Package 'flipr'

October 13, 2022

Title Flexible Inference via Permutations in R

Version 0.3.2

Description A flexible permutation framework for making inference such as point estimation, confidence intervals or hypothesis testing, on any kind of data, be it univariate, multivariate, or more complex such as network-valued data, topological data, functional data or density-valued data.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.0

Imports purrr, rlang, magrittr, cli, ggplot2, tibble, withr, viridisLite, Rcpp, R6, pbapply, dials, usethis, optimParallel, rgenoud

Suggests rmarkdown, knitr, covr, tidyverse, plotly, htmlwidgets, htmltools, testthat (>= 3.0.0), interp

VignetteBuilder knitr

URL https://LMJL-Alea.github.io/flipr/, https://github.com/LMJL-Alea/flipr/

BugReports https://github.com/LMJL-Alea/flipr/issues/

Depends R (>= 2.10)

LinkingTo Rcpp

Config/testthat/edition 3

NeedsCompilation yes

Author Alessia Pini [aut], Aymeric Stamm [aut, cre] (<https://orcid.org/0000-0002-8725-3654>), Simone Vantini [aut], Juliette Chiapello [ctb]

Maintainer Aymeric Stamm <aymeric.stamm@math.cnrs.fr>

Repository CRAN

Date/Publication 2022-07-18 07:40:05 UTC

R topics documented:

`flipr` *flipr: Flexible inference via permutations in R*

Description

The flipr package provides a flexible permutation framework for making inference such as point estimation, confidence intervals or hypothesis testing, on any kind of data, be it univariate, multivariate, or more complex such as network-valued data, topological data, functional data or density-valued data.

`grid_biregular` *Create a biregular grid around a center point*

Description

Biregular grids can be created for any number of parameter objects.

Usage

```
grid_biregular(
  x,
  ...,
  center = NULL,
  levels = 3,
  original = TRUE,
  filter = NULL
)
```

Arguments

x	A param object, list, or parameters.
...	One or more param objects (such as mtry() or penalty()). None of the objects can have unknown() values in the parameter ranges or values.
center	A numeric vector specifying the point onto which the biregular grid should be centered. Defaults to NULL, in which case grid_regular is used instead.
levels	An integer for the number of values of each parameter to use to make the regular grid. levels can be a single integer or a vector of integers that is the same length as the number of parameters in .... levels can be a named integer vector, with names that match the id values of parameters.
original	A logical: should the parameters be in the original units or in the transformed space (if any)?
filter	A logical: should the parameters be filtered prior to generating the grid. Must be a single expression referencing parameter names that evaluates to a logical vector.

Details

Note that there may a difference in grids depending on how the function is called. If the call uses the parameter objects directly the possible ranges come from the objects in dials. For example:

```
mixture()
```

```
## Proportion of Lasso Penalty (quantitative)
## Range: [0, 1]
```

```
set.seed(283)
mix_grid_1 <- grid_random(mixture(), size = 1000)
range(mix_grid_1$mixture)
```

```
## [1] 0.001490161 0.999741096
```

However, in some cases, the parsnip and recipe packages overrides the default ranges for specific models and preprocessing steps. If the grid function uses a parameters object created from a model or recipe, the ranges may have different defaults (specific to those models). Using the example above, the mixture argument above is different for glmnet models:

```
library(parsnip)
library(tune)

# When used with glmnet, the range is [0.05, 1.00]
glmn_mod <-
  linear_reg(mixture = tune()) %>%
  set_engine("glmnet")

set.seed(283)
mix_grid_2 <- grid_random(extract_parameter_set_dials(glmn_mod), size = 1000)
range(mix_grid_2$mixture)
```

```
## [1] 0.05141565 0.99975404
```

Value

A tibble. There are columns for each parameter and a row for every parameter combination.

Examples

```
grid_biregular(dials::mixture(), center = 0.2)
```

`one-sample-stats` *Test Statistics for the One-Sample Problem*

Description

This is a collection of functions that provide test statistics to be used into the permutation scheme for performing one-sample testing.

Usage

```
stat_max(data, flips, ...)
```

Arguments

`data`	A list storing the sample from which the user wants to make inference.
`flips`	A numeric vectors of `-1`s and `1`s to be used to randomly flip some data points around the center of symmetric of the distribution of the sample.
`...`	Extra parameters specific to some statistics.

Value

A numeric value evaluating the desired test statistic.

Examples

```
n <- 10
x <- as.list(rnorm(n))
flips <- sample(c(-1, 1), n, replace = TRUE)
stat_max(x, flips)
```

`one_sample_test` *One-Sample Permutation Test*

Description

This function carries out an hypothesis test where the null hypothesis is that the sample is governed by a generative probability distribution which is centered and symmetric against the alternative hypothesis that they are governed by a probability distribution that is either not centered or not symmetric.

Usage

```
one_sample_test(
  x,
  stats = list(stat_max),
  B = 1000L,
  M = NULL,
  alternative = "two_tail",
  combine_with = "tippett",
  type = "exact",
  seed = NULL,
  ...
)
```

Arguments

`x`	A numeric vector or a numeric matrix or a list representing the sample from which the user wants to make inference.
`stats`	A list of functions produced by `as_function` specifying the chosen test statistic(s). A number of test statistic functions are implemented in the package and can be used as such. Alternatively, one can provide its own implementation of test statistics that (s)he deems relevant for the problem at hand. See the section *User-supplied statistic function* for more information on how these user-supplied functions should be structured for compatibility with the **flipr** framework. Default is `list(stat_t)`.
`B`	The number of sampled permutations. Default is `1000L`.
`M`	The total number of possible permutations. Defaults to `NULL`, which means that it is automatically computed from the given sample size(s).
`alternative`	A single string or a character vector specifying whether the p-value is right-tailed, left-tailed or two-tailed. Choices are `"right_tail"`, `"left_tail"` and `"two_tail"`. Default is `"two_tail"`. If a single string is provided, it is assumed that it should be applied to all test statistics provided by the user. Alternative, the length of `alternative` should match the length of the `stats` parameter and it is assumed that there is a one-to-one correspondence.

`combine_with`	A string specifying the combining function to be used to compute the single test statistic value from the set of p-value estimates obtained during the non-parametric combination testing procedure. For now, choices are either `"tippett"` or `"fisher"`. Default is `"tippett"`, which picks Tippett's function.
`type`	A string specifying which formula should be used to compute the p-value. Choices are `exact` (default), `upper_bound` and `estimate`. See Phipson & Smith (2010) for details.
`seed`	An integer specifying the seed of the random generator useful for result reproducibility or method comparisons. Default is `NULL`.
`...`	Extra parameters specific to some statistics.

Value

A `list` with three components: the value of the statistic for the original two samples, the p-value of the resulting permutation test and a numeric vector storing the values of the permuted statistics.

User-supplied statistic function

A user-specified function should have at least two arguments:

- the first argument is `data` which should be a list of the `n` observations from the sample;
- the second argument is `flips` which should be an integer vector giving the signs by which each observation in `data` should be multiplied.

It is possible to use the `use_stat` function with `nsamples = 1` to have **flipr** automatically generate a template file for writing down your own test statistics in a way that makes it compatible with the **flipr** framework.

See the `stat_max` function for an example.

Examples

```
n <- 10L
mu <- 3
sigma <- 1

# Sample under the null distribution
x1 <- rnorm(n = n, mean = 0, sd = sigma)
t1 <- one_sample_test(x1)
t1$pvalue

# Sample under some alternative distribution
x2 <- rnorm(n = n, mean = mu, sd = sigma)
t2 <- one_sample_test(x2)
t2$pvalue
```

PlausibilityFunction *R6 Class representing a plausibility function*

Description

A plausibility function is...

Public fields

`nparams` An integer specifying the number of parameters to be inferred. Default is `1L`.

`nperms` An integer specifying the number of permutations to be sampled. Default is `1000L`.

`nperms_max` An integer specifying the total number of distinct permutations that can be made given the sample sizes.

`alternative` A string specifying the type of alternative hypothesis. Choices are `"two_tail"`, `"left_tail"` and `"right_tail`. Defaults to `"two_tail"`.

`aggregator` A string specifying which function should be used to aggregate test statistic values when non-parametric combination is used (i.e. when multiple test statistics are used). Choices are `"tippett"` and `"fisher` for now. Defaults to `"tippett"`.

`pvalue_formula` A string specifying which formula to use for computing the permutation p-value. Choices are either `probability` (default) or `estimator`. The former provides p-values that lead to exact hypothesis tests while the latter provides an unbiased estimate of the traditional p-value.

`max_conf_level` A numeric value specifying the maximum confidence level that we aim to achieve for the confidence regions. This is used to compute bounds on each parameter of interest in order to fit a Kriging model that approximates the expensive plausibility function on a hypercube. Defaults to `0.99`.

`point_estimate` A numeric vector providing point estimates for the parameters of interest.

`parameters` A list of functions of class `param` produced via `new_quant_param` that stores the parameters to be inferred along with important properties such as their name, range, etc. Defaults to `NULL`.

`grid` A tibble storing evaluations of the plausibility function on a regular centered grid of the parameter space. Defaults to `NULL`.

Methods

Public methods:

- `PlausibilityFunction$new()`
- `PlausibilityFunction$set_nperms()`
- `PlausibilityFunction$set_nperms_max()`
- `PlausibilityFunction$set_alternative()`
- `PlausibilityFunction$set_aggregator()`
- `PlausibilityFunction$set_pvalue_formula()`
- `PlausibilityFunction$get_value()`

- PlausibilityFunction$set_max_conf_level()
- PlausibilityFunction$set_point_estimate()
- PlausibilityFunction$set_parameter_bounds()
- PlausibilityFunction$set_grid()
- PlausibilityFunction$evaluate_grid()
- PlausibilityFunction$clone()

Method `new()`**:** Create a new plausibility function object.

Usage:

```
PlausibilityFunction$new(
  null_spec,
  stat_functions,
  stat_assignments,
  ...,
  seed = NULL
)
```

Arguments:

`null_spec` A function or an R object coercible into a function (via `rlang::as_function()`). For one-sample problems, it should transform the `x` sample (provided as first argument) using the parameters (as second argument) to make its distribution centered symmetric. For two-sample problems, it should transform the `y` sample (provided as first argument) using the parameters (as second argument) to make it exchangeable with the `x` sample under a null hypothesis.

`stat_functions` A vector or list of functions (or R objects coercible into functions via `rlang::as_function()`) specifying the whole set of test statistics that should be used.

`stat_assignments` A named list of integer vectors specifying which test statistic should be associated with each parameter. The length of this list should match the number of parameters under investigation and is thus used to set it. Each element of the list should be named after the parameter it identifies.

`...` Vectors, matrices or lists providing the observed samples.

`seed` A numeric value specifying the seed to be used. Defaults to `NULL` in which case `seed = 1234` is used and the user is informed of this setting.

Returns: A new `PlausibilityFunction` object.

Method `set_nperms()`**:** Change the value of the `nperms` field.

Usage:

```
PlausibilityFunction$set_nperms(val)
```

Arguments:

`val` New value for the number of permutations to be sampled.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
```

```
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$nperms
pf$set_nperms(10000)
pf$nperms
```

Method `set_nperms_max()`**:** Change the value of the `nperms_max` field.

Usage:

```
PlausibilityFunction$set_nperms_max(val)
```

Arguments:

`val` New value for the total number of of possible distinct permutations.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$nperms_max
pf$set_nperms_max(10000)
pf$nperms_max
```

Method `set_alternative()`**:** Change the value of the `alternative` field.

Usage:

```
PlausibilityFunction$set_alternative(val)
```

Arguments:

`val` New value for the type of alternative hypothesis.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
```

```
  stat_assignments = stat_assignments,
  x, y
)
pf$alternative
pf$set_alternative("right_tail")
pf$alternative
```

Method `set_aggregator()`**:** Change the value of the `aggregator` field.

Usage:

```
PlausibilityFunction$set_aggregator(val)
```

Arguments:

`val` New value for the string specifying which function should be used to aggregate test statistic values when non-parametric combination is used (i.e. when multiple test statistics are used).

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$aggregator
pf$set_aggregator("fisher")
pf$aggregator
```

Method `set_pvalue_formula()`**:** Change the value of the `pvalue_formula` field.

Usage:

```
PlausibilityFunction$set_pvalue_formula(val)
```

Arguments:

`val` New value for the string specifying which formula should be used to compute the permutation p-value.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
```

```
  stat_assignments = stat_assignments,
  x, y
)
pf$pvalue_formula
pf$set_pvalue_formula("estimate")
pf$pvalue_formula
```

Method `get_value()`**:** Computes an indicator of the plausibility of specific values for the parameters of interest in the form of a p-value of an hypothesis test against these values.

Usage:

```
PlausibilityFunction$get_value(
  parameters,
  keep_null_distribution = FALSE,
  keep_permutations = FALSE,
  ...
)
```

Arguments:

`parameters` A vector whose length should match the `nparams` field providing specific values of the parameters of interest for assessment of their plausibility in the form of a p-value of the corresponding hypothesis test.

`keep_null_distribution` A boolean specifying whether the empirical permutation null distribution should be returned as well. Defaults to `FALSE`.

`keep_permutations` A boolean specifying whether the list of sampled permutations used to compute the empirical permutation null distribution should be returned as well. Defaults to `FALSE`.

`...` Extra parameters specific to some statistics.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$set_nperms(50)
pf$get_value(2)
```

Method `set_max_conf_level()`**:** Change the value of the `max_conf_level` field.

Usage:

```
PlausibilityFunction$set_max_conf_level(val)
```

Arguments:

`val` New value for the maximum confidence level that we aim to achieve for the confidence regions.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$max_conf_level
pf$set_max_conf_level(0.999)
pf$max_conf_level
```

Method `set_point_estimate()`**:** Change the value of the `point_estimate` field.

Usage:

```
PlausibilityFunction$set_point_estimate(
  point_estimate = NULL,
  lower_bound = -10,
  upper_bound = 10,
  ncores = 1L,
  estimate = FALSE,
  overwrite = FALSE
)
```

Arguments:

`point_estimate` A numeric vector providing rough point estimates for the parameters under investigation.

`lower_bound` A scalar or numeric vector specifying the lower bounds for each parameter under investigation. If it is a scalar, the value is used as lower bound for all parameters. Defaults to `-10`.

`upper_bound` A scalar or numeric vector specifying the lower bounds for each parameter under investigation. If it is a scalar, the value is used as lower bound for all parameters. Defaults to `10`.

`ncores` An integer specifying the number of cores to use for maximizing the plausibility function to get a point estimate of the parameters. Defaults to `1L`.

`estimate` A boolean specifying whether the rough point estimate provided by `val` should serve as initial point for maximizing the plausibility function (`estimate = TRUE`) or as final point estimate for the parameters (`estimate = FALSE`). Defaults to `FALSE`.

`overwrite` A boolean specifying whether to force the computation if it has already been set. Defaults to `FALSE`.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$point_estimate
pf$set_point_estimate(mean(y) - mean(x))
pf$point_estimate
```

Method `set_parameter_bounds()`**:** Change the value of the `parameters` field.

Updates the range of the parameters under investigation.

Usage:

```
PlausibilityFunction$set_parameter_bounds(point_estimate, conf_level)
```

Arguments:

`point_estimate` A numeric vector providing a point estimate for each parameter under investigation. If no estimator is known by the user, (s)he can resort to the `$set_point_estimate()` method to get a point estimate by maximizing the plausibility function.

`conf_level` A numeric value specifying the confidence level to be used for setting parameter bounds. It should be in (0,1).

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$set_nperms(50)
pf$set_point_estimate(point_estimate = mean(y) - mean(x))
pf$parameters
pf$set_parameter_bounds(
  point_estimate = pf$point_estimate,
```

```
  conf_level = 0.8
)
pf$parameters
```

Method `set_grid()`**:** Computes a tibble storing a regular centered grid of the parameter space.

Usage:

```
PlausibilityFunction$set_grid(parameters, npoints = 20L)
```

Arguments:

`parameters` A list of `new_quant_param` objects containing information about the parameters under investigation. It should contain the fields `point_estimate` and `range`.

`npoints` An integer specifying the number of points to discretize each dimension. Defaults to `20L`.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$set_nperms(50)
pf$set_point_estimate(mean(y) - mean(x))
pf$set_parameter_bounds(
  point_estimate = pf$point_estimate,
  conf_level = 0.8
)
pf$set_grid(
  parameters = pf$parameters,
  npoints = 2L
)
```

Method `evaluate_grid()`**:** Updates the `grid` field with a `pvalue` column storing evaluations of the plausibility function on the regular centered grid of the parameter space.

Usage:

```
PlausibilityFunction$evaluate_grid(grid, ncores = 1L)
```

Arguments:

`grid` A `tibble` storing a grid that spans the space of parameters under investigation.

`ncores` An integer specifying the number of cores to run evaluations in parallel. Defaults to `1L`.

Examples:

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$set_nperms(50)
pf$set_point_estimate(mean(y) - mean(x))
pf$set_parameter_bounds(
  point_estimate = pf$point_estimate,
  conf_level = 0.8
)
pf$set_grid(
  parameters = pf$parameters,
  npoints = 2L
)
pf$evaluate_grid(grid = pf$grid)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PlausibilityFunction$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PlausibilityFunction$set_nperms`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
```

```
pf$nperms
pf$set_nperms(10000)
pf$nperms

## ------------------------------------------------
## Method `PlausibilityFunction$set_nperms_max`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$nperms_max
pf$set_nperms_max(10000)
pf$nperms_max

## ------------------------------------------------
## Method `PlausibilityFunction$set_alternative`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$alternative
pf$set_alternative("right_tail")
pf$alternative

## ------------------------------------------------
## Method `PlausibilityFunction$set_aggregator`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
```

```
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$aggregator
pf$set_aggregator("fisher")
pf$aggregator

## ------------------------------------------------
## Method `PlausibilityFunction$set_pvalue_formula`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$pvalue_formula
pf$set_pvalue_formula("estimate")
pf$pvalue_formula

## ------------------------------------------------
## Method `PlausibilityFunction$get_value`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$set_nperms(50)
pf$get_value(2)

## ------------------------------------------------
## Method `PlausibilityFunction$set_max_conf_level`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
```

```
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$max_conf_level
pf$set_max_conf_level(0.999)
pf$max_conf_level

## ------------------------------------------------
## Method `PlausibilityFunction$set_point_estimate`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$point_estimate
pf$set_point_estimate(mean(y) - mean(x))
pf$point_estimate

## ------------------------------------------------
## Method `PlausibilityFunction$set_parameter_bounds`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
```

```
pf$set_nperms(50)
pf$set_point_estimate(point_estimate = mean(y) - mean(x))
pf$parameters
pf$set_parameter_bounds(
  point_estimate = pf$point_estimate,
  conf_level = 0.8
)
pf$parameters

## ------------------------------------------------
## Method `PlausibilityFunction$set_grid`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$set_nperms(50)
pf$set_point_estimate(mean(y) - mean(x))
pf$set_parameter_bounds(
  point_estimate = pf$point_estimate,
  conf_level = 0.8
)
pf$set_grid(
  parameters = pf$parameters,
  npoints = 2L
)

## ------------------------------------------------
## Method `PlausibilityFunction$evaluate_grid`
## ------------------------------------------------

x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {
  purrr::map(y, ~ .x - parameters[1])
}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
```

```
)
pf$set_nperms(50)
pf$set_point_estimate(mean(y) - mean(x))
pf$set_parameter_bounds(
  point_estimate = pf$point_estimate,
  conf_level = 0.8
)
pf$set_grid(
  parameters = pf$parameters,
  npoints = 2L
)
pf$evaluate_grid(grid = pf$grid)
```

`plot_pf` *Visualization of Plausibility Functions*

Description

This function plots the plausibility function for up to two parameters of interest.

Usage

```
plot_pf(pf, alpha = 0.05, ngrid = 10, ncores = 1, subtitle = "")
```

Arguments

`pf`	A `PlausibilityFunction` object.
`alpha`	A numeric value specifying a significance level to contrast the plausibility function against. Defaults to `0.05`.
`ngrid`	An integer specifying the grid size on which the plausibility function will be evaluated. Specifically if `K` is the number of parameters under investigation, the grid will be of size `(ngrid + 1)^K`. Defaults to `10L`.
`ncores`	An integer specifying the number of cores to use for parallelized computations. Defaults to `1L`.
`subtitle`	A string for specifying a subtitle to the plot. Defaults to `""` leading to no subtitle.

Value

A `ggplot` object.

Examples

```
x <- rnorm(10)
y <- rnorm(10, mean = 2)
null_spec <- function(y, parameters) {purrr::map(y, ~ .x - parameters[1])}
stat_functions <- list(stat_t)
stat_assignments <- list(mean = 1)
pf <- PlausibilityFunction$new(
```

```
  null_spec = null_spec,
  stat_functions = stat_functions,
  stat_assignments = stat_assignments,
  x, y
)
pf$set_nperms(50)
pf$set_point_estimate(mean(y) - mean(x))
pf$set_parameter_bounds(
  point_estimate = pf$point_estimate,
  conf_level = 0.8
)
pf$set_grid(
  parameters = pf$parameters,
  npoints = 2L
)
pf$evaluate_grid(grid = pf$grid)
plot_pf(pf)
```

---

two-sample-stats *Test Statistics for the Two-Sample Problem*

Description

This is a collection of functions that provide test statistics to be used into the permutation scheme for performing two-sample testing. These test statistics can be divided into two categories: traditional statistics that use empirical moments and inter-point statistics that only rely on pairwise dissimilarities between data points.

Usage

```
stat_welch(data, indices1, ...)

stat_student(data, indices1, ...)

stat_t(data, indices1, ...)

stat_fisher(data, indices1, ...)

stat_f(data, indices1, ...)

stat_mean(data, indices1, ...)

stat_hotelling(data, indices1, ...)

stat_bs(data, indices1, ...)

stat_student_ip(data, indices1, ...)
```

```
stat_t_ip(data, indices1, ...)

stat_fisher_ip(data, indices1, ...)

stat_f_ip(data, indices1, ...)

stat_bg_ip(data, indices1, ...)

stat_energy_ip(data, indices1, alpha = 1L, ...)

stat_cq_ip(data, indices1, ...)

stat_mod_ip(data, indices1, ...)

stat_dom_ip(data, indices1, standardize = TRUE, ...)
```

Arguments

data	Either a list of the n1 + n2 concatenated observations with the original n1 observations from the first sample on top and the original n2 observations from the second sample below. Or a dissimilarity matrix stored as a dist object for all inter-point statistics whose function name should end with _ip().
indices1	An integer vector specifying the indices in data that are considered to belong to the first sample.
...	Extra parameters specific to some statistics.
alpha	A scalar value specifying the power to which the dissimilarities should be elevated in the computation of the inter-point energy statistic. Default is 1L.
standardize	A boolean specifying whether the distance between medoids in the stat_dom_ip function should be normalized by the pooled corresponding variances. Default is TRUE.

Value

A real scalar giving the value of test statistic for the permutation specified by the integer vector indices.

Traditional Test Statistics

- stat_hotelling implements Hotelling's T^2 statistic for multivariate data with $p < n$.
- stat_student or stat_t implements Student's statistic (originally assuming equal variances and thus using the pooled empirical variance estimator). See t.test for details.
- stat_welch implements Student-Welch statistic which is essentially a modification of Student's statistic accounting for unequal variances. See t.test for details.
- stat_fisher or stat_f implements Fisher's variance ratio statistic. See var.test for details.
- stat_mean implements a statistic that computes the difference between the means.
- stat_bs implements the statistic proposed by Bai & Saranadasa (1996) for high-dimensional multivariate data.

Inter-Point Test Statistics

- stat_student_ip or stat_t_ip implements a Student-like test statistic based on inter-point distances only as described in Lovato et al. (2020).
- stat_fisher_ip or stat_f_ip implements a Fisher-like test statistic based on inter-point distances only as described in Lovato et al. (2020).
- stat_bg_ip implements the statistic proposed by Biswas & Ghosh (2014).
- stat_energy_ip implements the class of energy-based statistics as described in Székely & Rizzo (2013);
- stat_cq_ip implements the statistic proposed by Chen & Qin (2010).
- stat_mod_ip implements a statistic that computes the mean of inter-point distances.
- stat_dom_ip implements a statistic that computes the distance between the medoids of the two samples, possibly standardized by the pooled corresponding variances.

References

Bai, Z., & Saranadasa, H. (1996). Effect of high dimension: by an example of a two sample problem. Statistica Sinica, 311-329.

Lovato, I., Pini, A., Stamm, A., & Vantini, S. (2020). Model-free two-sample test for network-valued data. Computational Statistics & Data Analysis, 144, 106896.

Biswas, M., & Ghosh, A. K. (2014). A nonparametric two-sample test applicable to high dimensional data. Journal of Multivariate Analysis, 123, 160-171.

Székely, G. J., & Rizzo, M. L. (2013). Energy statistics: A class of statistics based on distances. Journal of statistical planning and inference, 143(8), 1249-1272.

Chen, S. X., & Qin, Y. L. (2010). A two-sample test for high-dimensional data with applications to gene-set testing. The Annals of Statistics, 38(2), 808-835.

Examples

```
n <- 10L
mx <- 0
sigma <- 1
delta <- 10
my <- mx + delta
x <- rnorm(n = n, mean = mx, sd = sigma)
y <- rnorm(n = n, mean = my, sd = sigma)
D <- dist(c(x, y))

x <- as.list(x)
y <- as.list(y)

stat_welch(c(x, y), 1:n)
stat_t(c(x, y), 1:n)
stat_f(c(x, y), 1:n)
stat_mean(c(x, y), 1:n)
stat_hotelling(c(x, y), 1:n)
stat_bs(c(x, y), 1:n)
```

```
stat_t_ip(D, 1:n)
stat_f_ip(D, 1:n)
stat_bg_ip(D, 1:n)
stat_energy_ip(D, 1:n)
stat_cq_ip(D, 1:n)
stat_mod_ip(D, 1:n)
stat_dom_ip(D, 1:n)
```

`two_sample_test` *Two-Sample Permutation Test*

Description

This function carries out an hypothesis test in which the null hypothesis is that the two samples are governed by the same underlying generative probability distribution against the alternative hypothesis that they are governed by two different generative probability distributions.

Usage

```
two_sample_test(
  x,
  y,
  stats = list(stat_t),
  B = 1000L,
  M = NULL,
  alternative = "two_tail",
  combine_with = "tippett",
  type = "exact",
  seed = NULL,
  ...
)
```

Arguments

`x`	A numeric vector or a numeric matrix or a list representing the 1st sample. Alternatively, it can be a distance matrix stored as an object of class `dist`, in which case test statistics based on inter-point distances (marked with the `_ip` suffix) should be used.
`y`	A numeric vector if `x` is a numeric vector, or a numeric matrix if `x` is a numeric matrix, or a list if `x` is a list, representing the second sample. Alternatively, if `x` is an object of class `dist`, it should be a numeric scalar specifying the size of the first sample.
`stats`	A list of functions produced by `as_function` specifying the chosen test statistic(s). A number of test statistic functions are implemented in the package and can be used as such. Alternatively, one can provide its own implementation of

test statistics that (s)he deems relevant for the problem at hand. See the section *User-supplied statistic function* for more information on how these user-supplied functions should be structured for compatibility with the **flipr** framework. Default is `list(stat_t)`.

`B` The number of sampled permutations. Default is `1000L`.

`M` The total number of possible permutations. Defaults to `NULL`, which means that it is automatically computed from the given sample size(s).

`alternative` A single string or a character vector specifying whether the p-value is right-tailed, left-tailed or two-tailed. Choices are `"right_tail"`, `"left_tail"` and `"two_tail"`. Default is `"two_tail"`. If a single string is provided, it is assumed that it should be applied to all test statistics provided by the user. Alternative, the length of `alternative` should match the length of the `stats` parameter and it is assumed that there is a one-to-one correspondence.

`combine_with` A string specifying the combining function to be used to compute the single test statistic value from the set of p-value estimates obtained during the non-parametric combination testing procedure. For now, choices are either `"tippett"` or `"fisher"`. Default is `"tippett"`, which picks Tippett's function.

`type` A string specifying which formula should be used to compute the p-value. Choices are `exact` (default), `upper_bound` and `estimate`. See Phipson & Smith (2010) for details.

`seed` An integer specifying the seed of the random generator useful for result reproducibility or method comparisons. Default is `NULL`.

`...` Extra parameters specific to some statistics.

Value

A `list` with three components: the value of the statistic for the original two samples, the p-value of the resulting permutation test and a numeric vector storing the values of the permuted statistics.

User-supplied statistic function

A user-specified function should have at least two arguments:

- the first argument is `data` which should be a list of the `n1 + n2` concatenated observations with the original `n1` observations from the first sample on top and the original `n2` observations from the second sample below;
- the second argument is `perm_data` which should be an integer vector giving the indices in `data` that are considered to belong to the first sample.

It is possible to use the `use_stat` function with `nsamples = 2` to have **flipr** automatically generate a template file for writing down your own test statistics in a way that makes it compatible with the **flipr** framework.

See the `stat_t` function for an example.

Examples

```
n <- 10L
mx <- 0
sigma <- 1

# Two different models for the two populations
x <- rnorm(n = n, mean = mx, sd = sigma)
delta <- 10
my <- mx + delta
y <- rnorm(n = n, mean = my, sd = sigma)
t1 <- two_sample_test(x, y)
t1$pvalue

# Same model for the two populations
x <- rnorm(n = n, mean = mx, sd = sigma)
delta <- 0
my <- mx + delta
y <- rnorm(n = n, mean = my, sd = sigma)
t2 <- two_sample_test(x, y)
t2$pvalue
```

---

use_stat *Test Statistic Template*

Description

This function is a helper to automatically generate an `.R` file populated with a skeleton of a typical test function compatible with `flipr`.

Usage

```
use_stat(nsamples = 1, stat_name = "mystat")
```

Arguments

`nsamples`	An integer specifying the number of samples to be used. Defaults to `1L`. Currently only works for one- or two-sample problems.
`stat_name`	A string specifying the name of the test statistic that is being implemented. Defaults to `mystat`.

Value

Creates a dedicated `.R` file with a template of code for the function that implements the test statistic and saves it to the `R/` folder of your package.

Examples

```
## Not run:
use_stat()

## End(Not run)
```

Index