Package 'rfvimptest'

May 9, 2026

Type Package

Title Sequential Permutation Testing of Random Forest Variable Importance Measures

Version 0.1.4

Date 2025-05-08

Description Sequential permutation testing for statistical significance of predictors in random forests and other prediction methods. The main function of the package is rfvimptest(), which allows to test for the statistical significance of predictors in random forests using different (sequential) permutation test strategies [1]. The advantage of sequential over conventional permutation tests is that they are computationally considerably less intensive, as the sequential procedure is stopped as soon as there is sufficient evidence for either the null or the alternative hypothesis.
Reference:
[1] Hapfelmeier, A., Hornung, R. & Haller, B. (2023) Efficient permutation testing of variable importance measures by the example of random forests. Computational Statistics & Data Analysis 181:107689, <doi:10.1016/j.csda.2022.107689>.

License GPL-3

Depends R (>= 3.5.0)

Imports party, ranger, permimp

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Author Alexander Hapfelmeier [aut], Roman Hornung [aut, cre]

Maintainer Roman Hornung <hornung@ibe.med.uni-muenchen.de>

Repository CRAN

Date/Publication 2025-05-08 22:10:02 UTC

Contents

`allinone` *Apply all available (sequential) permutation testing approaches of variable importance measures with one function call*

Description

This is a helper function, which allows to perform all (sequential) permutation testing approaches of variable importance measures described in `rfvimptest` with a single function call. This may be useful for comparing the results obtained using the different approaches. Importantly, this function is computationally efficient by re-using the permuted variable importance values obtained for the conventional permutation test (that performs all `Mmax` permutations) for the other approaches. For details on the different approaches see `rfvimptest`.

Usage

```
allinone(
  data,
  yname,
  Mmax = 500,
  varnames = NULL,
  p0 = 0.06,
  p1 = 0.04,
  alpha = 0.05,
  beta = 0.2,
  A = 0.1,
  B = 10,
  h = 8,
  nperm = 1,
  ntree = 500,
  progressbar = TRUE,
  condinf = FALSE,
  ...
)
```

Arguments

`data`	A `data.frame` containing the variables in the model.
`yname`	Name of outcome variable.

Mmax	Maximum number of permutations used in each permutation test. Default is 500.
varnames	Optional. Names of the variables for which testing should be performed. By default all variables in data with the exception of the outcome variable are used.
p0	The value of the p-value in the null hypothesis (H0: p = p0) of SPRT and SAPT. Default is 0.06.
p1	The value of the p-value in the alternative hypothesis (H1: p = p1) of SPRT and SAPT. Default is 0.04.
alpha	The significance level of SPRT when p = p0. Also known as type I error. Default is 0.05.
beta	One minus the power of SPRT when p = p1. Also known as type II error. Default is 0.2.
A	The quantity A in the formula of SAPT. Default is 0.1 for a type I error of 0.05. Usually not changed by the user.
B	The quantity B in the formula of SAPT. Default is 10 (1/A) for a type I error of 0.05. Usually not changed by the user.
h	The quantity h in the formula for the sequential Monte Carlo p-value. The default value for h is 8. Larger values lead to more precise p-value estimates, but are computationally more expensive.
nperm	The numbers of permutations of the out-of-bag observations over which the results are averaged, when calculating the variable importance measure values. Default is 1. Larger values than 1 can only be considered when condinf=TRUE, that is, when using random forests with conditional inference trees (Hothorn et al., 2006) as base learners.
ntree	Number of trees per forest. Default is 500.
progressbar	Output the current progress of the calculations for each variable to the console? Default is TRUE.
condinf	Set this value to TRUE if random forests using conditional inference trees (Hothorn et al., 2006) should be used and to FALSE if classical random forests using CART trees should be used. Default is FALSE.
...	Further arguments passed to ranger::ranger (if condinf=FALSE) or party::cforest_unbiased() (if condinf=TRUE).

Value

Object of class allinone with elements

varimp	Variable importance for each considered independent variable.
testres	The results ("keep H0" vs. "accept H1") of the tests for each considered independent variable.
pvalues	The p-values of the tests for each considered independent variable. Note that p-values are only obtained for the method types "pval" and "complete".
stoppedearly	For each independent variable, whether the calculations stopped early ("yes") or the maximum of Mmax permutations was reached ("no").

perms	The number of permutations performed for each independent variable.
Mmax	Maximum number of permutations used in each permutation test.
ntree	Number of trees per forest.
comptime	The time the computations needed.

Author(s)

Alexander Hapfelmeier, Roman Hornung

References

- Breiman, L. (2001). Random forests. Mach Learn, 45:5-32, <doi:10.1023/A:1010933404324>.
- Coleman, T., Peng, W., Mentch, L. (2019). Scalable and efficient hypothesis testing with random forests. arXiv preprint arXiv:1904.07830, <doi:10.48550/arXiv.1904.07830>.
- Hapfelmeier, A., Hornung, R., Haller, B. (2023). Efficient permutation testing of variable importance measures by the example of random forests. Comput Stat Data Anal, 181:107689, <doi:10.1016/j.csda.2022.107689>.
- Hapfelmeier, A., Ulm, K. (2013). A new variable selection approach using Random Forests. CSDA 60:50–69, <doi:10.1016/j.csda.2012.09.020>.
- Hapfelmeier, A., Hothorn, T., Ulm, K., Strobl, C. (2014). A new variable importance measure for random forests with missing data. Stat Comput 24:21–34, <doi:10.1007/s112220129349-1>.
- Hothorn, T., Hornik, K., Zeileis, A. (2006). Unbiased Recursive Partitioning: A Conditional Inference Framework. J Comput Graph Stat 15(3):651–674, <doi:10.1198/106186006X133933>.
- Wright, M. N., Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. J Stat Softw 77:1-17, <doi:10.18637/jss.v077.i01>.

See Also

rfvimptest

Examples

```
# Load package:
library("rfvimptest")

# Set seed to obtain reproducible results:
set.seed(1234)

# Load example data:
data(hearth2)

# NOTE: For illustration purposes very small numbers of maximum
# permutations are considered in the below examples.
# This number would be much too small for actual applications.
# The default number is Max=500.
```

```
# When using condinf=FALSE (default) the results for the two-sample
# permutation tests are not obtained:
(ptest <- allinone(data=hearth2, yname="Class",  Mmax=20))

# Variable importance values with p-values from the Monte Carlo p-value
# and the complete approach:
ptest$varimp
ptest$pvalues$pval
ptest$pvalues$complete


# When setting condinf=TRUE the results are obtained for all approaches,
# that is, including those for the two-sample permutation tests
# (in this illustration very small number of trees ntree=30 are used,
# in practice much larger numbers should be used; the default is ntree=500):
(ptest_ci <- allinone(data=hearth2, yname="Class", condinf=TRUE, ntree=30, Mmax=10))
ptest_ci$testres
```

`hearth2` *Data on Coronary Artery Disease*

Description

This data includes 294 patients undergoing angiography at the Hungarian Institute of Cardiology in Budapest between 1983 and 1987.

Format

A data frame with 294 observations, ten covariates and one two-class outcome variable

Details

The variables are as follows:

- `age`. numeric. Age in years
- `sex`. factor. Sex (1 = male; 0 = female)
- `chest_pain`. factor. Chest pain type (1 = typical angina; 2 = atypical angina; 3 = non-anginal pain; 4 = asymptomatic)
- `trestbps`. numeric. Resting blood pressure (in mm Hg on admission to the hospital)
- `chol`. numeric. Serum cholestoral in mg/dl
- `fbs`. factor. Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
- `restecg`. factor. Resting electrocardiographic results (1 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV); 0 = normal)
- `thalach`. numeric. Maximum heart rate achieved

- exang. factor. Exercise induced angina (1 = yes; 0 = no)
- oldpeak. numeric. ST depression induced by exercise relative to rest
- Class. factor. Disease satus (1 = no disease; 2 = coronary artery disease)

The original openML dataset was pre-processed in the following way:

1. The variables were re-named according to the description given on openML.

2. The missing values which were coded as "-9" were replaced by NA values.

3. The variables slope, ca, and thal were excluded, because these featured too many missing values.

4. The categorical covariates were transformed into factors.

5. There were 6 restecg values of "2" which were replaced by "1".

6. The missing values were imputed: The missing values of the numerical covariates were replaced by the means of the corresponding non-missing values. The missing values of the categorical covariates were replaced by the modes of the corresponding non-missing values.

Note that this dataset is also included in a slightly different form in the R package ordinalForest (version 2.4-2) under the name hearth. The only difference is that in hearth2, the ordinal outcome variable Class was transformed into a two-class outcome by only differentiating between diseased vs. healthy, rather than differentiating between different levels of disease severity.

Source

OpenML: data.name: heart-h, data.id: 1565, link: https://www.openml.org/d/1565/

References

- Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J.-J., Sandhu, S., Guppy, K. H., Lee, S., Froelicher, V. (1989) International application of a new probability algorithm for the diagnosis of coronary artery disease. The American Journal Of Cardiology, 64, 304–310.
- Vanschoren, J., van Rijn, J. N., Bischl, B., Torgo, L. (2013) OpenML: networked science in machine learning. SIGKDD Explorations, 15(2), 49–60.

Examples

```
data(hearth2)

table(hearth2$Class)
dim(hearth2)

head(hearth2)
```

`rfvimptest` *Testing the statistical significance of predictors in random forests using sequential permutation testing*

Description

Implements several strategies for testing the statistical significance of predictors in random forests using sequential permutation testing procedures based on the permutation variable importance measure. See Hapfelmeier et al. (2023) for details.

Usage

```
rfvimptest(
  data,
  yname,
  Mmax = 500,
  varnames = NULL,
  p0 = 0.06,
  p1 = 0.04,
  alpha = 0.05,
  beta = 0.2,
  A = 0.1,
  B = 10,
  h = 8,
  nperm = 1,
  ntree = 500,
  progressbar = TRUE,
  test = c("general", "twosample")[1],
  type = c("SPRT", "SAPT", "pval", "certain", "complete")[1],
  condinf = FALSE,
  ...
)
```

Arguments

`data`	A `data.frame` containing the variables in the model.
`yname`	Name of outcome variable.
`Mmax`	Maximum number of permutations used in each permutation test. Default is 500.
`varnames`	Optional. Names of the variables for which testing should be performed. By default all variables in `data` with the exception of the outcome variable are used.
`p0`	The value of the p-value in the null hypothesis (H0: p = p0) of SPRT and SAPT. Default is 0.06.
`p1`	The value of the p-value in the alternative hypothesis (H1: p = p1) of SPRT and SAPT. Default is 0.04.

alpha	The significance level of SPRT when p = p0. Also known as type I error. Default is 0.05.
beta	One minus the power of SPRT when p = p1. Also known as type II error. Default is 0.2.
A	The quantity A in the formula of SAPT. Default is 0.1 for a type I error of 0.05. Usually not changed by the user.
B	The quantity B in the formula of SAPT. Default is 10 (1/A) for a type I error of 0.05. Usually not changed by the user.
h	The quantity h in the formula for the sequential Monte Carlo p-value. The default value for h is 8. Larger values lead to more precise p-value estimates, but are computationally more expensive.
nperm	The numbers of permutations of the out-of-bag observations over which the results are averaged, when calculating the variable importance measure values. Default is 1. Larger values than 1 can only be considered when `condinf=TRUE`, that is, when using random forests with conditional inference trees (Hothorn et al., 2006) as base learners.
ntree	Number of trees per forest. Default is 500.
progressbar	Output the current progress of the calculations for each variable to the console? Default is TRUE.
test	Type of the permutation test to perform. This can be either "general" or "twosample", where "general" refers to the usual (sequential) permutation test and "twosample" refers to the two-sample (sequential) permutation test. For the latter, see also Coleman et al. (2019). Note, however, that "twosample" is experimental and should not be used for formal testing. See the details section below.
type	Type of the sequential method to use in the permutation tests. The choices are: "SPRT", "SAPT", "pval", "certain", and "complete". See the 'Details' section below for details.
condinf	Set this value to `TRUE` if random forests using conditional inference trees (Hothorn et al., 2006) should be used and to `FALSE` if classical random forests using CART trees should be used. Default is `FALSE`.
...	Further arguments passed to `ranger::ranger` (if `condinf=FALSE`) or `party::cforest_unbiased()` (if `condinf=TRUE`).

Details

Only the general permutation test (`test="general"`) controls the type I error. In contrast, the two-sample permutation test (`test="twosample"`) is associated with inflated type I error, which can lead to false positive findings. An advantage of the two-sample permutation test is that it is very fast. Therefore, this experimental approach may be used as an informal screening tool for finding informative variables. It is, however, not a valid testing procedure. Note also that the paper of Coleman et al. (2019) on which the two-sample test is based has not yet been published in a peer-reviewed journal and that the theory underlying this procedure might thus still need further review.

SPRT (`type="SPRT"`) and SAPT (`type="SAPT"`) are similar sequential procedures, where SPRT is faster with respect to accepting H0, that is, detecting non-informative variables, whereas SAPT

is faster with respect to accepting H1, that is, detecting informative variables. Therefore, SPRT may be preferred for datasets with only few informative variables, whereas SAPT is preferable for datasets with many informative variables. The Monte Carlo p-value based testing procedure (`type="pval"`) should be used, when p-values are required. The choice `type="complete"` offers a conventional permutation test (that is, without sequential testing) (Hapfelmeier and Ulm, 2013). This choice is computationally the most intensive. Lastly, the choice `type="certain"` is similar to `type="complete"`, but performs early stopping by ending the permutation iterations as soon as it is certain which outcome the conventional permutation test would take. That is, `type="certain"` can be considered as a computationally more effective version of `type="complete"`.

Value

Object of class `rfvimptest` with elements

`testtype`	Type of the permutation test performed and sequential method used.
`varimp`	Variable importance for each considered independent variable.
`testres`	The results ("keep H0" vs. "accept H1") of the tests for each considered independent variable.
`pvalues`	The p-values of the tests for each considered independent variable. Note that p-values are only obtained for the method types "pval" and "complete".
`stoppedearly`	For each independent variable, whether the calculations stopped early ("yes") or the maximum of `Mmax` permutations was reached ("no").
`perms`	The number of permutations performed for each independent variable.
`Mmax`	Maximum number of permutations used in each permutation test.
`ntree`	Number of trees per forest.
`comptime`	The time the computations needed.

Author(s)

Alexander Hapfelmeier, Roman Hornung

References

- Breiman, L. (2001). Random forests. Mach Learn, 45:5-32, <doi:10.1023/A:1010933404324>.
- Coleman, T., Peng, W., Mentch, L. (2019). Scalable and efficient hypothesis testing with random forests. arXiv preprint arXiv:1904.07830, <doi:10.48550/arXiv.1904.07830>.
- Hapfelmeier, A., Hornung, R., Haller, B. (2023). Efficient permutation testing of variable importance measures by the example of random forests. Comput Stat Data Anal, 181:107689, <doi:10.1016/j.csda.2022.107689>.
- Hapfelmeier, A., Ulm, K. (2013). A new variable selection approach using Random Forests. CSDA 60:50–69, <doi:10.1016/j.csda.2012.09.020>.
- Hapfelmeier, A., Hothorn, T., Ulm, K., Strobl, C. (2014). A new variable importance measure for random forests with missing data. Stat Comput 24:21–34, <doi:10.1007/s112220129349-1>.
- Hothorn, T., Hornik, K., Zeileis, A. (2006). Unbiased Recursive Partitioning: A Conditional Inference Framework. J Comput Graph Stat 15(3):651–674, <doi:10.1198/106186006X133933>.

- Wright, M. N., Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. J Stat Softw 77:1-17, <doi:10.18637/jss.v077.i01>.

Examples

```
## Load package:
library("rfvimptest")

## Set seed to obtain reproducible results:
set.seed(1234)

# Load example data:
data(hearth2)

# NOTE: For illustration purposes a very small number (Mmax=20) of maximum
# permutations is considered. This number would be much too small for actual
# applications. The default number is Max=500.

# By default, SPRT is performed:
(ptest_sprt <- rfvimptest(data=hearth2, yname="Class", Mmax=20))
ptest_sprt$varimp
ptest_sprt$testres

# Calculation of p-values using the Monte Carlo p-value based testing procedure:
(ptest_pval <- rfvimptest(data=hearth2, yname="Class", type="pval", Mmax=20))
ptest_pval$pvalues

# If the frequency of informative variables is expected to be high SAPT can be used:
(ptest_sapt <- rfvimptest(data=hearth2, yname="Class", type="SAPT", Mmax=20))
ptest_sapt$testres


# If it is only of interest to test specific variables in the dataset these variables
# should be passed to rfvimptest() vias the argument 'varnames' because this
# reduces the computational burden considerably:

(ptest_twovar <- rfvimptest(data=hearth2, yname="Class", varnames=c("age", "sex"), Mmax=20))
ptest_twovar$varimp
ptest_twovar$testres


# Two-sample permutation test procedures:

# NOTE: These should be used only for informal screening for informative variables.
# They are not valid statistical tests.

# Here, the maximum number of permutations can be much higher because it is necessary
# here to construct a new forest for each permutation:
rfvimptest(data=hearth2, yname="Class", test="twosample", condinf=TRUE, Mmax=1000)

rfvimptest(data=hearth2, yname="Class", test="twosample", type="pval", condinf=TRUE, Mmax=1000)
```

```
rfvimptest(data=hearth2, yname="Class", test="twosample", type="SAPT", condinf=TRUE, Mmax=1000)
```

`threshold_values` *Threshold values of a SPRT and SAPT, and calculation of sequential Monte Carlo p-values*

Description

Function to produce threshold values for the number of cumulative successes a sequential permutation test of `type="SRPT"` or `type="SAPT"` needs to show in order to "keep H0" or "accept H1" early in the sequence of tests.

Usage

```
threshold_values(
  p0 = 0.06,
  p1 = 0.04,
  alpha = 0.05,
  beta = 0.2,
  A = 0.1,
  B = 10,
  Mmax = 500,
  type = c("SPRT", "SAPT")[1]
)
```

Arguments

`p0`	The value of the p-value in the null hypothesis (H0: p = p0) of SPRT and SAPT. Default is 0.06.
`p1`	The value of the p-value in the alternative hypothesis (H1: p = p1) of SPRT and SAPT. Default is 0.04.
`alpha`	The significance level of SPRT when p = p0. Also known as type I error. Default is 0.05.
`beta`	One minus the power of SPRT when p = p1. Also known as type II error. Default is 0.2.
`A`	The quantity A in the formula of SAPT. Default is 0.1 for a type I error of 0.05. Usually not changed by the user.
`B`	The quantity B in the formula of SAPT. Default is 10 (1/A) for a type I error of 0.05. Usually not changed by the user.
`Mmax`	Maximum number of permutations used in each permutation test. Default is 500.
`type`	Type of the sequential test method. The choices are "SPRT" and "SAPT".

Details

An example of how to calculate sequential Monte Carlo p-values is also given.

The function and examples are intended to support the use of SPRT, SAPT and sequential p-values for any type of modelling approach, even outside the application in `rfvimptest::rfvimptest()`.

Value

List of two vectors, each containing `Mmax` threshold values for the number of cumulative successes a sequential permutation test of `type="SRPT"` or `type="SAPT"` needs to show in order to "keep H0" or "accept H1" early in the sequence of tests.

Author(s)

Alexander Hapfelmeier, Roman Hornung

References

- Wald, A. (1945). Sequential tests of statistical hypotheses. Ann. Math. Stat. 16, 117-186, <doi:10.1214/aoms/1177731118>.
- Lock, R.H. (1991). A sequential approximation to a permutation test. Commun. Stat., Simul. Comput. 20, 341-363, <doi:10.1080/03610919108812956>.
- Besag, J., Clifford, P. (1991). Sequential Monte Carlo p-values. Biometrika 78, 301-304, <doi:10.1093/biomet/78.2.301>.
- Hapfelmeier, A., Hornung, R., Haller, B. (2023). Efficient permutation testing of variable importance measures by the example of random forests. Comput Stat Data Anal, 181:107689, <doi:10.1016/j.csda.2022.107689>.

Examples

```
## Load package:
library("rfvimptest")

## Set seed to obtain reproducible results:
set.seed(1234)


# Load example data
data(hearth2)


# Calculation of a SAPT

# Create critical values for a SAPT
myvals <- threshold_values(p0 = 0.06, p1 = 0.04, A = 0.1, B = 10, Mmax = 200, type = "SAPT")
myvals

# Fit a model to the original data
mod <- glm(I(Class == 2) ~ ., data = hearth2)
```

```
summary(mod)

# Derive a statistic of interest from the model. Here, the negative AIC is used as a statistic
# to be maximised.
stat <- -1 * mod$aic

# Perform the permutations, extract the statistics and stop early according to the critical values
myresult <- sapply(setdiff(names(hearth2), "Class"), function(j) {
  permdat <- hearth2
  permstats <- c()
  for (i in 1:length(myvals[[1]])) {
    permdat[, j] <- sample(permdat[, j])
    permstats <- c(permstats, -1 * glm(I(Class == 2) ~ ., data = permdat)$aic)
   if (sum(permstats >= stat) >= myvals[[1]][i] | sum(permstats >= stat) <= myvals[[2]][i]) break
  }
  permstats
})

# Statistics obtained after permutation of the data
myresult

# Number of permutations performed until (early) stopping
sapply(myresult, length)

# Derive the SAPT decisions from the statistics
sapply(myresult, function(x) {
  if (sum(x >= stat) >= myvals[[1]][length(x)]) return("keep H0")
  else if (sum(x >= stat) <= myvals[[2]][length(x)]) return("accept H1")
  else ifelse(sum(x >= stat) / length(myvals[[1]]) <= 0.05, "accept H1", "keep H0")
})


# Calculation of sequential Monte Carlo p-values

h <- 10 # Parameter h of the sequential Monte Carlo p-value calculation.

# Perform the permutations, extract the statistics and stop early when h successes are reached
mypvals <- sapply(setdiff(names(hearth2), "Class"), function(j) {
  permdat <- hearth2
  permstats <- c()
  Mmax <- length(myvals[[1]])
  for (i in 1:Mmax) {
    permdat[, j] <- sample(permdat[, j])
    permstats <- c(permstats, -1 * glm(I(Class == 2) ~ ., data = permdat)$aic)
    d <- sum(permstats >= stat)
    if (d == h) {pval <- d/length(permstats); m <- i; break}
    else if (i == Mmax) {pval <- (d + 1)/(Mmax + 1); m <- i}
  }
  c(pval, m)
})

# p-values and number of permutations performed until (early) stopping
rownames(mypvals) <- c("p-value", "m")
```

```
t(mypvals)
```

Index