogit, nnet, parallel, rmarkdown, spelling, testthat (>= 3.0.0), VGAM

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Language en-US

VignetteBuilder knitr

NeedsCompilation no

Author Mateusz Kolek [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0001-6470-4830>>)

Maintainer Mateusz Kolek <mati.kolek13@gmail.com>

Repository CRAN

Date/Publication 2026-09-08 18:10:03 UTC

Contents

balqual	2
cancer	6
csregion	7
estimate_gps	8
get_select_params	11
make_opt_args	12
match_gps	15
mosaic	19
optimize_gps	21
raincloud	24
run_selected_matching	28
select_opt	29
Index	33

balqual	<i>Evaluate Matching Quality</i>
---------	----------------------------------

Description

The `balqual()` function evaluates the balance quality of a dataset after matching, comparing it to the original unbalanced dataset. It computes various summary statistics and provides an easy interpretation using user-specified cutoff values.

Usage

```
balqual(
  matched_data = NULL,
  formula = NULL,
  type = c("smd", "r", "var_ratio"),
  statistic = c("mean", "max"),
  cutoffs = NULL,
  round = 3
)
```

Arguments

<code>matched_data</code>	An object of class <code>matched</code> , generated by the <code>match_gps()</code> function. This object is essential for the <code>balqual()</code> function as it contains the final <code>data.frame</code> and attributes required to compute the quality coefficients.
<code>formula</code>	A valid R formula used to compute generalized propensity scores during the first step of the vector matching algorithm in <code>estimate_gps()</code> . This formula must match the one used in <code>estimate_gps()</code> .

type	A character vector specifying the quality metrics to calculate, provided in a vector created by the <code>c()</code>. The values fall into two groups. <code>smd</code>, <code>r</code> and <code>var_ratio</code> are balance metrics summarizing <i>pairwise</i> comparisons between the treatment levels. <code>desc_full</code> and <code>desc_reduced</code> instead describe the marginal distribution of each covariate within every treatment level, which makes it possible to directly compare the covariate distributions of the unmatched and the matched dataset. Note that they describe the covariates as they are given in the formula, whereas the balance metrics are computed on the model matrix: a factor is described once, by its levels, rather than once per dummy-coded column, and interaction terms are not described at all, since they are model terms rather than covariates. The balance metrics still cover them. Neither descriptive metric is part of the default, and both have to be requested explicitly, either on their own or together with the balance metrics, e.g. <code>type = c("smd", "desc_reduced")</code>. As <code>desc_reduced</code> is a subset of <code>desc_full</code>, the two are mutually exclusive. Possible values include: • <code>smd</code> - Calculates standardized mean differences (SMD) between groups, defined as the difference in means divided by the standard deviation of the treatment group (Rubin, 2001). • <code>r</code> - Computes Pearson's r coefficient using the Z statistic from the U-Mann-Whitney test. • <code>var_ratio</code> - Measures the dispersion differences between groups, calculated as the ratio of the larger variance to the smaller one. • <code>desc_full</code> - Computes standard descriptive statistics for every balancing variable, separately for each treatment level, before and after matching. Numeric covariates are summarized by the number of observations (N), mean, standard deviation, minimum, first quartile, median, third quartile, maximum, skewness and excess kurtosis, where skewness and excess kurtosis are the classical moment (type 1) estimators. Categorical covariates (factor, character and logical) are instead cross-tabulated, reporting the count and the percentage of every level within each treatment level, as means and quantiles are not meaningful for them. The two are printed as two separate tables. • <code>desc_reduced</code> - As <code>desc_full</code>, but the summary of the numeric covariates is restricted to the four most commonly reported statistics: minimum, mean, median and maximum. The cross-tabulation of the categorical covariates is unaffected. Useful when the full table is too wide to read comfortably in the console.
statistic	A character vector specifying the type of statistics used to summarize the quality metrics. Since quality metrics are calculated for all pairwise comparisons between treatment levels, they need to be aggregated for the entire dataset. • <code>max</code>: Returns the maximum values of the statistics defined in the <code>type</code> argument (as suggested by Lopez and Gutman, 2017). • <code>mean</code>: Returns the corresponding averages. To compute both, provide both names using the <code>c()</code> function. This argument is ignored by the <code>desc_full</code> and <code>desc_reduced</code> metrics, which are not based on pairwise comparisons and are therefore never aggregated across them.

cutoffs	A numeric vector with the same length as the number of balance metrics specified in the <code>type</code> argument. Defines the cutoffs for each corresponding metric, below which the dataset is considered balanced. If <code>NULL</code> , the default cutoffs are used: 0.1 for <code>smd</code> and <code>r</code> , and 2 for <code>var_ratio</code> . The descriptive metrics have no cutoff and are not counted here, so <code>type = c("smd", "desc_full")</code> still requires a single cutoff value.
round	A single non-negative integer specifying the number of decimal places to round the output to.

Value

If assigned to a name, returns a list of summary statistics of class `quality` containing:

- `quality_mean` - A data frame with the mean values of the statistics specified in the `type` argument for all balancing variables used in formula.
- `quality_max` - A data frame with the maximal values of the statistics specified in the `type` argument for all balancing variables used in formula.
- `quality_desc` - A data frame with the descriptive statistics of every balancing variable, laid out like the balance tables above: the statistics are the rows and the two matching stages are the `Before` and `After` columns. It holds the numeric and the categorical covariates in a single table, with the columns `Variable`, `Group`, `Type`, `Statistic`, `Before`, `After`, `Percent_Before` and `Percent_After`. `Type` is either `"continuous"` or `"categorical"`. For a continuous covariate, `Statistic` names the reported statistic - `N`, `Mean`, `SD`, `Min`, `Q1`, `Median`, `Q3`, `Max`, `Skewness` and `Kurtosis` for `desc_full`, or `N`, `Min`, `Mean`, `Median` and `Max` for `desc_reduced` - and the two `Percent` columns are `NA`. For a categorical covariate, `Statistic` names a level of that covariate, `Before` and `After` hold its counts, and the two `Percent` columns hold its share within the treatment level, computed separately for each matching stage. `NULL` unless one of the two descriptive metrics is included in the `type` argument. When printed, the counts and percentages of a categorical covariate are compressed into a single `N (%)` cell, with the percentage always shown to one decimal place.
- `perc_matched` - A single numeric value indicating the percentage of observations in the original dataset that were matched.
- `statistic` - A single string defining which statistic will be displayed in the console.
- `summary_head` - A summary of the matching process. If `max` is included in the `statistic`, it contains the maximal observed values for each variable; otherwise, it includes the mean values.
- `n_before` - The number of observations in the dataset before matching.
- `n_after` - The number of observations in the dataset after matching.
- `count_table` - A contingency table showing the distribution of the treatment variable before and after matching.

The `balqual()` function also prints a well-formatted table with the defined summary statistics for each variable in the formula to the console.

References

Rubin, D.B. Using Propensity Scores to Help Design Observational Studies: Application to the Tobacco Litigation. *Health Services & Outcomes Research Methodology* 2, 169–188 (2001). <https://doi.org/10.1023/A:1020363010465>

Michael J. Lopez, Roee Gutman "Estimation of Causal Effects with Multiple Treatments: A Review and New Ideas," *Statistical Science, Statist. Sci.* 32(3), 432-454, (August 2017)

See Also

`match_gps()` for matching the generalized propensity scores; `estimate_gps()` for the documentation of the formula argument.

Examples

```
# We try to balance the treatment variable in the cancer dataset based on age
# and sex covariates
data(cancer)

# Then we can estimate the generalized propensity scores
gps_cancer <- estimate_gps(formula(status ~ age * sex),
  cancer,
  method = "multinom",
  reference = "control",
  verbose_output = TRUE
)

# ... and drop observations based on the common support region...
csr_cancer <- csregion(gps_cancer)

# ... to match the samples using `match_gps()`
matched_cancer <- match_gps(csr_cancer,
  reference = "control",
  caliper = 1,
  kmeans_cluster = 5,
  kmeans_args = list(n.iter = 100),
  verbose_output = TRUE
)

# At the end we can assess the quality of matching using `balqual()`
balqual(
  matched_data = matched_cancer,
  formula = formula(status ~ age * sex),
  type = "smd",
  statistic = "max",
  round = 3,
  cutoffs = 0.2
)

# Adding `desc_full` to `type` additionally reports the descriptive
# statistics of every covariate per treatment level, before and after
# matching, which allows a direct comparison of the covariate distributions
```

```

balqual(
  matched_data = matched_cancer,
  formula = formula(status ~ age * sex),
  type = c("smd", "desc_full"),
  statistic = "max",
  round = 3,
  cutoffs = 0.2
)

# `desc_reduced` gives the same table restricted to the minimum, mean,
# median and maximum
balqual(
  matched_data = matched_cancer,
  formula = formula(status ~ age * sex),
  type = c("smd", "desc_reduced"),
  statistic = "max",
  round = 3,
  cutoffs = 0.2
)

```

cancer

Patients with Colorectal Cancer and Adenoma

Description

This is a synthetically generated dataset containing metadata for healthy individuals and patients diagnosed with colorectal cancer or adenomas. The primary purpose of this dataset in the context of matching is to balance the status groups across various covariates and achieve optimal matching quality.

Usage

```
data(cancer)
```

Format

A data frame (cancer) with 1,224 rows and 5 columns:

status Patient's health status, which can be one of the following: healthy, adenoma, crc_benign (benign colorectal carcinoma), or crc_malignant (malignant colorectal carcinoma).

sex Patient's biological sex, recorded as either M (male) or F (female).

age Patient's age, represented as a continuous numeric variable.

bmi Patient's Body Mass Index (BMI), represented as a continuous numeric variable.

smoker Smoking status of the patient, recorded as yes or no.

Source

Data generated artificially

csregion

*Filter the Data Based on Common Support Region***Description**

The `csregion()` function estimates the boundaries of the rectangular common support region, as defined by Lopez and Gutman (2017), and filters the matrix of generalized propensity scores based on these boundaries. The function returns a matrix of observations whose generalized propensity scores lie within the treatment group-specific boundaries.

Usage

```
csregion(gps_matrix, borders = "include", refit = TRUE)
```

Arguments

- | | |
|-------------------------|---|
| <code>gps_matrix</code> | An object of classes <code>gps</code> and <code>data.frame</code> (e.g., created by the <code>estimate_gps()</code> function). The first column corresponds to the treatment or grouping variable, while the other columns represent the treatment assignment probabilities calculated separately for each hypothetical treatment group. The number of columns should therefore be equal to the number of unique levels of the treatment variable plus one (for the treatment variable itself). The number of rows should correspond to the number of subjects for which generalized propensity scores were estimated. |
| <code>borders</code> | A character string specifying how to handle observations at the edges of the Common Support Region (CSR). Acceptable values are "include" and "exclude". If "include" is selected (default), observations with Generalized Propensity Scores (GPS) exactly equal to the CSR boundaries are retained for further analysis. This corresponds to a non-strict inequality: $\text{lower_bound} \leq \text{GPS} \leq \text{upper_bound}$. If "exclude" is selected, observations lying exactly on the CSR boundaries are removed. This corresponds to a strict inequality: $\text{lower_bound} < \text{GPS} < \text{upper_bound}$. Using "exclude" will typically result in a slightly smaller matched sample size compared to "include", but may be preferred for more conservative matching. |
| <code>refit</code> | Logical. If TRUE (default), the model used to estimate the GPS is refitted after excluding samples outside the common support region, using the same formula and method as in the original <code>estimate_gps()</code> call. If FALSE, the model is not refitted, but still only samples within the CSR are retained. Refitting is recommended, as suggested by Lopez and Gutman (2017). |

Value

A numeric matrix similar to the one returned by `estimate_gps()`, but with the number of rows reduced to exclude those observations that do not fit within the common support region (CSR) boundaries. The returned object also possesses additional attributes that summarize the calculation process of the CSR boundaries:

- `filter_matrix` - A logical matrix with the same dimensions as the `gps`-part of `gps_matrix`, indicating which treatment assignment probabilities fall within the CSR boundaries,
- `filter_vector` - A vector indicating whether each observation was kept (TRUE) or removed (FALSE), essentially a row-wise sum of `filter_matrix`,
- `csr_summary` - A summary of the CSR calculation process, including details of the boundaries and the number of observations filtered.
- `csr_data` - The original dataset used for the estimation of generalized propensity scores (`original_data` attribute of the `gps` object) filtered by the `filter_vector`

Examples

```
# We could estimate simples generalized propensity scores for the `iris`
# dataset
gps <- estimate_gps(Species ~ Sepal.Length, data = iris)

# And then define the common support region boundaries using `csregion()`
gps_csr <- csregion(gps)

# The additional information of the CSR-calculation process are
# accessible through the attributes described in the `*Value*` section
attr(gps_csr, "filter_matrix")
attr(gps_csr, "csr_summary")
attr(gps_csr, "csr_data")
```

estimate_gps

Calculate Treatment Allocation Probabilities

Description

`estimate_gps()` computes generalized propensity scores for treatment groups by applying a user-defined formula and method. It returns a matrix of GPS probabilities for each subject and treatment group

Usage

```
estimate_gps(
  formula,
  data = NULL,
  method = "multinom",
  link = NULL,
  reference = NULL,
  by = NULL,
  subset = NULL,
  ordinal_treat = NULL,
  fit_object = FALSE,
  verbose_output = FALSE,
  ...
)
```

Arguments

formula	a valid R formula, which describes the model used to calculating the probabilities of receiving a treatment. The variable to be balanced is on the left side, while the covariates used to predict the treatment variable are on the right side. To define the interactions between covariates, use *. For more details, refer to stats::formula() .
data	a data frame with columns specified in the formula argument.
method	a single string describing the model used for the calculation of generalized propensity scores. The default value is set to multinom. For available methods refer to the Details section below.
link	a single string; determines an alternative model for a method used for estimation. For available links, see Details.
reference	a single string describing one class from the treatment variable, referred to as the baseline category in the calculation of generalized propensity scores.
by	a single string with the name of a column, contained in the data argument. The dataset will be divided by the groups created by the grouping by variable and the calculation of the propensity scores will be carried out separately for each group. The results will then be merged and presented to the user as a single GPS matrix.
subset	a logical atomic vector of length equal to the number of rows in the data arguments. Allows to filter out observations from the further analysis, for which the value of the vector is equal to FALSE.
ordinal_treat	an atomic vector of the length equal to the length of unique levels of the treatment variable. Confirms, that the treatment variable is an ordinal variable and adjusts its levels, to the order of levels specified in the argument. Is a call to the function <code>factor(treat, levels = ordinal_treat, ordered = TRUE)</code> .
fit_object	a logical flag. If TRUE, the fitted object is returned instead of the GPS matrix.
verbose_output	a logical flag. If TRUE a more verbose version of the function is run and the output is printed out to the console.
...	additional arguments, that can be passed to the fitting function and are not controlled by the above arguments. For more details and examples refer to the Details section and documentations of corresponding functions.

Details

The main goal of the `estimate_gps()` function is to calculate the generalized propensity scores aka. treatment allocation probabilities. It is the first step in the workflow of the vector matching algorithm and is essential for the further analysis. The returned matrix of class `gps` can then be passed to the `csregion()` function to calculate the rectangular common support region boundaries and drop samples not eligible for the further analysis. The list of available methods operated by the `estimate_gps()` is provided below with a short description and function used for the calculations:

- `multinom` - multinomial logistic regression model [nnet::multinom\(\)](#)
- `vglm` - vector generalized linear model for multinomial data [VGAM::vglm\(\)](#),
- `brglm2` - bias reduction model for multinomial responses using the Poisson trick [brglm2::brmultinom\(\)](#),

- `mblogit` - baseline-category logit models `mclogit::mblogit()`.
- `polr` - ordered logistic or probit regression only for ordered factor variables from `MASS::polr()`. The method argument of the underlying `MASS::polr()` package function can be controlled with the `link` argument. Available options: `link = c("logistic", "probit", "loglog", "cloglog", "cauchit")`

Value

A data frame of class `gps` with the number of columns equal to the number of unique treatment variable levels plus one (for the treatment variable itself) and the number of rows equal to the number of subjects in the initial dataset. The original dataset used for estimation can be accessed as the `original_data` attribute.

See Also

`csregion()` for the calculation of common support region, `match_gps()` for the matching of generalized propensity scores

Examples

```
## Example 1: multinomial bias-reduced model (brglm2) on `airquality`
if (requireNamespace("brglm2", quietly = TRUE)) {
  library(brglm2)

  # Initial imbalance of means
  tapply(airquality$Wind, airquality$Month, mean, na.rm = TRUE)

  # Formula definition
  formula_air <- Month ~ Wind

  # Estimating the generalized propensity scores using brglm2
  gp_scores <- estimate_gps(
    formula_air,
    data = airquality,
    method = "brglm2",
    reference = "5",
    verbose_output = TRUE,
    control = brglm2::brglmControl(type = "MPL_Jeffreys")
  )

  # Filtering the observations outside the csr region
  gps_csr <- csregion(gp_scores)
  filter_which <- attr(gps_csr, "filter_vector")
  filtered_air <- airquality[filter_which, ]

  # Imbalance after csr
  tapply(filtered_air$Wind, filtered_air$Month, mean, na.rm = TRUE)

  # Visual inspection using raincloud
  raincloud(
    filtered_air,
```

```

    y = Wind,
    group = Month,
    significance = "t_test"
  )
}

## Example 2: ordered treatment, subset, by, and non-default link
if (requireNamespace("MASS", quietly = TRUE)) {
  library(MASS)

  # Prepare a clean subset of `airquality`
  aq <- subset(
    airquality,
    !is.na(Month) & !is.na(Wind) & !is.na(Temp)
  )

  # Grouping variable used in the `by` argument
  aq$Temp_group <- ifelse(
    aq$Temp > stats::median(aq$Temp, na.rm = TRUE),
    "high",
    "low"
  )

  # Explicit order for the (ordinal) treatment variable
  ord_month <- sort(unique(aq$Month))

  gps_ord <- estimate_gps(
    Month ~ Wind + Temp,
    data = aq,
    method = "polr",
    link = "probit",
    subset = NULL,
    by = "Temp_group",
    ordinal_treat = ord_month,
    reference = "5"
  )
}

```

get_select_params

Extract Parameter Grid for Selected Configurations

Description

Extract Parameter Grid for Selected Configurations

Usage

```
get_select_params(x, smd_group = NULL)
```

Arguments

- x** A `select_result` object returned by `select_opt()`.
- smd_group** Optional character vector of SMD groups (e.g. "0.10-0.15"). If provided, the returned `data.frame` is subsetted to these groups. If `NULL`, all rows are returned.

Value

A `data.frame` with the parameter combinations corresponding to the selected configurations.

Examples

```
# Define formula and set up optimization
formula_cancer <- formula(status ~ age * sex)
opt_args <- make_opt_args(cancer, formula_cancer, gps_method = "m1")
withr::with_seed(8252, {
  opt_results <- optimize_gps(
    data = cancer,
    formula = formula_cancer,
    opt_args = opt_args,
    n_iter = 2000
  )
})
# Select optimal combinations prioritizing SMD balance and matching in key
# groups
select_results <- select_opt(
  x = opt_results,
  smd_groups = list(
    c("adenoma", "control"),
    c("control", "crc_benign"),
    c("crc_malignant", "control")
  ),
  smd_variables = "age",
  smd_type = "max",
  perc_matched = c("adenoma", "crc_malignant")
)

# Extract the parameter grid from select_results for smd_group = "0.05-0.10"
get_select_params(select_results, smd_group = "0.05-0.10")
```

make_opt_args

Define the Optimization Parameter Space for Matching

Description

`make_opt_args()` creates an object of class "opt_args" that defines the parameter search space for `optimize_gps()`.

The function accepts **vectors of values** for each customizable argument involved in GPS estimation and matching. It computes the **Cartesian product** of all parameter combinations, which serves as the input search space for the random search algorithm used by `optimize_gps()`.

To ensure valid optimization, the data and formula arguments must exactly match those passed to `optimize_gps()`.

Usage

```
make_opt_args(
  data = NULL,
  formula,
  reference = NULL,
  gps_method = paste0("m", 1:10),
  matching_method = c("fullopt", "nnm"),
  caliper = seq(0.01, 10, 0.01),
  order = c("desc", "asc", "original", "random"),
  cluster = 2,
  replace = c(TRUE, FALSE),
  ties = c(TRUE, FALSE),
  ratio = 1,
  min_controls = 1,
  max_controls = 1
)
```

Arguments

data	A <code>data.frame</code> containing all variables referenced in formula. Must match the dataset used in <code>optimize_gps()</code> .
formula	A valid formula specifying the treatment variable (left-hand side) and covariates (right-hand side). Interaction terms can be included using <code>*</code> . Must match the formula used in <code>optimize_gps()</code> .
reference	A single string or vector of treatment group levels to be used as the reference (baseline) group in both GPS estimation and matching.
gps_method	A string or vector of strings specifying GPS estimation methods. Allowed values are "m1" to "m10". See <i>Details</i> below.
matching_method	A string or vector of strings specifying the matching method(s) to evaluate. Currently supported options are "nnm" and "fullopt". See match_gps() .
caliper	A numeric value or vector of values specifying caliper widths (i.e., maximum allowed GPS distance for matching). Same as in match_gps() , but allows multiple values.
order	A string or vector of strings indicating the sorting order of logit-transformed GPS values before matching. Options are: "desc": sort from highest to lowest (default), "asc": sort from lowest to highest, "original": keep original order,

- "random": randomize order (use `set.seed()` for reproducibility).

cluster	An integer or vector of integers specifying the number of clusters for k-means clustering (if applicable).
replace	Logical value or vector of logicals indicating whether to allow matching with replacement. Same meaning as in <code>match_gps()</code> , but supports multiple settings.
ties	Logical value or vector of logicals defining how ties should be handled during nearest-neighbor matching.
ratio	A numeric value or vector specifying the ratio of control to treated units for matching (used in "nnm").
min_controls	A scalar or vector specifying the minimum number of controls to be matched to each treated unit (used in "fullopt").
max_controls	A scalar or vector specifying the maximum number of controls to be matched to each treated unit (used in "fullopt").

Details

The returned object is of class "opt_args" and is intended to be passed directly to `optimize_gps()`. Internally, the function calculates the full Cartesian product of all supplied parameter values and validates the structure of each.

The `gps_method` argument must contain one or more of the following codes:

gps_method	Method	Link Function
"m1"	multinom	generalized_logit
"m2"	polr	logistic
"m3"	polr	probit
"m4"	polr	loglog
"m5"	polr	cloglog
"m6"	polr	cauchit
"m7"	vglm	multinomial_logit
"m8"	vglm	reduced_rank_ml
"m9"	brglm2	baseline_category_logit
"m10"	mblogit	baseline_category_logit

The object includes a custom S3 `print()` method that displays:

- A summary table of all allowed values for each optimization parameter,
- The total number of unique parameter combinations (i.e., the size of the search space).

Value

An object of class "opt_args", containing all valid parameter combinations to be sampled by `optimize_gps()`. Use `print()` to explore the defined search space.

See Also

`optimize_gps()`, `match_gps()`, `estimate_gps()`

Examples

```
# Create search space with multiple values for GPS and matching
make_opt_args(
  data = cancer,
  formula = formula(status ~ age * sex),
  gps_method = c("m1", "m2", "m9"),
  matching_method = c("nnm", "fullopt"),
  caliper = c(0.1, 0.2),
  order = c("desc", "random"),
  reference = "control"
)
```

match_gps

Match the Data Based on Generalized Propensity Scores

Description

The `match_gps()` function performs sample matching based on generalized propensity scores (GPS). It applies a k-means clustering step to the GPS to partition the data into clusters and then matches all treatment groups within each cluster. This yields balanced comparisons across treatment levels while preserving the GPS structure. To our knowledge, `vecmatch` provides the first implementation in R of the vector-matching algorithm described by Lopez and Gutman (2017).

Usage

```
match_gps(
  csmatrix = NULL,
  method = "nnm",
  caliper = 0.2,
  reference = NULL,
  ratio = NULL,
  replace = NULL,
  order = NULL,
  ties = NULL,
  min_controls = NULL,
  max_controls = NULL,
  kmeans_args = NULL,
  kmeans_cluster = 5,
  verbose_output = FALSE,
  ...
)
```

Arguments

<code>csmatrix</code>	An object of class <code>gps</code> and/or <code>csr</code> representing a data frame of generalized propensity scores. The first column must be the treatment variable, with additional attributes describing the calculation of the common support region and
-----------------------	---

the estimation of generalized propensity scores. It is crucial that the common support region was calculated using the `csregion()` function to ensure compatibility.

method	A single string specifying the matching method to use. The default is "nnm", which applies the k-nearest neighbors matching algorithm. See the Details section for a full list of available methods.
caliper	A numeric value specifying the caliper width, which defines the allowable range within which observations can be matched. It is expressed as a percentage of the standard deviation of the logit-transformed generalized propensity scores. To perform matching without a caliper, set this parameter to a very large value. For exact matching, set <code>caliper = 0</code> and enable the exact option by setting it to <code>TRUE</code> .
reference	A single string specifying the exact level of the treatment variable to be used as the reference in the matching process. All other treatment levels will be matched to this reference level. Ideally, this should be the control level. If no natural control is present, avoid selecting a level with extremely low or high covariate or propensity score values. Instead, choose a level with covariate or propensity score distributions that are centrally positioned among all treatment groups to maximize the number of matches.
ratio	A scalar for the number of matches which should be found for each control observation. The default is one-to-one matching. Only available for the methods "nnm" and "pairopt".
replace	Logical value indicating whether matching should be done with replacement. If <code>FALSE</code> , the order of matches generally matters. Matches are found in the same order as the data is sorted. Specifically, the matches for the first observation will be found first, followed by those for the second observation, and so on. Matching without replacement is generally not recommended as it tends to increase bias. However, in cases where the dataset is large and there are many potential matches, setting <code>replace = FALSE</code> often results in a substantial speedup with negligible or no bias. Only available for the method "nnm"
order	A string specifying the order in which logit-transformed GPS values are sorted before matching. The available options are: • "desc" – sorts GPS values from highest to lowest (default). • "asc" – sorts GPS values from lowest to highest. • "original" – preserves the original order of GPS values. • "random" – randomly shuffles GPS values. To generate different random orders, set a seed using <code>set.seed()</code>.
ties	A logical flag indicating how tied matches should be handled. Available only for the "nnm" method, with a default value of <code>FALSE</code> (all tied matches are included in the final dataset, but only unique observations are retained). For more details, see the <code>ties</code> argument in <code>Matching::Matchby()</code> .
min_controls	The minimum number of treatment observations that should be matched to each control observation. Available only for the "fullopt" method. For more details, see the <code>min.controls</code> argument in <code>optmatch::fullmatch()</code> .

max_controls	The maximum number of treatment observations that can be matched to each control observation. Available only for the "fullopt" method. For more details, see the max_controls argument in <code>optmatch::fullmatch()</code> .
kmeans_args	A list of arguments to pass to <code>stats::kmeans</code> . These arguments must be provided inside a <code>list()</code> in the paired name = value format.
kmeans_cluster	An integer specifying the number of clusters to pass to <code>stats::kmeans</code> .
verbose_output	a logical flag. If TRUE a more verbose version of the function is run and the output is printed out to the console.
...	Additional arguments to be passed to the matching function.

Details

Propensity score matching can be performed using various matching algorithms. Lopez and Gutman (2017) do not explicitly specify the matching algorithm used, but it is assumed they applied the commonly used k-nearest neighbors matching algorithm, implemented as `method = "nnm"`. However, this algorithm can sometimes be challenging to use, especially when treatment and control groups have unequal sizes. When `replace = FALSE`, the number of matches is strictly limited by the smaller group, and even with `replace = TRUE`, the results may not always be satisfactory. To address these limitations, we have implemented an additional matching algorithm to maximize the number of matched observations within a dataset.

The available matching methods are:

- "nnm" – classic k-nearest neighbors matching, implemented using `Matching::Matchby()`. The tunable parameters in `match_gps()` are `caliper`, `ratio`, `replace`, `order`, and `ties`. Additional arguments can be passed to `Matching::Matchby()` via the `...` argument.
- "fullopt" – optimal full matching algorithm, implemented with `optmatch::fullmatch()`. This method calculates a discrepancy matrix to identify all possible matches, often optimizing the percentage of matched observations. The available tuning parameters are `caliper`, `min_controls`, and `max_controls`.
- "pairmatch" – optimal 1:1 and 1:k matching algorithm, implemented using `optmatch::pairmatch()`, which is actually a wrapper around `optmatch::fullmatch()`. Like "fullopt", this method calculates a discrepancy matrix and finds matches that minimize its sum. The available tuning parameters are `caliper` and `ratio`.

Each method only accepts the tuning parameters listed above. Arguments that do not apply to the selected method are dropped, and a warning naming them is emitted only when they were explicitly supplied. Tuning parameters that are left at their default are filled in silently, so a call relying on the defaults runs without any warnings.

Value

A data.frame similar to the one provided as the data argument in the `estimate_gps()` function, containing the same columns but only the observations for which a match was found. The returned object includes two attributes, accessible with the `attr()` function:

- `original_data`: A data.frame with the original data returned by the `csregion()` or `estimate_gps()` function, after the estimation of the csr and filtering out observations not within the csr.
- `matching_filter`: A logical vector indicating which rows from `original_data` were included in the final matched dataset.

References

Michael J. Lopez, Roee Gutman "Estimation of Causal Effects with Multiple Treatments: A Review and New Ideas," Statistical Science, Statist. Sci. 32(3), 432-454, (August 2017)

See Also

`estimate_gps()` for the calculation of generalized propensity scores; `MatchIt::matchit()`, `optmatch::fullmatch()` and `optmatch::pairmatch()` for the documentation of the matching functions; `stats::kmeans()` for the documentation of the k-Means algorithm.

Examples

```
# Step 1.) Estimation of the generalized propensity scores
gp_scores <- estimate_gps(
  formula(status ~ age * sex),
  data = cancer,
  method = "multinom",
  reference = "control",
  verbose_output = TRUE
)

# Step 2.) Defining the common support region
gps_csr <- csregion(gp_scores)

# Example 1: optimal full matching ("fullopt") with explicit tuning params
matched_cancer_full <- match_gps(
  gps_csr,
  caliper = 0.25,
  reference = "control",
  method = "fullopt",
  kmeans_cluster = 2,
  kmeans_args = list(
    iter.max = 200,
    algorithm = "Forgy"
  ),
  min_controls = 0,
  max_controls = Inf,
  order = "desc",
  verbose_output = TRUE
)

# Example 2: nearest neighbour matching ("nnm") demonstrating ratio /
# replace / order / ties
matched_cancer_nnm <- match_gps(
  gps_csr,
  caliper = 0.25,
  reference = "control",
  method = "nnm",
  ratio = 2,
  replace = TRUE,
  order = "asc",
  ties = TRUE,
```

```

    verbose_output = FALSE
  )

```

mosaic

Plot the Distribution of Categorical Covariates

Description

The `mosaic()` function generates imbalance plots for contingency tables with up to three variables. Frequencies in the contingency table are represented as tiles (rectangles), with each tile's size proportional to the frequency of the corresponding group within the entire dataset. The x-axis scale remains fixed across mosaic plots, enabling straightforward comparisons of imbalance across treatment groups.

Usage

```

mosaic(
  data = NULL,
  y = NULL,
  group = NULL,
  facet = NULL,
  ncol = 1,
  group_counts = FALSE,
  group_counts_size = 4,
  significance = FALSE,
  plot_name = NULL,
  overwrite = FALSE,
  ...
)

```

Arguments

<code>data</code>	A non-empty <code>data.frame</code> containing at least one numeric column, as specified by the <code>y</code> argument. This argument must be provided and does not have a default value.
<code>y</code>	A single string or unquoted symbol representing the name of a numeric column in the data. In the vector matching workflow, it is typically a numeric covariate that requires balancing.
<code>group</code>	A single string or unquoted symbol representing the name of a factor or character column in data. In <code>raincloud()</code> plots, the groups specified by <code>group</code> argument will be distinguished by separate fill and color aesthetics. For clarity, it is recommended to plot fewer than 10 groups, though there is no formal limit.
<code>facet</code>	A single string or unquoted symbol representing the name of a variable in data to facet by. This argument is used in a call to <code>ggplot2::facet_wrap()</code> , creating separate distribution plots for each unique group in the facet variable.

<code>ncol</code>	A single integer giving the number of columns in the facet layout. When facet is not NULL, <code>ncol</code> must be between 1 and the number of unique categories in the facet variable; values outside this range result in an error. This argument is ignored when facet is NULL.
<code>group_counts</code>	A logical flag. If TRUE, the sizes of the groups will be displayed inside the rectangles in the plot created by the <code>mosaic()</code> function. If FALSE (default), the group sizes will not be shown.
<code>group_counts_size</code>	A single numeric value that specifies the size of the group count labels in millimeters ('mm'). This value is passed to the size argument of <code>ggplot2::geom_text()</code> .
<code>significance</code>	A logical flag; defaults to FALSE. When TRUE, a Chi-squared test of independence is performed on the contingency table of <code>y</code> and <code>group</code> . Note that <code>group</code> must be specified for the test to be calculated. If facet is provided, the significance is assessed separately for each facet subgroup. Additionally, the function calculates standardized Pearson residuals (differences between observed and expected counts) and fills mosaic plot cells based on the level of partial significance for each cell.
<code>plot_name</code>	A string specifying a valid file name or path for the plot. If set to NULL, the plot is displayed to the current graphical device but not saved locally. If a valid name with <code>.png</code> or <code>.pdf</code> extension is provided, the plot is saved locally. Users can also include a subdirectory in <code>plot_name</code> . Ensure the file path follows the correct syntax for your operating system.
<code>overwrite</code>	A logical flag (default FALSE) that is evaluated only if the <code>save.name</code> argument is provided. If TRUE, the function checks whether a plot with the same name already exists. If it does, the existing plot will be overwritten. If FALSE and a plot with the same name exists, an error is thrown. If no such plot exists, the plot is saved normally.
<code>...</code>	Additional arguments to pass to <code>rstatix::chisq_test</code> when <code>significance = TRUE</code> .

Value

A ggplot object representing the contingency table of `y` and `group` as a mosaic plot, optionally grouped by facet if specified.

Examples

```
## Example: Creating a Mosaic Plot of the Titanic Dataset
## This plot visualizes survival rates by gender across different passenger
## classes. By setting `significance = TRUE`, you can highlight statistically
## significant differences within each rectangle of the mosaic plot.
library(ggplot2)

# Load Titanic dataset and convert to data frame
titanic_df <- as.data.frame(Titanic)

# Expand the dataset by repeating rows according to 'Freq'
titanic_long <- titanic_df[rep(
```

```

    seq_len(nrow(titanic_df)),
    titanic_df$Freq
  ), ]

# Remove the 'Freq' column as it is no longer needed
titanic_long$Freq <- NULL

# Plot the data using mosaic() and modify the result using additional ggplot2
# functions
p <- mosaic(
  data = titanic_long,
  y = Survived,
  group = Sex,
  facet = Class,
  ncol = 2,
  significance = TRUE,
  plot_name = NULL
)

p <- p +
  theme_minimal()

p

```

optimize_gps

Optimize the Matching Process via Random Search

Description

The `optimize_gps()` function performs a random search to identify optimal combinations of parameters for the `match_gps()` and `estimate_gps()` functions. The goal is to maximize the percentage of matched samples (`perc_matched`) while minimizing the maximum standardized mean difference (`smd`), thereby improving the overall balance of covariates across treatment groups. The function supports parallel execution through the `foreach` and `future` packages, enabling multi-threaded computation to accelerate the optimization process, particularly when dealing with large datasets or complex parameter spaces.

Usage

```

optimize_gps(
  data = NULL,
  formula,
  ordinal_treat = NULL,
  n_iter = 1000,
  opt_args = NULL
)

```

Arguments

<code>data</code>	A <code>data.frame</code> containing all variables specified in the <code>formula</code> argument. If <code>opt_args</code> is used, the data provided within <code>opt_args</code> must match this input exactly.
<code>formula</code>	A valid formula object used to estimate the generalized propensity scores (GPS). The treatment variable appears on the left-hand side, and covariates on the right-hand side. Interactions can be specified using <code>*</code> . See <code>stats::formula()</code> and <code>estimate_gps()</code> for more details. If <code>opt_args</code> is provided, the formula within it must be identical to this argument.
<code>ordinal_treat</code>	An atomic vector defining the ordered levels of the treatment variable. This confirms the variable is ordinal and adjusts its levels accordingly using <code>factor(treat, levels = ordinal_treat, ordered = TRUE)</code> . It is passed directly to <code>estimate_gps()</code> . If <code>NULL</code> , ordinal GPS estimation methods such as <code>polr</code> will be excluded from the optimization. See <code>estimate_gps()</code> for details.
<code>n_iter</code>	Integer. Number of unique parameter combinations to evaluate during optimization. Higher values generally yield better results but increase computation time. For large datasets or high-dimensional parameter spaces, increasing <code>n_iter</code> is recommended. When using parallel processing (<code>n_cores > 1</code>), performance gains become more apparent with larger <code>n_iter</code> . Too many cores with too few iterations may introduce overhead and reduce efficiency.
<code>opt_args</code>	An object of class <code>"opt_args"</code> containing optimization parameters and argument settings. Use <code>make_opt_args()</code> to create this object. It specifies the search space for the GPS estimation and matching procedure.

Details

The output is an S3 object of class `best_opt_result`. Its core component is a `data.frame` containing the parameter settings for the best-performing models, grouped and ranked based on their balance quality.

Optimization results are categorized into seven bins based on the maximum standardized mean difference (SMD):

- 0.00-0.05
- 0.05-0.10
- 0.10-0.15
- 0.15-0.20
- 0.20-0.25
- 0.25-0.30
- Greater than 0.30

Within each SMD group, the parameter combination(s) achieving the highest `perc_matched` (i.e., percentage of matched samples) is selected. In cases where multiple combinations yield identical `smd` and `perc_matched`, all such results are retained. Combinations where matching failed or GPS estimation did not converge will return `NA` in the result columns (e.g., `perc_matched`, `smd`).

The returned `data.frame` includes the following columns (depending on the number of treatment levels):

- `iter_ID`: Unique identifier for each parameter combination
- `method_match`: Matching method used in `match_gps()`, e.g., "nnm" or "fullopt"
- `caliper`: Caliper value used in `match_gps()`
- `order`: Ordering of GPS scores prior to matching
- `kmeans_cluster`: Number of k-means clusters used
- `replace`: Whether replacement was used in matching (nnm only)
- `ties`: Tie-breaking rule in nearest-neighbor matching (nnm only)
- `ratio`: Control-to-treated ratio for nnm
- `min_controls`, `max_controls`: Minimum and maximum controls for fullopt
- `reference`: Reference group used in both `estimate_gps()` and `match_gps()`
- `perc_matched`: Percentage of matched samples (from `balqual()`)
- `smd`: Maximum standardized mean difference (from `balqual()`)
- `p_{group_name}`: Percent matched per treatment group (based on group sample size)
- `method_gps`: GPS estimation method used (from `estimate_gps()`)
- `link`: Link function used in GPS model
- `smd_group`: SMD range category for the row

The resulting `best_opt_result` object also includes a custom `summary()` method that summarizes:

- The number of optimal parameter sets per SMD group
- Their associated SMD and match rates
- Total combinations tested
- Total runtime of the optimization loop

Value

An S3 object of class `best_opt_result`. The core component is a `data.frame` containing the parameter combinations and results of the optimization procedure. You can access it using `attr(result, "opt_results")` or by calling `View(result)`, where `result` is your `best_opt_result` object.

The object contains the following custom attributes:

- `opt_results`: A `data.frame` of optimization results. Each row corresponds to a unique parameter combination tested. For a complete description of columns, see the **Details** section.
- `optimization_time`: Time (in seconds) taken by the optimization loop (i.e., the core for-loop that evaluates combinations). This does **not** include the time needed for GPS estimation, pre-processing, or merging of results after loop completion. On large datasets, these excluded steps can still be substantial.
- `combinations_tested`: Total number of unique parameter combinations evaluated during optimization.
- `smd_results`: A detailed table of standardized mean differences (SMDs) for all pairwise treatment group comparisons and for all covariates specified in the formula. This is used by the `select_opt()` function to filter optimal models based on covariate-level balance across groups.

- `treat_names`: A character vector with the names of the unique treatment groups.
- `model_covs`: A character vector listing the model covariates (main effects and interactions) used in the formula. These names correspond to the variables shown in the `smd_results` table.

Examples

```
# Define formula for GPS estimation and matching
formula_cancer <- formula(status ~ age * sex)

# Set up the optimization parameter space
opt_args <- make_opt_args(cancer, formula_cancer, gps_method = "m1")

# Run optimization with 2000 random parameter sets and a fixed seed

withr::with_seed(
  8252,
  {
    optimize_gps(
      data = cancer,
      formula = formula_cancer,
      opt_args = opt_args,
      n_iter = 2000
    )
  }
)
```

raincloud

Examine the Imbalance of Continuous Covariates

Description

The `raincloud()` function allows to generate distribution plots for continuous data in an easy and uncomplicated way. The function is based on the `ggplot2` package, which must already be preinstalled. Raincloud plots consist of three main elements:

- Distribution plots, specifically violin plots with the mean values and standard deviations of respective groups,
- Jittered point plots depicting the underlying distribution of the data in the rawest form,
- Boxplots, summarizing the most important statistics of the underlying distribution.

Usage

```
raincloud(
  data = NULL,
  y = NULL,
  group = NULL,
```

```

    facet = NULL,
    ncol = 1,
    significance = NULL,
    sig_label_size = 2L,
    sig_label_color = FALSE,
    smd_type = "mean",
    density_scale = "area",
    limits = NULL,
    jitter = 0.1,
    alpha = 0.4,
    plot_name = NULL,
    overwrite = FALSE,
    ...
)

```

Arguments

<code>data</code>	A non-empty <code>data.frame</code> containing at least one numeric column, as specified by the <code>y</code> argument. This argument must be provided and does not have a default value.
<code>y</code>	A single string or unquoted symbol representing the name of a numeric column in the data. In the vector matching workflow, it is typically a numeric covariate that requires balancing.
<code>group</code>	A single string or unquoted symbol representing the name of a factor or character column in data. In <code>raincloud()</code> plots, the groups specified by <code>group</code> argument will be distinguished by separate fill and color aesthetics. For clarity, it is recommended to plot fewer than 10 groups, though there is no formal limit.
<code>facet</code>	A single string or unquoted symbol representing the name of a variable in data to facet by. This argument is used in a call to <code>ggplot2::facet_wrap()</code> , creating separate distribution plots for each unique group in the <code>facet</code> variable.
<code>ncol</code>	A single integer giving the number of columns in the facet layout. When <code>facet</code> is not <code>NULL</code> , <code>ncol</code> must be between 1 and the number of unique categories in the <code>facet</code> variable; values outside this range result in an error. This argument is ignored when <code>facet</code> is <code>NULL</code> .
<code>significance</code>	A single string specifying the method for calculating p-values in multiple comparisons between groups defined by the <code>group</code> argument. Significant comparisons are represented by bars connecting the compared groups on the left side of the boxplots. Note that if there are many significant tests, the plot size may adjust accordingly. For available methods refer to the <i>Details</i> section. If the <code>significance</code> argument is not <code>NULL</code> , standardized mean differences (SMDs) are also calculated and displayed on the right side of the jittered point plots.
<code>sig_label_size</code>	A single integer between 1 and 20 specifying the size of the significance and standardized mean difference (SMD) labels shown on the left and right side of the plot.
<code>sig_label_color</code>	Logical flag. If <code>FALSE</code> (default), significance and SMD bars and text are displayed in the default color (black). If <code>TRUE</code> , colors are applied dynamically

	based on value: nonsignificant tests and SMD values below 0.10 are displayed in green, while significant tests and SMD values of 0.10 or higher are displayed in red.
smd_type	A single string indicating the type of effect size to calculate and display on the left side of the jittered point plots: • mean - Cohen's d is calculated, • median - the Wilcoxon effect size (r) is calculated based on the Z statistic extracted from the Wilcoxon test.
density_scale	Character(1). Scaling method for the violin density. • "area": all violins have the same total area (default). • "count": violin areas are proportional to the number of observations. • "width": all violins have the same maximum height (width when they are placed horizontally), regardless of sample size.
limits	A numeric atomic vector of length two, specifying the y axis limits in the distribution plots. The first element sets the minimum value, and the second sets the maximum. This vector is passed to the <code>ggplot2::xlim()</code> function to adjust the axis scale.
jitter	A single numeric value between 0 and 1 that controls the amount of jitter applied to points in the <code>ggplot2::geom_jitter()</code> plots. Higher values of the jitter argument produce more jittered plot. It's recommended to keep this value low, as higher jitter can make the plot difficult to interpret.
alpha	A single numeric value between 0 and 1 that controls the transparency of the density plots, boxplots, and jittered point plots. Lower values result in higher transparency. It is recommended to keep this value relatively high to maintain the interpretability of the plots when using the group argument, as excessive transparency may cause overlap between groups, making it difficult to distinguish them visually.
plot_name	A string specifying a valid file name or path for the plot. If set to NULL, the plot is displayed to the current graphical device but not saved locally. If a valid name with .png or .pdf extension is provided, the plot is saved locally. Users can also include a subdirectory in plot_name. Ensure the file path follows the correct syntax for your operating system.
overwrite	A logical flag (default FALSE) that is evaluated only if the save.name argument is provided. If TRUE, the function checks whether a plot with the same name already exists. If it does, the existing plot will be overwritten. If FALSE and a plot with the same name exists, an error is thrown. If no such plot exists, the plot is saved normally.
...	Additional arguments passed to the function for calculating p-values when the significance argument is specified. For available functions associated with different significance methods, please refer to the <i>Details</i> section and consult the documentation for the relevant functions in the <i>rstatix</i> package.

Details

Available methods for the argument significance are:

- "t_test" - Performs a pairwise comparison using the two-sample t-test, with the default Holm adjustment for multiple comparisons. This test assumes normally distributed data and equal variances. The adjustment can be modified via the `p.adjust.method` argument. The test is implemented via `rstatix::pairwise_t_test()`
- "dunn_test" - Executes Dunn's test for pairwise comparisons following a Kruskal-Wallis test. It is a non-parametric alternative to the t-test when assumptions of normality or homogeneity of variances are violated. Implemented via `rstatix::dunn_test()`.
- "tukeyHSD_test" - Uses Tukey's Honest Significant Difference (HSD) test for pairwise comparisons between group means. Suitable for comparing all pairs when the overall ANOVA is significant. The method assumes equal variance between groups and is implemented via `rstatix::tukey_hsd()`.
- "games_howell_test" - A post-hoc test used after ANOVA, which does not assume equal variances or equal sample sizes. It's particularly robust for data that violate homogeneity of variance assumptions. Implemented via `rstatix::games_howell_test()`.
- "wilcoxon_test" - Performs the Wilcoxon rank-sum test (also known as the Mann-Whitney U test) for non-parametric pairwise comparisons. Useful when data are not normally distributed. Implemented via `rstatix::pairwise_wilcox_test()`.

Value

A ggplot object representing the distribution of the y variable across the levels of the group and facet variables in data.

See Also

`mosaic()` which summarizes the distribution of discrete data

Examples

```
## Example: Creating a raincloud plot for the ToothGrowth dataset.
## This plot visualizes the distribution of the `len` variable by
## `dose` (using different colors) and facets by `supp`. Group
## differences by `dose` are calculated using a `t_test`, and standardized
## mean differences (SMDs) are displayed through jittered points.
library(ggplot2)
library(ggpubr)

p <- raincloud(ToothGrowth, len, dose, supp,
  significance = "t_test",
  jitter = 0.15,
  alpha = 0.4
)

## As `p` is a valid `ggplot` object, we can manipulate its
## characteristics using the `ggplot2` or `ggpubr` packages
## to create a publication-grade plot:
p <- p +
  theme_classic2() +
  theme(
    axis.line.y = element_blank(),
```

```

    axis.ticks.y = element_blank()
  ) +
  guides(fill = guide_legend("Dose [mg]")) +
  ylab("Length [cm]")

p

## Example: demonstrate `limits` and `plot_name` (no file is saved
## because `plot_name = NULL`)
p2 <- raincloud(
  ToothGrowth, len, dose, supp,
  significance = "t_test",
  limits = c(0, 40),
  plot_name = NULL
)

p2

```

run_selected_matching *Rerun GPS Estimation and Matching for a Selected Configuration*

Description

Rerun GPS Estimation and Matching for a Selected Configuration

Usage

```
run_selected_matching(x, data, formula, smd_group = NULL, row = 1L, ...)
```

Arguments

x	A select_result object returned by select_opt().
data	Data frame used in the original optimization (pass it the same way as in your original analysis, e.g. data = cancer).
formula	Model formula used for GPS estimation (e.g. formula_cancer).
smd_group	Optional SMD bin to filter on (e.g. "0.10-0.15"). If NULL, no filtering is applied.
row	Integer index of the row (after optional filtering by smd_group) to use. Defaults to 1.
...	Extra args forwarded to match_gps().

Value

The result of match_gps().

Examples

```
# Define formula and set up optimization
formula_cancer <- formula(status ~ age * sex)
opt_args <- make_opt_args(cancer, formula_cancer, gps_method = "m1")
withr::with_seed(8252, {
  opt_results <- optimize_gps(
    data = cancer,
    formula = formula_cancer,
    opt_args = opt_args,
    n_iter = 2000
  )
})
# Select optimal combinations prioritizing SMD balance and matching in key
# groups
select_results <- select_opt(
  x = opt_results,
  smd_groups = list(
    c("adenoma", "control"),
    c("control", "crc_benign"),
    c("crc_malignant", "control")
  ),
  smd_variables = "age",
  smd_type = "max",
  perc_matched = c("adenoma", "crc_malignant")
)

# Extract the parameter grid from select_results for smd_group = "0.05-0.10"
get_select_params(select_results, smd_group = "0.05-0.10")

# Rerun the analysis
```

select_opt

Select Optimal Parameter Combinations from Optimization Results

Description

`select_opt()` is a helper function to filter and prioritize results from `optimize_gps()` based on the specific goals of a study. Depending on the research design, certain pairwise comparisons or treatment groups may be more important than others. For example:

- You may want to prioritize matching between a specific groups (e.g. specific disease vs. controls), while ignoring other group comparisons during SMD evaluation.
- You may wish to retain as many samples as possible from a critical group or set of groups, regardless of matching rates in other groups.

This function enables targeted selection of optimal parameter combinations by:

- Evaluating SMDs for specific pairwise treatment group comparisons,

- Selecting key covariates to assess balance,
- Prioritizing matched sample size in selected treatment groups.

By combining these criteria, `select_opt()` allows you to tailor the optimization output to your study's focus - whether it emphasizes covariate balance in targeted group comparisons or maximizing sample retention for specific subgroups.

Usage

```
select_opt(
  x,
  smd_groups = NULL,
  smd_variables = NULL,
  smd_type = c("mean", "max"),
  perc_matched = NULL
)
```

Arguments

<code>x</code>	An object of class <code>best_opt_result</code> , produced by the <code>optimize_gps()</code> function.
<code>smd_groups</code>	A list of pairwise comparisons (as character vectors of length 2) specifying which treatment group comparisons should be prioritized in SMD evaluation. Each element must be a valid pair of treatment levels. If <code>NULL</code> , all pairwise comparisons are used. Example: <code>list(c("adenoma", "crc_malignant"), c("controls", "adenoma"))</code>
<code>smd_variables</code>	A character vector of covariate names to include in the SMD evaluation. Must match variables listed in <code>attr(x, "model_covs")</code> .
<code>smd_type</code>	A character string ("mean" or "max"), defining how to aggregate SMDs across covariates and comparisons. "max" selects combinations with the lowest maximum SMD; "mean" uses the average SMD.
<code>perc_matched</code>	A character vector of treatment levels for which the matching rate should be maximized. If <code>NULL</code> , overall <code>perc_matched</code> is used. If specified, only the sum of matching percentages for the listed groups is used for selection within each SMD category.

Details

Optimization results are grouped into bins based on the **maximum SMD** observed for each parameter combination. These bins follow the same structure as in `optimize_gps()`:

- 0.00-0.05
- 0.05-0.10
- 0.10-0.15
- 0.15-0.20
- 0.20-0.25
- 0.25-0.30

- 0.30-0.35
- 0.35-0.40
- 0.40-0.45
- 0.45-0.50
- more than 0.50

Within each bin, models are first filtered based on their aggregated SMD across the specified `smd_groups` and `smd_variables`, using the method defined by `smd_type`. Then, among the remaining models, the best-performing one(s) are selected based on the percentage of matched samples - either overall or in the specified treatment groups (`perc_matched`).

Value

An S3 object of class `select_result`, containing the filtered and prioritized optimization results. The object includes:

- A `data.frame` with selected parameter combinations and performance metrics.
- **Attribute** `param_df`: A `data.frame` with full parameter specifications (`iter_ID`, GPS/matching parameters, etc.), useful for manually refitting or reproducing results.

The object also includes a custom `print()` method that summarizes:

- Number of selected combinations per SMD bin
- Corresponding aggregated SMD (mean or max)
- Overall or group-specific percentage matched

Examples

```
# Define formula and set up optimization
formula_cancer <- formula(status ~ age * sex)
opt_args <- make_opt_args(cancer, formula_cancer, gps_method = "m1")
withr::with_seed(8252, {
  opt_results <- optimize_gps(
    data = cancer,
    formula = formula_cancer,
    opt_args = opt_args,
    n_iter = 2000
  )
})
# Select optimal combinations prioritizing SMD balance and matching in key
# groups
select_opt(
  x = opt_results,
  smd_groups = list(
    c("adenoma", "control"),
    c("control", "crc_benign"),
    c("crc_malignant", "control")
  ),
  smd_variables = "age",
  smd_type = "max",
```

```
perc_matched = c("adenoma", "crc_malignant")  
)
```

Index

- * **datasets**
 - cancer, [6](#)
- balqual, [2](#)
- balqual(), [23](#)
- brglm2::brmultinom(), [9](#)
- cancer, [6](#)
- csregion, [7](#)
- csregion(), [10](#), [17](#)
- estimate_gps, [8](#)
- estimate_gps(), [2](#), [5](#), [14](#), [17](#), [18](#), [22](#), [23](#)
- get_select_params, [11](#)
- ggplot2::facet_wrap(), [19](#), [25](#)
- ggplot2::geom_jitter(), [26](#)
- ggplot2::geom_text(), [20](#)
- ggplot2::xlim(), [26](#)
- make_opt_args, [12](#)
- make_opt_args(), [22](#)
- MASS::polr(), [10](#)
- match_gps, [15](#)
- match_gps(), [2](#), [5](#), [10](#), [13](#), [14](#), [23](#)
- Matching::Matchby(), [16](#), [17](#)
- MatchIt::matchit(), [18](#)
- mclogit::mblogit(), [10](#)
- mosaic, [19](#)
- mosaic(), [27](#)
- nnet::multinom(), [9](#)
- optimize_gps, [21](#)
- optimize_gps(), [14](#)
- optmatch::fullmatch(), [16–18](#)
- optmatch::pairmatch(), [17](#), [18](#)
- raincloud, [24](#)
- rstatix::dunn_test(), [27](#)
- rstatix::games_howell_test(), [27](#)
- rstatix::pairwise_t_test(), [27](#)
- rstatix::pairwise_wilcox_test(), [27](#)
- rstatix::tukey_hsd(), [27](#)
- run_selected_matching, [28](#)
- select_opt, [29](#)
- select_opt(), [23](#)
- set.seed(), [14](#), [16](#)
- stats::formula(), [9](#), [22](#)
- stats::kmeans, [17](#)
- stats::kmeans(), [18](#)
- VGAM::vglm(), [9](#)