Package 'permimp'

March 28, 2025

Type Package

Title Conditional Permutation Importance

Version 1.1-0

Date 2025-03-26

Author Dries Debeer [aut, cre],
Torsten Hothorn [aut],
Carolin Strobl [aut]

Maintainer Dries Debeer <debeer.dries@gmail.com>

Description An add-on to the 'party' package, with a faster implementation of the partial-conditional permutation importance for random forests. The standard permutation importance is implemented exactly the same as in the 'party' package. The conditional permutation importance can be computed faster, with an option to be backward compatible to the 'party' implementation. The package is compatible with random forests fit using the 'party' and the 'randomForest' package. The methods are described in Strobl et al. (2007) <doi:10.1186/1471-2105-8-25> and Debeer and Strobl (2020) <doi:10.1186/s12859-020-03622-2>.

Depends R (>= 3.6.0)

Imports graphics, grDevices, ipred (>= 0.9-6), methods, party (>= 1.3-3), pbapply, randomForest (>= 4.6-14), stats, survival (>= 2.44-1.1), utils

Suggests knitr, rmarkdown, scales (>= 0.5.0), testthat

License GPL-2 | GPL-3

VignetteBuilder knitr

URL https://ddebeer.github.io/permimp/

NeedsCompilation no

Repository CRAN

Date/Publication 2025-03-28 10:50:02 UTC

Contents

permimp-package 2
permimp 3
ranks 7
selFreq 8
VarImp 9
VarImp-Methods 10

Index **13**

`permimp-package` *Conditional Permutation Importance*

Description

An add-on to the 'party' package, with a faster implementation of the partial-conditional permutation importance for random forests. The standard permutation importance is implemented exactly the same as in the 'party' package. The conditional permutation importance can be computed faster, with an option to be backward compatible to the 'party' implementation. The package is compatible with random forests fit using the 'party' and the 'randomForest' package. The methods are described in Strobl et al. (2007) <doi:10.1186/1471-2105-8-25> and Debeer and Strobl (2020) <doi:10.1186/s12859-020-03622-2>.

Details

Index of help topics:

`VarImp`	`VarImp Objects`
`VarImp-methods`	`Methods for VarImp Objects`
`permimp`	`Random Forest Permutation Importance for random forests`
`permimp-package`	`Conditional Permutation Importance`
`ranks`	`Reversed Rankings`
`selFreq`	`Predictor Selection Frequency in Random Forests`

Author(s)

Maintainer: Dries Debeer <dries.debeer@uzh.ch>

Authors:

- Carolin Strobl
- Torsten Hothorn

permimp *Random Forest Permutation Importance for random forests*

Description

Standard and partial/conditional permutation importance for random forest-objects fit using the **party** or **randomForest** packages, following the permutation principle of the 'mean decrease in accuracy' importance in **randomForest** . The partial/conditional permutation importance is implemented differently, selecting the predictions to condition on in each tree using Pearson Chi-squared tests applied to the by-split point-categorized predictors. In general the new implementation has similar results as the original `varimp` function. With `asParty = TRUE`, the partial/conditional permutation importance is fully backward-compatible but faster than the original `varimp` function in **party**.

Usage

```
permimp(object, ...)
## S3 method for class 'randomForest'
permimp(object, nperm = 1, OOB = TRUE, scaled = FALSE,
     conditional = FALSE, threshold = .95, whichxnames = NULL,
     thresholdDiagnostics = FALSE, progressBar = interactive(), do_check = TRUE,
     oldSeedSelection = FALSE, cl = NULL, ...)
## S3 method for class 'RandomForest'
permimp(object, nperm = 1, OOB = TRUE, scaled = FALSE,
     conditional = FALSE, threshold = .95, whichxnames = NULL,
     thresholdDiagnostics = FALSE, progressBar = interactive(),
     pre1.0_0 = conditional, AUC = FALSE, asParty = FALSE, mincriterion = 0,
     oldSeedSelection = FALSE, cl = NULL, ...)
```

Arguments

`object`	an object as returned by `cforest` or `randomForest`.
`mincriterion`	the value of the test statistic or 1 - p-value that must be exceeded in order to include a split in the computation of the importance. The default `mincriterion = 0` guarantees that all splits are included.
`conditional`	a logical that determines whether unconditional or conditional permutation is performed.
`threshold`	the threshold value for (1 - p-value) of the association between the predictor of interest and another predictor, which must be exceeded in order to include the other predictor in the conditioning scheme for the predictor of interest (only relevant if `conditional = TRUE`). A threshold value of zero includes all other predictors.
`nperm`	the number of permutations performed.
`OOB`	a logical that determines whether the importance is computed from the out-of-bag sample or the learning sample (not suggested).

`pre1.0_0` Prior to **party** version 1.0-0, the actual data values were permuted according to the original permutation importance suggested by Breiman (2001). Now the assignments to child nodes of splits in the variable of interest are permuted as described by Hapfelmeier et al. (2012), which allows for missing values in the predictors and is more efficient with respect to memory consumption and computing time. This method does not apply to the conditional permutation importance, nor to random forests that were not fit using the **party** package.

`scaled` a logical that determines whether the differences in prediction accuracy should be scaled by the total (null-model) error.

`AUC` a logical that determines whether the Area Under the Curve (AUC) instead of the accuracy is used to compute the permutation importance (cf. Janitza et al., 2012). The AUC-based permutation importance is more robust towards class imbalance, but it is only applicable to binary classification.

`asParty` a logical that determines whether or not exactly the same values as the original `varimp` function in **party** should be obtained.

`whichxnames` a character vector containing the predictor variable names for which the permutation importance should be computed. Only use when aware of the implications, see section 'Details'.

`thresholdDiagnostics` a logical that specifies whether diagnostics with respect to the threshold-value should be prompted as warnings.

`progressBar` a logical that determines whether a progress bar should be displayed.

`do_check` a logical that determines whether a check requiring user input should be included.

`oldSeedSelection` a logical that determines whether the selection of random numbers should be the same is in the 1.1 version of the package. The default is `FALSE`, so that seeds are generated for each tree, and the results are reproducible, also when parallel processing is used.

`cl` A cluster object created by `makeCluster`, or an integer to indicate number of child-processes (integer values are ignored on Windows) for parallel evaluations (see Details on parallel computing). `NULL` (default) refers to sequential evaluation.

`...` additional arguments to be passed to the Methods

Details

Function `permimp` is highly comparable to `varimp` in **party**, but the partial/conditional variable importance has a different, more efficient implementation. Compared to the original `varimp` in **party**, `permimp` applies a different strategy to select the predictors to condition on (ADD REFERENCE TO PAPER).

With `asParty = TRUE`, permimp returns exactly the same values as `varimp` in **party**, but the computation is done more efficiently.

If `conditional = TRUE`, the importance of each variable is computed by permuting within a grid defined by the predictors that are associated (with 1 - p-value greater than `threshold`) to the vari-

able of interest. The `threshold` can be interpreted as a parameter that moves the permutation importance across a dimension from fully conditional (`threshold = 0`) to completely unconditional (`threshold = 1`), see Debeer and Strobl (2020).

Using the `wichxnames` argument, the computation of the permutation importance can be limited to a smaller number of specified predictors. Note, however, that when `conditional = TRUE`, the (other) predictors to condition on are also limited to this selection of predictors. Only use when fully aware of the implications.

For parallel processing, the **pbapply** package, a wrapper around the **parallel** package is used. Parallel processing can be enabled through the `cl` argument. `parLapply` is called when `cl` is a 'cluster' object, `mclapply` is called when `cl` is an integer.

When doing parallel processing, other objects might need to pushed to the workers, and random numbers must be handled with care (see the Examples of the **pbapply** package).

When using parallel processing, showing the progress bar increases the communication overhead between the main process and nodes / child processes compared to the parallel equivalents of the functions without the progress bar. The functions fall back to their original equivalents when `progressBar = FALSE`. This is the default when `interactive()` is `FALSE` (i.e. called from command line R script)

For further details, please refer to the documentation of `varimp`.

Value

An object of class `varimp`, with the mean decrease in accuracy as its `$values`.

References

Leo Breiman (2001). Random Forests. *Machine Learning*, 45(1), 5–32.

Alexander Hapfelmeier, Torsten Hothorn, Kurt Ulm, and Carolin Strobl (2012). A New Variable Importance Measure for Random Forests with Missing Data. *Statistics and Computing*, https://link.springer.com/article/10.1007/s11222-012-9349-1

Torsten Hothorn, Kurt Hornik, and Achim Zeileis (2006b). Unbiased Recursive Partitioning: A Conditional Inference Framework. *Journal of Computational and Graphical Statistics*, **15** (3), 651-674. Preprint available from https://www.zeileis.org/papers/Hothorn+Hornik+Zeileis-2006.pdf

Silke Janitza, Carolin Strobl and Anne-Laure Boulesteix (2013). An AUC-based Permutation Variable Importance Measure for Random Forests. BMC Bioinformatics.2013, **14** 119. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-119

Carolin Strobl, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis (2008). Conditional Variable Importance for Random Forests. *BMC Bioinformatics*, **9**, 307. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-9-307

Debeer Dries and Carolin Strobl (2020). Conditional Permutation Importance Revisited. *BMC Bioinformatics*, **21**, 307. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-020-03622-2

See Also

`varimp`, `VarImp`

Examples

```
### for RandomForest-objects, by party::cforest()
set.seed(290875)
readingSkills.cf <- party::cforest(score ~ ., data = party::readingSkills,
                           control = party::cforest_unbiased(mtry = 2, ntree = 25))

### conditional importance, may take a while...
# party implementation:
set.seed(290875)
party::varimp(readingSkills.cf, conditional = TRUE)
# faster implementation but same results
set.seed(290875)
permimp(readingSkills.cf, conditional = TRUE, asParty = TRUE)

# different implementation with similar results
set.seed(290875)
permimp(readingSkills.cf, conditional = TRUE, asParty = FALSE)

### standard (unconditional) importance is unchanged
set.seed(290875)
party::varimp(readingSkills.cf)
set.seed(290875)
permimp(readingSkills.cf, oldSeedSelection = TRUE)


###
set.seed(290875)
readingSkills.rf <- randomForest::randomForest(score ~ ., data = party::readingSkills,
                           mtry = 2, ntree = 25, importance = TRUE,
                           keep.forest = TRUE, keep.inbag = TRUE)


### (unconditional) Permutation Importance
set.seed(290875)
permimp(readingSkills.rf, do_check = FALSE)

# very close to
readingSkills.rf$importance[,1]

### Conditional Permutation Importance
set.seed(290875)
permimp(readingSkills.rf, conditional = TRUE, threshold = .8, do_check = FALSE)

## Not run:
### Parallel processing - Windows
# Only relevant for large trees, for small trees, there may not even be a
# 'speed up', but a 'slow down'

# Make a larger forest
set.seed(290875)
readingSkills.cf <- party::cforest(score ~ ., data = party::readingSkills,
                                   control = party::cforest_unbiased(mtry = 2,
```

```
                                                                    ntree = 200))

  # sequentiall processing
  set.seed(290875)
  system.time(print(permimp(readingSkills.cf, conditional = TRUE, asParty = FALSE)))

  # parallel processing
  # note that the results are reproducible despite using multiple cores
  cluster <- parallel::makeCluster(2)

  set.seed(290875)
  system.time(print(permimp(readingSkills.cf, conditional = TRUE,
                            asParty = FALSE, cl = cluster, progressBar = FALSE)))
  parallel::stopCluster(cluster)

## End(Not run)
```

---

ranks *Reversed Rankings*

Description

Method for giving the reversed rankings of the numerical values of a vector or VarImp object.

Usage

```
   ranks(x, note = TRUE, ...)
## Default S3 method:
ranks(x, note = TRUE, ...)
## S3 method for class 'VarImp'
ranks(x, note = TRUE, ...)
```

Arguments

x	an object to be reverse ranked.
note	a logical specifying whether the (reversed) rankings should be printed instead of the importance values.
...	additional arguments to be passed to rank.

Details

The ranks function is nothing more than (length(x) - rank(x, ...) + 1L). But it also works for objects of class VarImp.

Value

A vector containing the reversed rankings.

Examples

```
## High Jump data
HighJumps <- c(Anna = 1.45, Betty = 1.53, Cara = 1.37, Debby = 1.61,
               Emma = 1.29, Hanna = 1.44, Juno = 1.71)
HighJumps
## ranking of high jump data
ranks(HighJumps)
```

---

selFreq *Predictor Selection Frequency in Random Forests*

Description

counts how many times each predictor variable was selected for splitting in a random forest. Only implemented for cforest form the **party** package.

Usage

```
selFreq(object, whichxnames = NULL)
```

Arguments

object	an object as returned by cforest.
whichxnames	a character vector containing the predictor variable names that for which the permutation importance should be computed. See section 'Details'.

Details

Function selFreq counts how many times each predictor variable was selected for splitting in a random forest. In the current implementation this selFreq can only be applied to random forests as returned by cforest.

Value

An object of class VarImp, with as $values the mean of the sum of the selection frequencies across the trees.

See Also

VarImp,

Examples

```
set.seed(290875)
readingSkills.cf <- party::cforest(score ~ ., data = party::readingSkills,
                            control = party::cforest_unbiased(mtry = 2, ntree = 100))

## Selection Frequency
selFreq(readingSkills.cf)
```

VarImp *VarImp Objects*

Description

A class for random forest variable importance measures VarImp objects.

Usage

```
as.VarImp(object, ...)

## S3 method for class 'data.frame'
## S3 method for class 'data.frame'
as.VarImp(object, FUN = mean,
             type = c("Permutation", "Conditional Permutation",
                      "Selection Frequency", "See Info"),
             info = NULL, ...)

## S3 method for class 'matrix'
## S3 method for class 'matrix'
as.VarImp(object, FUN = mean,
             type = c("Permutation", "Conditional Permutation",
                      "Selection Frequency", "See Info"),
             info = NULL, ...)

## S3 method for class 'numeric'
## S3 method for class 'numeric'
as.VarImp(object, perTree = NULL,
              type = c("Permutation", "Conditional Permutation",
                       "Selection Frequency", "See Info"),
              info = NULL, ...)

is.VarImp(VarImp)
```

Arguments

object	an R object.
perTree	a matrix or data frame of size ntree x p containing the variable importance measures for each tree in the random forest.
type	a character indicating the type of variable importance measure.
info	a list with additional information about the variable importance measure.
FUN	a function to compute the variable importance. See section 'Details'.
VarImp	an object of the class VarImp.
...	additional arguments.

Details

as.VarImp creates an object of class 'VarImp'. When object is a matrix or a data.frame, the final values are computed by applying FUN to its columns. is.VarImp returns a logical indicating whether the evaluated object is of class 'VarImp'.

See Also

VarImp-methods

Examples

```
## Matrix of fake importance measures per Tree
set.seed(290875)
ntree <- 500
p <- 15
fakeVIM <- matrix(rnorm(ntree * p), nrow = ntree, ncol = p,
           dimnames = list(paste0("pred", seq_len(ntree)), paste0("pred", seq_len(p))))
is.VarImp(fakeVIM)

## make a 'VarImp' object
fakeVarImp <- as.VarImp(fakeVIM, type = "See Info",
                  info = list("The Vims are based on fake data.",
                  "The mean was used to aggregate across the trees"))
is.VarImp(fakeVarImp)
```

VarImp-Methods *Methods for VarImp Objects*

Description

Methods for computing on VarImp objects..

Usage

```
## S3 method for class 'VarImp'
plot(x, nVar = length(x$values), type = c("bar", "box", "dot", "rank"),
                      sort = TRUE, interval = c( "no", "quantile", "sd"),
                      intervalProbs = c(.25, .75), intervalColor = NULL,
                      horizontal = FALSE, col = NULL, pch = NULL,
                      main = NULL, margin = NULL, ...)
## S3 method for class 'VarImp'
print(x, ranks = FALSE, ...)
## S3 method for class 'VarImp'
subset(x, subset, ...)
```

Arguments

x	an object of the class VarImp.
nVar	an integer specifying the number of predictor variables that should be included in the plot. The nVar predictor variables with the highest variable importance measure are retained.
type	a character string that indicates the type of plot. Must be one of the following: "bar", "box", "dot" or "rank" (see Details).
sort	a logical that specifies whether the predictors should be ranked according to the importance measures.
interval	a character string that indicates if, and which type of intervals should be added to the plot. Must be one of the following: "no", "quantile", or "sd" (see Details).
intervalProbs	a numerical vector of the form c(bottom, top), specifying the two quantiles that should be used for the interval. Only meaningful when interval = "quantile".
intervalColor	a color code or name, see par.
horizontal	a logical that specifies whether the plot should be horizontal (= importance values on the x-axis. The default is FALSE.
col	a color code or name, see par.
pch	Either a single character or an integer code specifying the plotting 'character', see par.
main	an overall title for the plot: see title.
margin	a numerical vector of the form c(bottom, left, top, right), which gives the number of lines of margin to be specified on the four sides of the plot. See par.
ranks	a logical specifying whether the (reversed) rankings should be printed instead of the importance values.
subset	a character, integer or logical vector, specifying the subset of predictor variables.
...	additional arguments.

Details

plot gives visualization of the variable importance values. print prints the importance values, or their (reversed) rankings if ranks = TRUE. ranks returns the reversed rankings of the variable importance values. The subset method for VarImp objects returns a VarImp object for only a subset of the original predictors in the random forest.

In plot, the type = "bar" results in a barplot, type = "dot" in a point-plot, type = "rank" in a point-plot with the importance rankings as the plotting 'characters', see ranks. In each of these three options an interval (based on either two quantiles or on the standard deviation of the perTree values) can be added to the plot. type = "box" results in boxplots, and is only meaningful when perTree values are available.

See Also

VarImp

Examples

```
## Fit a random forest (using cforest)
set.seed(290875)
readingSkills.cf <- party::cforest(score ~ ., data = party::readingSkills,
                           control = party::cforest_unbiased(mtry = 2, ntree = 50))

## compute permutation variable importance:
set.seed(290875)
permVIM <- permimp(readingSkills.cf)

## print the variable importance values
permVIM
print(permVIM, ranks = TRUE)
ranks(permVIM)

## Visualize the variable importance values
plot(permVIM, type = "bar", margin = c(6,3,3,1))
plot(permVIM, nVar = 2, type = "box", horizontal = TRUE)

## note the rankings
plot(subset(permVIM, c("age", "nativeSpeaker")), intervalColor = "pink")
plot(subset(permVIM, c("shoeSize", "nativeSpeaker")), intervalColor = "pink")
```

Index

∗ **package**
    permimp-package, 2
∗ **tree**
    permimp, 3
    selFreq, 8
    VarImp, 9
    VarImp-Methods, 10

as.VarImp *(*VarImp*)*, 9

cforest, *8*

data.frame, *10*

is.VarImp *(*VarImp*)*, 9

makeCluster, *4*
matrix, *10*
mclapply, *5*

par, *11*
parLapply, *5*
permimp, 3
permimp-package, 2
plot.VarImp *(*VarImp-Methods*)*, 10
print.VarImp *(*VarImp-Methods*)*, 10

rank, *7*
ranks, 7, *11*

selFreq, 8
subset.VarImp *(*VarImp-Methods*)*, 10

title, *11*

VarImp, *5*, *7–9*, 9, *10, 11*
varimp, *3–5*
VarImp-Methods, 10
VarImp-methods *(*VarImp-Methods*)*, 10