Package: Boruta (via r-universe)

September 1, 2024

Title Wrapper Algorithm for All Relevant Feature Selection

Version 8.0.0

Imports ranger

Suggests mlbench, rFerns, randomForest, testthat, xgboost, survival

Description An all relevant feature selection wrapper algorithm. It finds relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies (shadows).

BugReports https://gitlab.com/mbq/Boruta/-/issues

License GPL (>= 2)

URL https://gitlab.com/mbq/Boruta/

RoxygenNote 7.2.1

Encoding UTF-8

NeedsCompilation no

Author Miron Bartosz Kursa [aut, cre] (<https://orcid.org/0000-0001-7672-648X>), Witold Remigiusz Rudnicki [aut]

Maintainer Miron Bartosz Kursa <M.Kursa@icm.edu.pl>

Repository CRAN

Date/Publication 2022-11-12 08:30:16 UTC

Contents

attStats 2
Boruta 3
conditionalTransdapter 6
decohereTransdapter 7
getConfirmedFormula 7
getImpExtra 8
getImpFerns 9

getImpLegacyRf 9
getImpRf 10
getImpXgboost 11
getSelectedAttributes 12
imputeTransdapter 13
plot.Boruta 14
plotImpHistory 15
print.Boruta 16
srx 17
TentativeRoughFix 17

Index **19**

`attStats` *Extract attribute statistics*

Description

`attStats` shows a summary of a Boruta run in an attribute-centred way. It produces a data frame containing some importance stats as well as the number of hits that attribute scored and the decision it was given.

Usage

```
attStats(x)
```

Arguments

`x`	an object of a class Boruta, from which attribute stats should be extracted.

Value

A data frame containing, for each attribute that was originally in information system, mean, median, maximal and minimal importance, number of hits normalised to number of importance source runs performed and the decision copied from `finalDecision`.

Note

When using a Boruta object generated by a `TentativeRoughFix`, the resulting data frame will consist a rough-fixed decision.

`x` has to be made with `holdHistory` set to `TRUE` for this code to run.

Examples

```
## Not run:
library(mlbench); data(Sonar)
#Takes some time, so be patient
Boruta(Class~.,data=Sonar,doTrace=2)->Bor.son
print(Bor.son)
stats<-attStats(Bor.son)
print(stats)
plot(normHits~meanImp,col=stats$decision,data=stats)

## End(Not run)
```

Boruta *Feature selection with the Boruta algorithm*

Description

Boruta is an all relevant feature selection wrapper algorithm, capable of working with any classification method that output variable importance measure (VIM); by default, Boruta uses Random Forest. The method performs a top-down search for relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies, and progressively eliminating irrelevant features to stabilise that test.

Usage

```
Boruta(x, ...)

## Default S3 method:
Boruta(
  x,
  y,
  pValue = 0.01,
  mcAdj = TRUE,
  maxRuns = 100,
  doTrace = 0,
  holdHistory = TRUE,
  getImp = getImpRfZ,
  ...
)

## S3 method for class 'formula'
Boruta(formula, data, ...)
```

Arguments

x	data frame of predictors.
...	additional parameters passed to getImp.

y	response vector; factor for classification, numeric vector for regression, Surv object for survival (supports depends on importance adapter capabilities).
pValue	confidence level. Default value should be used.
mcAdj	if set to TRUE, a multiple comparisons adjustment using the Bonferroni method will be applied. Default value should be used; older (1.x and 2.x) versions of Boruta were effectively using FALSE.
maxRuns	maximal number of importance source runs. You may increase it to resolve attributes left Tentative.
doTrace	verbosity level. 0 means no tracing, 1 means reporting decision about each attribute as soon as it is justified, 2 means the same as 1, plus reporting each importance source run, 3 means the same as 2, plus reporting of hits assigned to yet undecided attributes.
holdHistory	if set to TRUE, the full history of importance is stored and returned as the ImpHistory element of the result. Can be used to decrease a memory footprint of Boruta in case this side data is not used, especially when the number of attributes is huge; yet it disables plotting of such made Boruta objects and the use of the TentativeRoughFix function.
getImp	function used to obtain attribute importance. The default is getImpRfZ, which runs random forest from the ranger package and gathers Z-scores of mean decrease accuracy measure. It should return a numeric vector of a size identical to the number of columns of its first argument, containing importance measure of respective attributes. Any order-preserving transformation of this measure will yield the same result. It is assumed that more important attributes get higher importance. +-Inf are accepted, NaNs and NAs are treated as 0s, with a warning.
formula	alternatively, formula describing model to be analysed.
data	in which to interpret formula.

Details

Boruta iteratively compares importances of attributes with importances of shadow attributes, created by shuffling original ones. Attributes that have significantly worst importance than shadow ones are being consecutively dropped. On the other hand, attributes that are significantly better than shadows are admitted to be Confirmed. Shadows are re-created in each iteration. Algorithm stops when only Confirmed attributes are left, or when it reaches maxRuns importance source runs. If the second scenario occurs, some attributes may be left without a decision. They are claimed Tentative. You may try to extend maxRuns or lower pValue to clarify them, but in some cases their importances do fluctuate too much for Boruta to converge. Instead, you can use TentativeRoughFix function, which will perform other, weaker test to make a final decision, or simply treat them as undecided in further analysis.

Value

An object of class Boruta, which is a list with the following components:

finalDecision	a factor of three value: Confirmed, Rejected or Tentative, containing final result of feature selection.

ImpHistory	a data frame of importances of attributes gathered in each importance source run. Beside predictors' importances, it contains maximal, mean and minimal importance of shadow attributes in each run. Rejected attributes get -Inf importance. Set to NULL if holdHistory was given FALSE.
timeTaken	time taken by the computation.
impSource	string describing the source of importance, equal to a comment attribute of the getImp argument.
call	the original call of the Boruta function.

References

Miron B. Kursa, Witold R. Rudnicki (2010). Feature Selection with the Boruta Package. *Journal of Statistical Software, 36(11)*, p. 1-13. URL: doi:10.18637/jss.v036.i11

Examples

```
set.seed(777)

#Boruta on the "small redundant XOR" problem; read ?srx for details
data(srx)
Boruta(Y~.,data=srx)->Boruta.srx

#Results summary
print(Boruta.srx)

#Result plot
plot(Boruta.srx)

#Attribute statistics
attStats(Boruta.srx)

#Using alternative importance source, rFerns
Boruta(Y~.,data=srx,getImp=getImpFerns)->Boruta.srx.ferns
print(Boruta.srx.ferns)

#Verbose
Boruta(Y~.,data=srx,doTrace=2)->Boruta.srx

## Not run:
#Boruta on the iris problem extended with artificial irrelevant features
#Generate said features
iris.extended<-data.frame(iris,apply(iris[,-5],2,sample))
names(iris.extended)[6:9]<-paste("Nonsense",1:4,sep="")
#Run Boruta on this data
Boruta(Species~.,data=iris.extended,doTrace=2)->Boruta.iris.extended
#Nonsense attributes should be rejected
print(Boruta.iris.extended)

## End(Not run)

## Not run:
```

```
#Boruta on the HouseVotes84 data from mlbench
library(mlbench); data(HouseVotes84)
na.omit(HouseVotes84)->hvo
#Takes some time, so be patient
Boruta(Class~.,data=hvo,doTrace=2)->Bor.hvo
print(Bor.hvo)
plot(Bor.hvo)
plotImpHistory(Bor.hvo)

## End(Not run)
## Not run:
#Boruta on the Ozone data from mlbench
library(mlbench); data(Ozone)
library(randomForest)
na.omit(Ozone)->ozo
Boruta(V4~.,data=ozo,doTrace=2)->Bor.ozo
cat('Random forest run on all attributes:\n')
print(randomForest(V4~.,data=ozo))
cat('Random forest run only on confirmed attributes:\n')
print(randomForest(ozo[,getSelectedAttributes(Bor.ozo)],ozo$V4))

## End(Not run)
## Not run:
#Boruta on the Sonar data from mlbench
library(mlbench); data(Sonar)
#Takes some time, so be patient
Boruta(Class~.,data=Sonar,doTrace=2)->Bor.son
print(Bor.son)
#Shows important bands
plot(Bor.son,sort=FALSE)

## End(Not run)
```

`conditionalTransdapter` *Conditional transdapter*

Description

Applies downstream importance source on a given object strata and averages their outputs.

Usage

```
conditionalTransdapter(groups, adapter = getImpRfZ)
```

Arguments

`groups`	groups.
`adapter`	importance adapter to transform.

Value

transformed importance adapter which can be fed into `getImp` argument of the `Boruta` function.

---

decohereTransdapter *Decohere transdapter*

Description

Applies the decoherence transformation to the input, destroying all multivariate interactions. It will trash the Boruta result, only apply if you know what are you doing! Works only for categorical decision.

Usage

```
decohereTransdapter(adapter = getImpRfZ)
```

Arguments

`adapter`	importance adapter to transform.

Value

transformed importance adapter which can be fed into `getImp` argument of the `Boruta` function.

Examples

```
set.seed(777)
# SRX data only contains multivariate interactions
data(srx)
# Decoherence transform removes them all,
# leaving no confirmed features
Boruta(Y~.,data=srx,getImp=decohereTransdapter())
```

---

getConfirmedFormula *Export Boruta result as a formula*

Description

Functions which convert the Boruta selection into a formula, so that it could be passed further to other functions.

Usage

```
getConfirmedFormula(x)

getNonRejectedFormula(x)
```

Arguments

x	an object of a class Boruta, made using a formula interface.

Value

Formula, corresponding to the Boruta results. `getConfirmedFormula` returns only Confirmed attributes, `getNonRejectedFormula` also adds Tentative ones.

Note

This operation is possible only when Boruta selection was invoked using a formula interface.

`getImpExtra` *ranger Extra-trees importance adapters*

Description

Those function is intended to be given to a `getImp` argument of `Boruta` function to be called by the Boruta algorithm as an importance source. `getImpExtraZ` generates default, normalized permutation importance, `getImpExtraRaw` raw permutation importance, finally `getImpExtraGini` generates Gini impurity importance.

Usage

```
getImpExtraZ(x, y, ntree = 500, num.trees = ntree, ...)

getImpExtraGini(x, y, ntree = 500, num.trees = ntree, ...)

getImpExtraRaw(x, y, ntree = 500, num.trees = ntree, ...)
```

Arguments

x	data frame of predictors including shadows.
y	response vector.
ntree	Number of trees in the forest; copied into `ranger`'s native num.trees, put to retain transparent compatibility with randomForest.
num.trees	Number of trees in the forest, as according to `ranger`'s nomenclature. If not given, set to `ntree` value. If both are given, `num.trees` takes precedence.
...	parameters passed to the underlying `ranger` call; they are relayed from `...` of `Boruta`. Note that these function work just by setting `splitrule` to `"extratrees"`.

getImpFerns *Random Ferns importance*

Description

This function is intended to be given to a `getImp` argument of `Boruta` function to be called by the Boruta algorithm as an importance source.

Usage

```
getImpFerns(x, y, ...)
```

Arguments

`x`	data frame of predictors including shadows.
`y`	response vector.
`...`	parameters passed to the underlying `rFerns` call; they are relayed from `...` of `Boruta`.

Note

Random Ferns importance calculation should be much faster than using Random Forest; however, one must first optimize the value of the `depth` parameter and it is quite likely that the number of ferns in the ensemble required for the importance to converge will be higher than the number of trees in case of Random Forest.

getImpLegacyRf *randomForest importance adapters*

Description

Those function is intended to be given to a `getImp` argument of `Boruta` function to be called by the Boruta algorithm as an importance source. `getImpLegacyRfZ` generates default, normalized permutation importance, `getImpLegacyRfRaw` raw permutation importance, finally `getImpLegacyRfGini` generates Gini index importance, all using `randomForest` as a Random Forest algorithm implementation.

Usage

```
getImpLegacyRfZ(x, y, ...)

getImpLegacyRfRaw(x, y, ...)

getImpLegacyRfGini(x, y, ...)
```

Arguments

x	data frame of predictors including shadows.
y	response vector.
...	parameters passed to the underlying randomForest call; they are relayed from ... of Boruta.

Note

The getImpLegacyRfZ function was a default importance source in Boruta versions prior to 5.0; since then ranger Random Forest implementation is used instead of randomForest, for speed, memory conservation and an ability to utilise multithreading. Both importance sources should generally lead to the same results, yet there are differences.

Most notably, ranger by default treats factor attributes as ordered (and works very slow if instructed otherwise with respect.unordered.factors=TRUE); on the other hand it lifts 32 levels limit specific to randomForest. To this end, Boruta decision for factor attributes may be different.

Random Forest methods has two main parameters, number of attributes tried at each split and the number of trees in the forest; first one is called mtry in both implementations, but the second ntree in randomForest and num.trees in ranger. To this end, to maintain compatibility, getImpRf* functions still accept ntree parameter relaying it into num.trees. Still, both parameters take the same defaults in both implementations (square root of the number all all attributes and 500 respectively).

Moreover, ranger brings some addition capabilities to Boruta, like analysis of survival problems or sticky variables which are always considered on splits.

Finally, the results for the same PRNG seed will be different.

Examples

```
set.seed(777)
#Add some nonsense attributes to iris dataset by shuffling original attributes
iris.extended<-data.frame(iris,apply(iris[,-5],2,sample))
names(iris.extended)[6:9]<-paste("Nonsense",1:4,sep="")
#Run Boruta on this data
Boruta(Species~.,getImp=getImpLegacyRfZ,
 data=iris.extended,doTrace=2)->Boruta.iris.extended
#Nonsense attributes should be rejected
print(Boruta.iris.extended)
```

---

getImpRf *ranger Random Forest importance adapters*

Description

Those function is intended to be given to a getImp argument of Boruta function to be called by the Boruta algorithm as an importance source. getImpRfZ generates default, normalized permutation importance, getImpRfRaw raw permutation importance, finally getImpRfGini generates Gini index importance.

Usage

```
getImpRfZ(x, y, ntree = 500, num.trees = ntree, ...)

getImpRfGini(x, y, ntree = 500, num.trees = ntree, ...)

getImpRfRaw(x, y, ntree = 500, num.trees = ntree, ...)
```

Arguments

x	data frame of predictors including shadows.
y	response vector.
ntree	Number of trees in the forest; copied into ranger's native num.trees, put to retain transparent compatibility with randomForest.
num.trees	Number of trees in the forest, as according to ranger's nomenclature. If not given, set to ntree value. If both are given, num.trees takes precedence.
...	parameters passed to the underlying ranger call; they are relayed from ... of Boruta.

Note

Prior to Boruta 5.0, getImpLegacyRfZ function was a default importance source in Boruta; see getImpLegacyRf for more details.

---

getImpXgboost *Xgboost importance*

Description

This function is intended to be given to a getImp argument of Boruta function to be called by the Boruta algorithm as an importance source. This functionality is inspired by the Python package BoostARoota by Chase DeHan. In practice, due to the eager way XgBoost works, this adapter changes Boruta into minimal optimal method, hence I strongly recommend against using this.

Usage

```
getImpXgboost(x, y, nrounds = 5, verbose = 0, ...)
```

Arguments

x	data frame of predictors including shadows.
y	response vector.
nrounds	Number of rounds; passed to the underlying xgboost call.
verbose	Verbosity level of xgboost; either 0 (silent) or 1 (progress reports). Passed to the underlying xgboost call.
...	other parameters passed to the underlying xgboost call. Similarly as nrounds and verbose, they are relayed from ... of Boruta. For convenience, this function sets nrounds to 5 and verbose to 0, but this can be overridden.

Note

Only dense matrix interface is supported; all predictions given to Boruta call have to be numeric (not integer). Categorical features should be split into indicator attributes.

References

https://github.com/chasedehan/BoostARoota

`getSelectedAttributes` *Extract names of the selected attributes*

Description

`getSelectedAttributes` returns a vector of names of attributes selected during a Boruta run.

Usage

```
getSelectedAttributes(x, withTentative = FALSE)
```

Arguments

`x`	an object of a class Boruta, from which relevant attributes names should be extracted.
`withTentative`	if set to `TRUE`, Tentative attributes will be also returned.

Value

A character vector with names of the relevant attributes.

Examples

```
## Not run:
data(iris)
#Takes some time, so be patient
Boruta(Species~.,data=iris,doTrace=2)->Bor.iris
print(Bor.iris)
print(getSelectedAttributes(Bor.iris))

## End(Not run)
```

`imputeTransdapter` *Impute transdapter*

Description

Wraps the importance adapter to accept NAs in input.

Usage

```
imputeTransdapter(adapter = getImpRfZ)
```

Arguments

`adapter`	importance adapter to transform.

Value

transformed importance adapter which can be fed into `getImp` argument of the `Boruta` function.

Note

An all-NA feature will be converted to all zeroes, which should be ok as a totally non-informative value with most methods, but it is not universally correct. Ideally, one should avoid having such features in input altogether.

Examples

```
## Not run:
set.seed(777)
data(srx)
srx_na<-srx
# Randomly punch 25 holes in the SRX data
holes<-25
holes<-cbind(
 sample(nrow(srx),holes,replace=TRUE),
 sample(ncol(srx),holes,replace=TRUE)
)
srx_na[holes]<-NA
# Use impute transdapter to mitigate them with internal imputation
Boruta(Y~.,data=srx_na,getImp=imputeTransdapter(getImpRfZ))

## End(Not run)
```

plot.Boruta *Plot Boruta object*

Description

Default plot method for Boruta objects, showing boxplots of attribute importances over run.

Usage

```
## S3 method for class 'Boruta'
plot(
  x,
  colCode = c("green", "yellow", "red", "blue"),
  sort = TRUE,
  whichShadow = c(TRUE, TRUE, TRUE),
  col = NULL,
  xlab = "Attributes",
  ylab = "Importance",
  ...
)
```

Arguments

x	an object of a class Boruta.
colCode	a vector containing colour codes for attribute decisions, respectively Confirmed, Tentative, Rejected and shadow.
sort	controls whether boxplots should be ordered, or left in original order.
whichShadow	a logical vector controlling which shadows should be drawn; switches respectively max shadow, mean shadow and min shadow.
col	standard col attribute. If given, suppresses effects of colCode.
xlab	X axis label that will be passed to boxplot.
ylab	Y axis label that will be passed to boxplot.
...	additional graphical parameter that will be passed to boxplot.

Value

Invisible copy of x.

Note

If col is given and sort is TRUE, the col will be permuted, so that its order corresponds to attribute order in ImpHistory.

This function will throw an error when x lacks importance history, i.e., was made with holdHistory set to FALSE.

Examples

```
## Not run:
library(mlbench); data(HouseVotes84)
na.omit(HouseVotes84)->hvo
#Takes some time, so be patient
Boruta(Class~.,data=hvo,doTrace=2)->Bor.hvo
print(Bor.hvo)
plot(Bor.hvo)

## End(Not run)
```

`plotImpHistory` *Plot Boruta object as importance history*

Description

Alternative plot method for Boruta objects, showing matplot of attribute importances over run.

Usage

```
plotImpHistory(
  x,
  colCode = c("green", "yellow", "red", "blue"),
  col = NULL,
  type = "l",
  lty = 1,
  pch = 0,
  xlab = "Classifier run",
  ylab = "Importance",
  ...
)
```

Arguments

`x`	an object of a class Boruta.
`colCode`	a vector containing colour codes for attribute decisions, respectively Confirmed, Tentative, Rejected and shadow.
`col`	standard `col` attribute, passed to `matplot`. If given, suppresses effects of `colCode`.
`type`	Plot type that will be passed to `matplot`.
`lty`	Line type that will be passed to `matplot`.
`pch`	Point mark type that will be passed to `matplot`.
`xlab`	X axis label that will be passed to `matplot`.
`ylab`	Y axis label that will be passed to `matplot`.
`...`	additional graphical parameter that will be passed to `matplot`.

Value

Invisible copy of x.

Note

This function will throw an error when x lacks importance history, i.e., was made with holdHistory set to FALSE.

Examples

```
## Not run:
library(mlbench); data(Sonar)
#Takes some time, so be patient
Boruta(Class~.,data=Sonar,doTrace=2)->Bor.son
print(Bor.son)
plotImpHistory(Bor.son)

## End(Not run)
```

---

print.Boruta *Print Boruta object*

Description

Print method for the Boruta objects.

Usage

```
## S3 method for class 'Boruta'
print(x, ...)
```

Arguments

x	an object of a class Boruta.
...	additional arguments passed to print.

Value

Invisible copy of x.

srx *Small redundant XOR data*

Description

A synthetic data set with 32 rows corresponding to all combinations of values of five logical features, A, B, N1, N2 and N3. The decision Y is equal to A xor B, hence N1–N3 are irrelevant attributes. The set also contains 3 additional features, A or B (AoB), A and B (AnB) and not A (nA), which provide a redundant, but still relevant way to reconstruct Y.

Usage

```
data(srx)
```

Format

A data frame with 8 predictors, 4 relevant: A, B, AoB, AnB and nA, as well as 3 irrelevant N1, N2 and N3, and decision attribute Y.

Details

This is set is an easy way to demonstrate the difference between all relevant feature selection methods, which should select all features except N1–N3, and minimal optimal ones, which will probably ignore most of them.

Source

https://blog.mbq.me/relevance-and-redundancy/

TentativeRoughFix *Rough fix of Tentative attributes*

Description

In some circumstances (too short Boruta run, unfortunate mixing of shadow attributes, tricky dataset...), Boruta can leave some attributes Tentative. `TentativeRoughFix` performs a simplified, weaker test for judging such attributes.

Usage

```
TentativeRoughFix(x, averageOver = Inf)
```

Arguments

`x`	an object of a class Boruta.
`averageOver`	Either number of last importance source runs to average over or Inf for averaging over the whole Boruta run.

Details

Function claims as Confirmed those attributes that have median importance higher than the median importance of maximal shadow attribute, and the rest as Rejected. Depending of the user choice, medians for the test are count over last round, all rounds or N last importance source runs.

Value

A Boruta class object with modified `finalDecision` element. Such object has few additional elements:

`originalDecision`
: Original `finalDecision`.

`averageOver`
: Copy of `averageOver` parameter.

Note

This function should be used only when strict decision is highly desired, because this test is much weaker than Boruta and can lower the confidence of the final result.

`x` has to be made with `holdHistory` set to `TRUE` for this code to run.

Index

∗ **datasets**
    srx, 17

attStats, 2

Boruta, 3, *7–13*
boxplot, *14*

conditionalTransdapter, 6

decohereTransdapter, 7

getConfirmedFormula, 7
getImpExtra, 8
getImpExtraGini *(*getImpExtra*)*, 8
getImpExtraRaw *(*getImpExtra*)*, 8
getImpExtraZ *(*getImpExtra*)*, 8
getImpFerns, 9
getImpLegacyRf, 9, *11*
getImpLegacyRfGini *(*getImpLegacyRf*)*, 9
getImpLegacyRfRaw *(*getImpLegacyRf*)*, 9
getImpLegacyRfZ *(*getImpLegacyRf*)*, 9
getImpRf, 10
getImpRfGini *(*getImpRf*)*, 10
getImpRfRaw *(*getImpRf*)*, 10
getImpRfZ *(*getImpRf*)*, 10
getImpXgboost, 11
getLegacyImpRfRaw *(*getImpLegacyRf*)*, 9
getNonRejectedFormula
        *(*getConfirmedFormula*)*, 7
getSelectedAttributes, 12

imputeTransdapter, 13

matplot, *15*

plot.Boruta, 14
plotImpHistory, 15
print, *16*
print.Boruta, 16

randomForest, *9, 10*
ranger, *8*, *10, 11*
rFerns, *9*

srx, 17

TentativeRoughFix, *2*, *4*, 17

xgboost, *11*