

Package ‘LearnNonparam’

April 29, 2025

Title 'R6'-Based Flexible Framework for Permutation Tests

Version 1.2.8

Description Implements non-parametric tests from Higgins (2004, ISBN:0534387756), including tests for one sample, two samples, k samples, paired comparisons, blocked designs, trends and association. Built with 'Rcpp' for efficiency and 'R6' for flexible, object-oriented design, the package provides a unified framework for performing or creating custom permutation tests.

BugReports <https://github.com/qddy/LearnNonparam/issues>

URL <https://github.com/qddy/LearnNonparam>,
<https://qddy.github.io/LearnNonparam/>

License GPL (>= 2)

Depends R (>= 3.5.0)

LinkingTo Rcpp (>= 1.0.10)

Imports R6 (>= 2.5.0), Rcpp (>= 1.0.10), stats, graphics, compiler

Suggests ggplot2

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

NeedsCompilation yes

Author Yan Du [aut, cre] (<<https://orcid.org/0009-0009-1169-921X>>)

Maintainer Yan Du <isduyan@outlook.com>

Repository CRAN

Date/Publication 2025-04-29 17:00:02 UTC

Contents

AnsariBradley	3
CDF	4
ChiSquare	5

ContingencyTableTest	5
Correlation	6
Difference	7
Friedman	8
JonckheereTerpstra	9
KolmogorovSmirnov	10
KruskalWallis	10
KSampleTest	11
MultipleComparison	12
OneSampleTest	12
OneWay	13
Page	14
PairedDifference	15
PermuTest	16
pmt	17
Quantile	20
RatioMeanDeviance	22
RCBDOneWay	23
RCBDTest	23
ScoreSum	24
SiegelTukey	25
Sign	26
Studentized	27
Table1.1.1	28
Table1.2.1	29
Table2.1.1	29
Table2.3.1	30
Table2.6.1	30
Table2.6.2	31
Table2.8.1	31
Table3.1.2	32
Table3.2.2	32
Table3.2.3	33
Table3.3.1	33
Table3.4.1	34
Table4.1.1	34
Table4.1.3	35
Table4.4.3	35
Table4.5.3	36
Table5.1.2	36
Table5.2.2	37
Table5.4.2	37
TwoSampleAssociationTest	38
TwoSampleLocationTest	38
TwoSamplePairedTest	38
TwoSampleTest	38
Wilcoxon	39

Description

Performs Ansari-Bradley test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> AnsariBradley

Methods

Public methods:

- [AnsariBradley\\$new\(\)](#)

Method `new()`: Create a new AnsariBradley object.

Usage:

```
AnsariBradley$new(  
  type = c("permu", "asyp"),  
  alternative = c("two_sided", "less", "greater"),  
  n_permu = 10000  
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`alternative` a character string specifying the alternative hypothesis.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A AnsariBradley object.

Examples

```
pmt(  
  "twosample.ansari",  
  alternative = "greater", n_permu = 0  
)$test(Table2.8.1)$print()
```

Description

Performs statistical inference on population cumulative distribution function.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::OneSampleTest](#) -> CDF

Methods**Public methods:**

- [CDF\\$new\(\)](#)
- [CDF\\$plot\(\)](#)

Method `new()`: Create a new CDF object.

Usage:

```
CDF$new(method = c("binomial", "dkw"), conf_level = 0.95)
```

Arguments:

`method` a character string specifying whether to use a confidence band based on the binomial distribution or the Dvoretzky–Kiefer–Wolfowitz inequality.

`conf_level` a number specifying confidence level of the confidence bounds.

Returns: A CDF object.

Method `plot()`: Plot the estimate and confidence bounds for population cumulative distribution function.

Usage:

```
CDF$plot(style = c("graphics", "ggplot2"))
```

Arguments:

`style` a character string specifying which package to use.

Returns: The object itself (invisibly).

Examples

```
pmt("onesample.cdf")$test(Table1.2.1)$plot(style = "graphic")
```

ChiSquare	<i>Chi-Square Test on Contingency Table</i>
-----------	---

Description

Performs chi-square test on contingency tables.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::ContingencyTableTest](#) -> ChiSquare

Methods**Public methods:**

- [ChiSquare\\$new\(\)](#)

Method `new()`: Create a new ChiSquare object.

Usage:

```
ChiSquare$new(type = c("permu", "asyp"), n_permu = 10000)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A ChiSquare object.

Examples

```
t <- pmt(
  "table.chisq", n_permu = 0
)$test(Table5.4.2)$print()

t$type <- "asyp"
t
```

ContingencyTableTest	<i>ContingencyTableTest Class</i>
----------------------	-----------------------------------

Description

Abstract class for tests on contingency tables.

Super class

[LearnNonparam::PermuTest](#) -> ContingencyTableTest

Correlation

*Test for Association Between Paired Samples***Description**

Performs correlation coefficient based two-sample association test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> [LearnNonparam::TwoSamplePairedTest](#)
-> [LearnNonparam::TwoSampleAssociationTest](#) -> [Correlation](#)

Methods**Public methods:**

- [Correlation\\$new\(\)](#)

Method `new()`: Create a new `Correlation` object.

Usage:

```
Correlation$new(
  type = c("permu", "asyp"),
  method = c("pearson", "kendall", "spearman"),
  alternative = c("two_sided", "less", "greater"),
  n_permu = 10000
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`method` a character string specifying the correlation coefficient to be used.

`alternative` a character string specifying the alternative hypothesis.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A `Correlation` object.

Examples

```
pmt(
  "association.corr", method = "pearson",
  alternative = "greater", n_permu = 10000
)$test(Table5.1.2)$print()

t <- pmt(
  "association.corr", method = "spearman",
  alternative = "two_sided", n_permu = 10000
)$test(Table5.1.2)$print()

t$type <- "asyp"
```

```
t
t <- pmt(
  "association.corr", method = "kendall",
  alternative = "greater", n_permu = 0
)$test(Table5.2.2)$print()

t$type <- "asypm"
t
```

Difference

Two-Sample Test Based on Mean or Median

Description

Performs mean/median based two-sample test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> [LearnNonparam::TwoSampleLocationTest](#)
-> Difference

Methods

Public methods:

- [Difference\\$new\(\)](#)

Method `new()`: Create a new Difference object.

Usage:

```
Difference$new(
  method = c("mean", "median"),
  alternative = c("two_sided", "less", "greater"),
  null_value = 0,
  n_permu = 10000
)
```

Arguments:

`method` a character string specifying whether to use the mean or the median.

`alternative` a character string specifying the alternative hypothesis.

`null_value` a number indicating the true value of the location shift.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A Difference object.

Examples

```
pmt(
  "twosample.difference", method = "mean",
  alternative = "greater", n_permu = 0
)$test(Table2.1.1)$print()$plot(
  style = "graphic", breaks = seq(-20, 25, length.out = 9)
)
```

```
pmt(
  "twosample.difference", method = "mean",
  alternative = "greater", n_permu = 1000
)$test(Table2.3.1)$print()
```

 Friedman

Friedman Test

Description

Performs Friedman test on samples collected in a randomized complete block design.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::RCBDTest](#) -> Friedman

Methods**Public methods:**

- [Friedman\\$new\(\)](#)

Method `new()`: Create a new Friedman object.

Usage:

```
Friedman$new(type = c("permu", "asyp"), n_permu = 10000)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A Friedman object.

Examples

```
t <- pmt(
  "rcbd.friedman", n_permu = 0
)$test(Table4.5.3)$print()
```

```
t$type <- "asyp"
t
```


Description

Performs Jonckheere-Terpstra test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::KSampleTest](#) -> JonckheereTerpstra

Methods

Public methods:

- [JonckheereTerpstra\\$new\(\)](#)

Method `new()`: Create a new JonckheereTerpstra object.

Usage:

```
JonckheereTerpstra$new(  
  type = c("permu", "asyp"),  
  alternative = c("two_sided", "less", "greater"),  
  n_permu = 10000  
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`alternative` a character string specifying the alternative hypothesis.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A JonckheereTerpstra object.

Examples

```
t <- pmt(  
  "ksample.jt", alternative = "greater"  
)$test(Table3.4.1)$print()
```

```
t$type <- "asyp"  
t
```

KolmogorovSmirnov *Two-Sample Kolmogorov-Smirnov Test*

Description

Performs two-sample Kolmogorov-Smirnov test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> KolmogorovSmirnov

Methods

Public methods:

- [KolmogorovSmirnov\\$new\(\)](#)

Method `new()`: Create a new KolmogorovSmirnov object.

Usage:

```
KolmogorovSmirnov$new(n_permu = 10000)
```

Arguments:

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A KolmogorovSmirnov object.

Examples

```
pmt(
  "twosample.ks", n_permu = 0
)$test(Table2.8.1)$print()
```

KruskalWallis *Kruskal-Wallis Test*

Description

Performs Kruskal-Wallis test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::KSampleTest](#) -> KruskalWallis

Methods**Public methods:**

- [KruskalWallis\\$new\(\)](#)

Method `new()`: Create a new `KruskalWallis` object.

Usage:

```
KruskalWallis$new(
  type = c("permu", "asyp"),
  scoring = c("rank", "vw", "expon"),
  n_permu = 10000
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`scoring` a character string specifying the scoring system.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A `KruskalWallis` object.

Examples

```
pmt(
  "ksample.kw", type = "asyp"
)$test(Table3.2.2)$print()

t <- pmt(
  "ksample.kw", type = "permu"
)$test(Table3.2.3)$print()

t$type <- "asyp"
t
```

KSampleTest

KSampleTest Class

Description

Abstract class for k-sample tests.

Super class

[LearnNonparam::PermuTest](#) -> KSampleTest

MultipleComparison *MultipleComparison Class*

Description

Abstract class for multiple comparisons.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::KSampleTest](#) -> MultipleComparison

OneSampleTest *OneSampleTest Class*

Description

Abstract class for one-sample tests.

Super class

[LearnNonparam::PermuTest](#) -> OneSampleTest

Methods**Public methods:**

- [OneSampleTest\\$plot\(\)](#)

Method plot():

Usage:

OneSampleTest\$plot(...)

Arguments:

... ignored.

OneWay	<i>One-Way Test for Equal Means</i>
--------	-------------------------------------

Description

Performs F statistic based one-way test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::KSampleTest](#) -> OneWay

Methods

Public methods:

- [OneWay\\$new\(\)](#)

Method `new()`: Create a new OneWay object.

Usage:

```
OneWay$new(type = c("permu", "asyp"), n_permu = 10000)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A OneWay object.

Examples

```
t <- pmt(
  "ksample.oneway", n_permu = 0
)$test(Table3.1.2)$print()
```

```
t$type <- "asyp"
t
```

Description

Performs Page test on samples collected in a randomized complete block design.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::RCBDTest](#) -> Page

Methods

Public methods:

- [Page\\$new\(\)](#)

Method `new()`: Create a new Page object.

Usage:

```
Page$new(  
  type = c("permu", "asyp"),  
  alternative = c("two_sided", "less", "greater"),  
  n_permu = 10000  
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`alternative` a character string specifying the alternative hypothesis.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A Page object.

Examples

```
t <- pmt(  
  "rcbd.page", alternative = "less"  
)$test(Table4.4.3)$print()
```

```
t$type <- "asyp"  
t
```

PairedDifference	<i>Paired Comparison Based on Differences</i>
------------------	---

Description

Performs differences based paired comparison on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> [LearnNonparam::TwoSamplePairedTest](#)
-> PairedDifference

Active bindings

correct Whether to apply continuity correction when scoring is set to "rank".

Methods**Public methods:**

- [PairedDifference\\$new\(\)](#)

Method new(): Create a new PairedDifference object.

Usage:

```
PairedDifference$new(
  type = c("permu", "asyp"),
  method = c("with_zeros", "without_zeros"),
  scoring = c("none", "rank", "vw", "expon"),
  alternative = c("two_sided", "less", "greater"),
  null_value = 0,
  n_permu = 10000,
  correct = TRUE
)
```

Arguments:

type a character string specifying the way to calculate the p-value.

method a character string specifying the method of ranking data in computing adjusted signed scores for tied data, must be one of "with_zeros" (default) or "without_zeros".

scoring a character string specifying the scoring system.

alternative a character string specifying the alternative hypothesis.

null_value a number indicating the true value of the location shift.

n_permu an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

correct a logical indicating whether to apply continuity correction in the normal approximation for the p-value when scoring is set to "rank".

Returns: A PairedDifference object.

Examples

```

pmt(
  "paired.difference",
  alternative = "greater", scoring = "none", n_permu = 0
)$test(Table4.1.1)$print()

pmt(
  "paired.difference", n_permu = 0
)$test(Table4.1.3)$print()

t <- pmt(
  "paired.difference", scoring = "rank",
  alternative = "greater", n_permu = 0
)$test(Table4.1.1)$print()

t$type <- "asypm"
t

```

PermuTest

PermuTest Class

Description

Abstract class for permutation tests.

Active bindings

`type` The way to calculate the p-value.

`method` The method used.

`scoring` The scoring system used.

`alternative` The alternative hypothesis.

`null_value` The hypothesized value of the parameter in the null hypothesis.

`conf_level` The confidence level of the interval.

`n_permu` The number of permutations used.

`data` The data.

`statistic` The test statistic.

`p_value` The p-value.

`estimate` The estimated value of the parameter.

`conf_int` The confidence interval of the parameter.

Methods

Public methods:

- [PermuTest\\$test\(\)](#)
- [PermuTest\\$print\(\)](#)
- [PermuTest\\$plot\(\)](#)

Method `test()`: Perform test on sample(s).

Usage:

```
PermuTest$test(...)
```

Arguments:

... sample(s). Can be numeric vector(s) or a data.frame or list containing them.

Details: A progress bar is shown by default. Use `options(LearnNonparam.pmt_progress = FALSE)` to disable it.

Returns: The object itself (invisibly).

Method `print()`: Print the results of the test.

Usage:

```
PermuTest$print()
```

Returns: The object itself (invisibly).

Method `plot()`: Plot histogram(s) of the permutation distribution. Note that this method only works if `type` is set to "permu".

Usage:

```
PermuTest$plot(style = c("graphics", "ggplot2"), ...)
```

Arguments:

`style` a character string specifying which package to use.

... passed to `graphics::hist.default()` or `ggplot2::stat_bin()`.

Returns: The object itself (invisibly).

Description

Construct test objects in a unified way.

Usage

```

pmt(key, ...)

pmts(
  which = c("all", "onesample", "twosample", "ksample", "multcomp", "paired", "rcbd",
            "association", "table")
)

define_pmt(
  inherit = c("twosample", "ksample", "paired", "rcbd", "association", "table"),
  statistic,
  rejection = c("lr", "l", "r"),
  scoring = "none",
  n_permu = 10000,
  name = "User-Defined Permutation Test",
  alternative = NULL,
  depends = character(),
  plugins = character(),
  includes = character()
)

```

Arguments

key	a character string specifying the test. Check <code>pmts()</code> for valid keys.
...	extra parameters passed to the constructor.
which	a character string specifying the desired tests.
inherit	a character string specifying the type of permutation test.
statistic	definition of the test statistic. See details.
rejection	a character string specifying where the rejection region is.
scoring	one of: - a character string in <code>c("none", "rank", "vw", "expon")</code> specifying the scoring system - a function that takes a numeric vector and returns an equal-length score vector
n_permu	an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.
name	a character string specifying the name of the test.
alternative	a character string describing the alternative hypothesis.
depends, plugins, includes	passed to <code>Rcpp::cppFunction()</code> .

Details

The test statistic can be defined using either R or Rcpp, with the `statistic` parameter specified as:

- R: a function returning a closure that returns a double.
- Rcpp: a character string defining a captureless lambda (since C++11) returning another lambda that captures by value, accepts parameters of the same type, and returns a double.

The purpose of this design is to pre-calculate certain constants that remain invariant during permutation.

When using Rcpp, the parameters for different `inherit` are listed as follows. Note that the names can be customized, and the types can be replaced with `auto` (thanks to the support for generic lambdas in C++14). See examples.

<code>inherit</code>	Parameter 1	Parameter 2
<code>"twosample"</code>	<code>const NumericVector& sample_1</code>	<code>const NumericVector& sample_2</code>
<code>"ksample"</code>	<code>const NumericVector& combined_sample</code>	<code>const IntegerVector& one_based_group_index</code>
<code>"paired"</code>	<code>const NumericVector& sample_1</code>	<code>const NumericVector& sample_2</code>
<code>"rcbd"</code>	<code>const NumericMatrix& block_as_column_data</code>	
<code>"association"</code>	<code>const NumericVector& sample_1</code>	<code>const NumericVector& sample_2</code>
<code>"table"</code>	<code>const IntegerMatrix& contingency_table</code>	

When using R, the parameters should be the R equivalents of these.

Value

a test object corresponding to the specified key.

a data frame containing keys and corresponding tests implemented in this package.

a test object based on the specified statistic.

Note

- `statistic` should not cause errors or return missing values.
- The data is permuted in-place. Therefore, modifications to the data within `statistic` may lead to incorrect results. Since R has copy-on-modify semantics but C++ does not, it is recommended to pass `const` references when using Rcpp in `define_pmt`, as shown in the table above.

Examples

```
pmt("twosample.wilcoxon")

pmts("ksample")

x <- rnorm(5)
y <- rnorm(5, 1)

t <- define_pmt(
  inherit = "twosample",
  scoring = base::rank, # equivalent to "rank"
  statistic = function(...) function(x, y) sum(x)
)$test(x, y)$print()

t$scoring <- function(x) qnorm(rank(x) / (length(x) + 1)) # equivalent to "vw"
t$print()
```

```

t$n_permu <- 0
t$print()

r <- define_pmt(
  inherit = "twosample", n_permu = 1e5,
  statistic = function(x, y) {
    m <- length(x)
    n <- length(y)
    function(x, y) sum(x) / m - sum(y) / n
  }
)

rcpp <- define_pmt(
  inherit = "twosample", n_permu = 1e5,
  statistic = "[](const auto& x, const auto& y) {
    auto m = x.length();
    auto n = y.length();
    return [=](const auto& x, const auto& y) {
      return sum(x) / m - sum(y) / n;
    };
  }"
)

# equivalent
# rcpp <- define_pmt(
#   inherit = "twosample", n_permu = 1e5,
#   statistic = "[](const NumericVector& x, const NumericVector& y) {
#     R_xlen_t m = x.length();
#     R_xlen_t n = y.length();
#     return [m, n](const NumericVector& x, const NumericVector& y) -> double {
#       return sum(x) / m - sum(y) / n;
#     };
#   }"
# )

options(LearnNonparam.pmt_progress = FALSE)
system.time(r$test(x, y))
system.time(rcpp$test(x, y))

```

Quantile

Quantile Test

Description

Performs quantile test on a single sample. In addition, an estimation and a confidence interval for the desired quantile will be calculated.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::OneSampleTest](#) -> Quantile

Active bindings

prob The probability associated with the quantile.

correct Whether to apply continuity correction.

Methods**Public methods:**

- [Quantile\\$new\(\)](#)

Method `new()`: Create a new Quantile object.

Usage:

```
Quantile$new(
  type = c("asympt", "exact"),
  alternative = c("two_sided", "less", "greater"),
  null_value = 0,
  conf_level = 0.95,
  prob = 0.5,
  correct = TRUE
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`alternative` a character string specifying the alternative hypothesis.

`null_value` a number indicating the hypothesized value of the quantile.

`conf_level` a number between zero and one indicating the confidence level to use.

`prob` a number between zero and one indicating the probability associated with the quantile.

`correct` a logical indicating whether to apply continuity correction in the normal approximation for the p-value.

Returns: A Quantile object.

Examples

```
pmt(
  "onesample.quantile", prob = 0.5,
  null_value = 75, alternative = "greater",
  type = "asympt", correct = FALSE
)$test(Table1.1.1)$print()
```

```
pmt(
  "onesample.quantile",
  prob = 0.25, conf_level = 0.90
)$test(Table1.2.1)$conf_int
```

RatioMeanDeviance	<i>Ratio Mean Deviance Test</i>
-------------------	---------------------------------

Description

Performs ratio mean deviance test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> RatioMeanDeviance

Methods

Public methods:

- [RatioMeanDeviance\\$new\(\)](#)

Method `new()`: Create a new RatioMeanDeviance object.

Usage:

```
RatioMeanDeviance$new(  
  alternative = c("two_sided", "less", "greater"),  
  n_permu = 10000  
)
```

Arguments:

`alternative` a character string specifying the alternative hypothesis.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A RatioMeanDeviance object.

Examples

```
pmt(  
  "twosample.rmd",  
  alternative = "greater", n_permu = 0  
)$test(Table2.8.1)$print()
```

RCBDOneWay

*One-Way Test for Equal Means in RCBD***Description**

Performs F statistic based one-way test on samples collected in a randomized complete block design.

Super classes

[LearnNonparam:PermuTest](#) -> [LearnNonparam:RCBDTest](#) -> RCBDOneWay

Methods**Public methods:**

- [RCBDOneWay\\$new\(\)](#)

Method `new()`: Create a new RCBDOneWay object.

Usage:

```
RCBDOneWay$new(type = c("permu", "asyp"), n_permu = 10000)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A RCBDOneWay object.

Examples

```
t <- pmt(
  "rcbd.oneway", n_permu = 5000
)$test(Table4.4.3)$print()

t$type <- "asyp"
t
```

RCBDTest

*RCBDTest Class***Description**

Abstract class for tests on samples collected in randomized complete block designs.

Super class

[LearnNonparam:PermuTest](#) -> RCBDTest

ScoreSum

Two-Sample Test Based on Sum of Scores

Description

Performs sum of scores based two-sample test on samples. It is almost the same as two-sample wilcoxon rank sum test but uses more scoring systems.

Super classes

`LearnNonparam::PermuTest` -> `LearnNonparam::TwoSampleTest` -> `LearnNonparam::TwoSampleLocationTest`
-> `ScoreSum`

Methods

Public methods:

- `ScoreSum$new()`

Method `new()`: Create a new `ScoreSum` object.

Usage:

```
ScoreSum$new(  
  scoring = c("rank", "vw", "expon"),  
  alternative = c("two_sided", "less", "greater"),  
  null_value = 0,  
  n_permu = 10000  
)
```

Arguments:

`scoring` a character string specifying the scoring system.

`alternative` a character string specifying the alternative hypothesis.

`null_value` a number indicating the true value of the location shift.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A `ScoreSum` object.

Examples

```
pmt(  
  "twosample.scoresum", scoring = "expon",  
  alternative = "greater", n_permu = 0  
)$test(Table2.6.2)$print()
```

 SiegelTukey

Siegel-Tukey Test

Description

Performs Siegel-Tukey test on samples.

Super classes

LearnNonparam::PermuTest -> LearnNonparam::TwoSampleTest -> LearnNonparam::TwoSampleLocationTest
 -> LearnNonparam::Wilcoxon -> SiegelTukey

Methods**Public methods:**

- [SiegelTukey\\$new\(\)](#)

Method `new()`: Create a new SiegelTukey object.

Usage:

```
SiegelTukey$new(
  type = c("permu", "asyp"),
  alternative = c("two_sided", "less", "greater"),
  n_permu = 10000,
  correct = TRUE
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`alternative` a character string specifying the alternative hypothesis.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

`correct` a logical indicating whether to apply continuity correction in the normal approximation for the p-value.

Returns: A SiegelTukey object.

Examples

```
pmt(
  "twosample.siegel",
  alternative = "greater", n_permu = 0
)$test(Table2.8.1)$print()
```

Sign

*Two-Sample Sign Test***Description**

Performs two-sample sign test on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> [LearnNonparam::TwoSamplePairedTest](#)
-> Sign

Active bindings

correct Whether to apply continuity correction.

Methods**Public methods:**

- [Sign\\$new\(\)](#)

Method new(): Create a new Sign object.

Usage:

```
Sign$new(
  type = c("permu", "asyp"),
  alternative = c("two_sided", "less", "greater"),
  n_permu = 10000,
  correct = TRUE
)
```

Arguments:

type a character string specifying the way to calculate the p-value.

alternative a character string specifying the alternative hypothesis.

n_permu an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

correct a logical indicating whether to apply continuity correction in the normal approximation for the p-value.

Returns: A Sign object.

Examples

```
t <- pmt(
  "paired.sign",
  alternative = "greater", n_permu = 0
)$test(
  rep(c(+1, -1), c(12, 5)), rep(0, 17)
)$print()
```

```
t$type <- "asyp"  
t
```

Studentized

Multiple Comparison Based on Studentized Statistic

Description

Performs studentized statistic based multiple comparison on samples.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::KSampleTest](#) -> [LearnNonparam::MultipleComparison](#)
-> Studentized

Methods

Public methods:

- [Studentized\\$new\(\)](#)

Method `new()`: Create a new Studentized object.

Usage:

```
Studentized$new(  
  type = c("permu", "asyp"),  
  method = c("bonferroni", "tukey"),  
  scoring = c("none", "rank", "vw", "expon"),  
  conf_level = 0.95,  
  n_permu = 10000  
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`method` a character string specifying whether to use Bonferroni's method or Tukey's HSD method.

`scoring` a character string specifying the scoring system.

`conf_level` a number between zero and one indicating the family-wise confidence level to use.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

Returns: A Studentized object.

Examples

```
t <- pmt(
  "multcomp.studentized", method = "bonferroni"
)$test(Table3.3.1)$print()

t$type <- "asyp"
t

t$scoring <- "rank"
t

t$method <- "tukey"
t

t$scoring <- "none"
t

t$type <- "permu"
t
```

Table1.1.1

Sodium Contents

Description

Sodium contents (in mg) of 40 servings of a food product.

Usage

Table1.1.1

Format

An object of class `numeric` of length 40.

Source

Table 1.1.1

Table1.2.1	<i>Cycles Until Failure</i>
------------	-----------------------------

Description

The number of cycles (in thousands) that it takes for 20 door latches to fail.

Usage

Table1.2.1

Format

An object of class `numeric` of length 20.

Source

Table 1.2.1

Table2.1.1	<i>Test Scores</i>
------------	--------------------

Description

Test scores of 7 employees for comparison of methods of instruction.

Usage

Table2.1.1

Format

An object of class `list` of length 2.

Source

Table 2.1.1

Table2.3.1	<i>Runoff Minutes</i>
------------	-----------------------

Description

The numbers of minutes it took to obtain various amounts of runoff on each plot.

Usage

Table2.3.1

Format

An object of class `data.frame` with 8 rows and 2 columns.

Source

Table 2.3.1

Table2.6.1	<i>Hours Until Recharge</i>
------------	-----------------------------

Description

The numbers of hours that 2 brands of laptop computers function before battery recharging is necessary.

Usage

Table2.6.1

Format

An object of class `data.frame` with 4 rows and 2 columns.

Source

Table 2.6.1

Table2.6.2	<i>Cerium Amounts</i>
------------	-----------------------

Description

The amounts of cerium measured in samples of granite and basalt.

Usage

Table2.6.2

Format

An object of class data.frame with 6 rows and 2 columns.

Source

Table 2.6.2

Table2.8.1	<i>Ounces Of Beverage</i>
------------	---------------------------

Description

The amounts of liquid in randomly selected beverage containers before and after the filling process has been repaired.

Usage

Table2.8.1

Format

An object of class data.frame with 5 rows and 2 columns.

Source

Table 2.8.1

Table3.1.2	<i>Normal Samples</i>
------------	-----------------------

Description

Observations randomly sampled from normal populations with means 15, 25 and 30, respectively, and standard deviation 9.

Usage

Table3.1.2

Format

An object of class `data.frame` with 5 rows and 3 columns.

Source

Table 3.1.2

Table3.2.2	<i>Logarithms of Bacteria Counts</i>
------------	--------------------------------------

Description

Logarithms of counts of bacteria in 4 samples, which respectively were treated with 3 kills and left untreated for the control.

Usage

Table3.2.2

Format

An object of class `list` of length 4.

Source

Table 3.2.2

Table3.2.3	<i>Saltiness Scores</i>
------------	-------------------------

Description

Saltiness scores, on a scale of 1 to 5, assigned by a taste expert to samples of 3 food products that differ in the amounts of soymeal they contain.

Usage

Table3.2.3

Format

An object of class `list` of length 3.

Source

Table 3.2.3

Table3.3.1	<i>Percentages of Clay</i>
------------	----------------------------

Description

The percentages of clay in 6 samples of soil selected from 4 locations.

Usage

Table3.3.1

Format

An object of class `data.frame` with 6 rows and 4 columns.

Source

Table 3.3.1

Table3.4.1 *Phosphorus Contents*

Description

Phosphorus contents of plants under 4 mowing treatments.

Usage

Table3.4.1

Format

An object of class data.frame with 6 rows and 4 columns.

Source

Table 3.4.1

Table4.1.1 *Caloric Intake*

Description

The estimated daily caloric intake from dietary information provided using 2 methods by a group of college women.

Usage

Table4.1.1

Format

An object of class data.frame with 5 rows and 2 columns.

Source

Table 4.1.1

Table4.1.3	<i>Cholesterol Reduction</i>
------------	------------------------------

Description

Reduction in cholesterol after twins were given 2 drugs separately.

Usage

Table4.1.3

Format

An object of class data.frame with 17 rows and 2 columns.

Source

Table 4.1.3

Table4.4.3	<i>Yield Data</i>
------------	-------------------

Description

Yield data for a randomized complete block design in which 4 different types of tractors were used in tilling the soil. The blocking factor is location of the fields.

Usage

Table4.4.3

Format

An object of class data.frame with 4 rows and 6 columns.

Source

Table 4.4.3

Table4.5.3	<i>Randomized Complete Block with Ties</i>
------------	--

Description

A randomized complete block design with 4 treatments and 3 blocks.

Usage

Table4.5.3

Format

An object of class `data.frame` with 4 rows and 3 columns.

Source

Table 4.5.3

Table5.1.2	<i>Heterophils and Lymphocytes</i>
------------	------------------------------------

Description

Counts of the heterophils and lymphocytes in blood samples from 18 healthy rabbits.

Usage

Table5.1.2

Format

An object of class `data.frame` with 18 rows and 2 columns.

Source

Table5.1.2

Table5.2.2	<i>Scores of Projects</i>
------------	---------------------------

Description

Scores of 10 projects at a science fair.

Usage

Table5.2.2

Format

An object of class data.frame with 10 rows and 2 columns.

Source

Table5.2.2

Table5.4.2	<i>Satisfaction with Pain-Relief Treatment</i>
------------	--

Description

Patients' responses with 2 methods of relieving postoperative pain.

Usage

Table5.4.2

Format

An object of class data.frame with 2 rows and 3 columns.

Source

Table5.4.2

TwoSampleAssociationTest
TwoSampleAssociationTest Class

Description

Abstract class for two-sample association tests.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> [LearnNonparam::TwoSamplePairedTest](#)
-> TwoSampleAssociationTest

TwoSampleLocationTest *TwoSampleLocationTest Class*

Description

Abstract class for two-sample location tests.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> TwoSampleLocationTest

TwoSamplePairedTest *TwoSamplePairedTest Class*

Description

Abstract class for paired two-sample tests.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> TwoSamplePairedTest

TwoSampleTest *TwoSampleTest Class*

Description

Abstract class for two-sample tests.

Super class

[LearnNonparam::PermuTest](#) -> TwoSampleTest

 Wilcoxon

Two-Sample Wilcoxon Test

Description

Performs two-sample wilcoxon test on samples. In addition, an estimation and a confidence interval for the location shift will be calculated.

Super classes

[LearnNonparam::PermuTest](#) -> [LearnNonparam::TwoSampleTest](#) -> [LearnNonparam::TwoSampleLocationTest](#)
-> Wilcoxon

Active bindings

`correct` Whether to apply continuity correction.

Methods**Public methods:**

- [Wilcoxon\\$new\(\)](#)

Method `new()`: Create a new Wilcoxon object.

Usage:

```
Wilcoxon$new(
  type = c("permu", "asyp"),
  alternative = c("two_sided", "less", "greater"),
  null_value = 0,
  conf_level = 0.95,
  n_permu = 10000,
  correct = TRUE
)
```

Arguments:

`type` a character string specifying the way to calculate the p-value.

`alternative` a character string specifying the alternative hypothesis.

`null_value` a number indicating the true value of the location shift.

`conf_level` a number between zero and one indicating the confidence level to use.

`n_permu` an integer indicating number of permutations for the permutation distribution. If set to 0, all permutations will be used.

`correct` a logical indicating whether to apply continuity correction in the normal approximation for the p-value.

Returns: A Wilcoxon object.

Examples

```
pmt(  
  "twosample.wilcoxon",  
  alternative = "greater", n_permu = 0  
)$test(Table2.1.1)$print()
```

```
pmt(  
  "twosample.wilcoxon",  
  alternative = "less", n_permu = 0  
)$test(Table2.6.1)$print()
```

```
pmt(  
  "twosample.wilcoxon", conf_level = 0.90  
)$test(Table2.6.2)$conf_int
```


Index

* datasets

- Table1.1.1, 28
 - Table1.2.1, 29
 - Table2.1.1, 29
 - Table2.3.1, 30
 - Table2.6.1, 30
 - Table2.6.2, 31
 - Table2.8.1, 31
 - Table3.1.2, 32
 - Table3.2.2, 32
 - Table3.2.3, 33
 - Table3.3.1, 33
 - Table3.4.1, 34
 - Table4.1.1, 34
 - Table4.1.3, 35
 - Table4.4.3, 35
 - Table4.5.3, 36
 - Table5.1.2, 36
 - Table5.2.2, 37
 - Table5.4.2, 37
- AnsariBradley, 3
- association.corr (Correlation), 6
- CDF, 4
- ChiSquare, 5
- class.association
(TwoSampleAssociationTest), 38
- class.ksample (KSampleTest), 11
- class.multcomp (MultipleComparison), 12
- class.onesample (OneSampleTest), 12
- class.paired (TwoSamplePairedTest), 38
- class.pmt (PermuTest), 16
- class.rcbd (RCBDTest), 23
- class.table (ContingencyTableTest), 5
- class.twosample (TwoSampleTest), 38
- class.twosample.location
(TwoSampleLocationTest), 38
- ContingencyTableTest, 5
- Correlation, 6
- define_pmt (pmt), 17
- Difference, 7
- Friedman, 8
- ggplot2::stat_bin(), 17
- graphics::hist.default(), 17
- JonckheereTerpstra, 9
- KolmogorovSmirnov, 10
- KruskalWallis, 10
- ksample.jt (JonckheereTerpstra), 9
- ksample.kw (KruskalWallis), 10
- ksample.oneway (OneWay), 13
- KSampleTest, 11
- LearnNonparam::ContingencyTableTest, 5
- LearnNonparam::KSampleTest, 9, 10, 12, 13, 27
- LearnNonparam::MultipleComparison, 27
- LearnNonparam::OneSampleTest, 4, 21
- LearnNonparam::PermuTest, 3–15, 21–27, 38, 39
- LearnNonparam::RCBDTest, 8, 14, 23
- LearnNonparam::TwoSampleAssociationTest, 6
- LearnNonparam::TwoSampleLocationTest, 7, 24, 25, 39
- LearnNonparam::TwoSamplePairedTest, 6, 15, 26, 38
- LearnNonparam::TwoSampleTest, 3, 6, 7, 10, 15, 22, 24–26, 38, 39
- LearnNonparam::Wilcoxon, 25
- multcomp.studentized (Studentized), 27
- MultipleComparison, 12
- onesample.cdf (CDF), 4
- onesample.quantile (Quantile), 20
- OneSampleTest, 12

OneWay, [13](#)

Page, [14](#)

paired.difference (PairedDifference), [15](#)

paired.sign (Sign), [26](#)

PairedDifference, [15](#)

PermuTest, [16](#)

pmt, [17](#)

pmts (pmt), [17](#)

Quantile, [20](#)

RatioMeanDeviance, [22](#)

rcbd.friedman (Friedman), [8](#)

rcbd.oneway (RCBDOneWay), [23](#)

rcbd.page (Page), [14](#)

RCBDOneWay, [23](#)

RCBDTest, [23](#)

Rcpp::cppFunction(), [18](#)

ScoreSum, [24](#)

SiegelTukey, [25](#)

Sign, [26](#)

Studentized, [27](#)

table.chisq (ChiSquare), [5](#)

Table1.1.1, [28](#)

Table1.2.1, [29](#)

Table2.1.1, [29](#)

Table2.3.1, [30](#)

Table2.6.1, [30](#)

Table2.6.2, [31](#)

Table2.8.1, [31](#)

Table3.1.2, [32](#)

Table3.2.2, [32](#)

Table3.2.3, [33](#)

Table3.3.1, [33](#)

Table3.4.1, [34](#)

Table4.1.1, [34](#)

Table4.1.3, [35](#)

Table4.4.3, [35](#)

Table4.5.3, [36](#)

Table5.1.2, [36](#)

Table5.2.2, [37](#)

Table5.4.2, [37](#)

twosample.ansari (AnsariBradley), [3](#)

twosample.difference (Difference), [7](#)

twosample.ks (KolmogorovSmirnov), [10](#)

twosample.rmd (RatioMeanDeviance), [22](#)

twosample.scoresum (ScoreSum), [24](#)

twosample.siegel (SiegelTukey), [25](#)

twosample.wilcoxon (Wilcoxon), [39](#)

TwoSampleAssociationTest, [38](#)

TwoSampleLocationTest, [38](#)

TwoSamplePairedTest, [38](#)

TwoSampleTest, [38](#)

Wilcoxon, [39](#)