Package 'aVirtualTwins'

May 7, 2026

Type Package

Title Adaptation of Virtual Twins Method from Jared Foster

Version 1.0.1

Date 2018-02-03

Description Research of subgroups in random clinical trials with binary outcome and two treatments groups. This is an adaptation of the Jared Foster method (<https://www.ncbi.nlm.nih.gov/pubmed/21815180>).

License GPL-3 | file LICENSE

URL https://github.com/prise6/aVirtualTwins

BugReports https://github.com/prise6/aVirtualTwins/issues

Imports rpart, party, methods, randomForest, stats

Suggests caret, knitr, rpart.plot, rmarkdown, e1071

Depends R (>= 3.2.0),

Collate 'aVirtualTwins.R' 'data.R' 'object.R' 'difft.R' 'setClass.R' 'predict.R' 'forest.R' 'forest.double.R' 'forest.fold.R' 'forest.one.R' 'forest.wrapper.R' 'formatRCTDataset.R' 'incidences.R' 'object.wrapper.R' 'tools.R' 'tree.R' 'tree.class.R' 'tree.reg.R' 'tree.wrapper.R'

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Francois Vieille [aut, cre], Jared Foster [aut]

Maintainer Francois Vieille <vieille.francois@gmail.com>

Repository CRAN

Date/Publication 2018-02-04 16:00:40 UTC

Contents

`aVirtualTwins` *aVirtualTwins : An adapation of VirtualTwins method created by Jared Foster.*

Description

aVirtualTwins is written mainly with reference classes. Briefly, there is three kinds of class :

- `VT.object` class to represent RCT dataset used by aVirtualTwins. To format correctly RCT dataset, use `formatRCTDataset`.
- `VT.difft` class to compute difference between twins. Family `VT.forest` extends it to compute twins by random forest. `vt.forest` is users function.
- `VT.tree` class to find subgroups from `difft` by CART trees. `VT.tree.class` and `VT.tree.reg` extend it. `vt.tree` is users function.

Details

See http://github.com/prise6/aVirtualTwins for last updates.

formatRCTDataset *RCT format for Virtual Twins*

Description

formatRCTDataset returns dataset that Virtual Twins is able to analyze.

Usage

```
formatRCTDataset(dataset, outcome.field, treatment.field, interactions = TRUE)
```

Arguments

dataset	data.frame representing RCT's
outcome.field	name of the outcome's field in dataset
treatment.field	name of the treatment's field in dataset
interactions	logical. If running VirtualTwins with treatment's interactions, set to TRUE (default value)

Details

This function check these differents topic: Outcome must be binary and a factor. If numeric with two distincts values, outcome becomes a factor where the favorable reponse is the second level. Also, outcome is moved on the first column of dataset.

Treatment must have two distinct numeric values, 0 : no treatment, 1 : treatment. Treatment is moved to the second column.

Qualitatives variables must be factor. If it has more than two levels, if running VirtualTwins with interaction, it creates dummy variables.

Value

return data.frame with good format (explained in details section) to run VirtualTwins

Examples

```
## Not run:
    data.format <- formatRCTDataset(data, "outcome", "treatment", TRUE)

## End(Not run)
data(sepsis)
data.format <- formatRCTDataset(sepsis, "survival", "THERAPY", T)
```

`sepsis` *Clinical Trial for Sepsis desease*

Description

Simulated clinical trial with two groups treatment about sepsis desease. See details.

Usage

```
data(sepsis)
```

Format

470 patients and 13 variables.

survival binary outcome

THERAPY 1 for active treatment, 0 for control treatment

TIMFIRST Time from first sepsis-organ fail to start drug

AGE Patient age in years

BLLPLAT Baseline local platelets

blSOFA Sum of baselin sofa (cardiovascular, hematology, hepaticrenal, and respiration scores)

BLLCREAT Base creatinine

ORGANNUM Number of baseline organ failures

PRAPACHE Pre-infusion apache-ii score

BLGCS Base GLASGOW coma scale score

BLIL6 Baseline serum IL-6 concentration

BLADL Baseline activity of daily living score

BLLBILI Baseline local bilirubin

Details

This dataset is taken from SIDES method.

`Sepsis` contains simulated data on 470 subjects with a binary outcome survival, that stores survival status for patient after 28 days of treatment, value of 1 for subjects who died after 28 days and 0 otherwise. There are 11 covariates, listed below, all of which are numerical variables.

Note that contrary to the original dataset used in SIDES, missing values have been imputed by random forest (`randomForest::rfImpute()`). See file data-raw/sepsis.R for more details.

True subgroup is *PRAPACHE <= 26 & AGE <= 49.80. NOTE:* This subgroup is defined with the *lower* event rate (survival = 1) in treatement arm.

Source

http://biopharmnet.com/subgroup-analysis-software/

vt.data *Initialize virtual twins data*

Description

vt.data is a wrapper of formatRCTDataset and VT.object. Allows to format your data.frame in order to create a VT.object object.

Usage

```
vt.data(dataset, outcome.field, treatment.field, interactions = TRUE, ...)
```

Arguments

dataset	data.frame representing RCT's
outcome.field	name of the outcome's field in dataset
treatment.field	name of the treatment's field in dataset
interactions	logical. If running VirtualTwins with treatment's interactions, set to TRUE (default value)
...	parameters of VT.object

Value

VT.object

See Also

formatRCTDataset

Examples

```
data(sepsis)
vt.o <- vt.data(sepsis, "survival", "THERAPY", T)
```

`VT.difft` *Difference between twins*

Description

A reference class to represent difference between twin1 and twin2

Details

Difft are calculated depending on the favorable outcome chosen. It is the second level of the outcome. For example, if the outcome is 0 and 1, the favorable outcome is 1. Then,

$$difft_i = twin1_i - twin2_i if T_i = 1$$

$$difft_i = twin2_i - twin1_i if T_i = 0$$

. So *absolute* method is :

$$P(Y = 1|T = 1) - P(Y = 1|T = 0)$$

So *relative* method is :

$$P(Y = 1|T = 1)/P(Y = 1|T = 0)$$

So *absolute* method is :

$$logit(P(Y = 1|T = 1)) - logit(P(Y = 1|T = 0))$$

Fields

`vt.object` VT.object (refClass) representing data

`twin1` vector of $E(Y|T = realtreatment)$

`twin2` vector of $E(Y|T = anothertreatment)$

`method` Method available to compute difft : c("absolute", "relative", "logit"). Absolute is default value. See details.

`difft` vector of difference between twin1 and twin2

Methods

`computeDifft()` Compute difference between twin1 and twin2. See details.

See Also

`VT.forest`, `VT.forest.one`, `VT.forest.double`

VT.forest *Difft by Random Forest*

Description

An abstract reference class to compute twin via random forests

`VT.forest` extends `VT.difft`

Fields

`...` see fields of `VT.difft`

Methods

`checkModel(model)` Checking model class: Must be : train, RandomForest, randomForest

`getFullData()` Return twin1, twin2 and difft in column

`run()` Compute twin1 and twin2 estimation. Switch treatment if necessary.

See Also

`VT.difft`, `VT.forest.one`, `VT.forest.double`

vt.forest *Create forest to compute difft*

Description

`vt.forest` is a wrapper of `VT.forest.one`, `VT.forest.double` and `VT.forest.fold`. With parameter forest.type, any of these class can be used with its own parameter.

Usage

```
vt.forest(forest.type = "one", vt.data, interactions = T,
  method = "absolute", model = NULL, model_trt1 = NULL,
  model_trt0 = NULL, ratio = 1, fold = 10, ...)
```

Arguments

`forest.type`	must be a character. "one" to use VT.forest.one class. "double" to use VT.forest.double. "fold" to use VT.forest.fold.
`vt.data`	`VT.object`. Can be return of `vt.data()` function
`interactions`	logical. If running VirtualTwins with treatment's interactions, set to TRUE (default value)
`method`	character c("absolute", "relative", "logit"). See `VT.difft`.

model	allows to give a model you build outside this function. Can be randomForest, train or cforest. Is only used with forest.type = "one". If NULL, a randomForest model is grown inside the function. NULL is default.
model_trt1	see model_trt0 explanation and VT.forest.double details.
model_trt0	works the same as model parameter. Is only used with forest.type = "double". If NULL, a randomForest model is grown inside the function. NULL is default. See VT.forest.double for details.
ratio	numeric value that allow sampsize to be a bit controlled. Default to 1. See VT.forest.fold.
fold	number of fold you want to construct forest with k-fold method. Is only used with forest.type = "fold". Default to 5. See VT.forest.fold
...	randomForest() function parameters. Can be used for any forest.type.

Value

VT.difft

Examples

```
data(sepsis)
vt.o <- vt.data(sepsis, "survival", "THERAPY", T)
# inside model :
vt.f <- vt.forest("one", vt.o)
# ...
# your model :
# library(randomForest)
# rf <- randomForest(y = vt.o$getY(),
#                    x = vt.o$getX(int = T),
#                    mtry = 3,
#                    nodesize = 15)
# vt.f <- vt.forest("one", vt.o, model = rf)
# ...
# Can also use ... parameters
vt.f <- vt.forest("one", vt.o, mtry = 3, nodesize = 15)
# ...
```

VT.forest.double *Difft by double random forest*

Description

A reference class to compute twins via double random forests

Details

`VT.forest.double` extends `VT.forest`.

$E(Y|T=1)$ if $T_i = 1$ is estimated by OOB predictions from `model_trt1`. $E(Y|T=0)$ if $T_i = 0$ is estimated by OOB predictions from `model_trt0`. This is what `computeTwin1()` does.

Then $E(Y|T=1)$ if $T_i = 0$ is estimated by model_trt1. Then $E(Y|T=0)$ if $T_i = 1$ is estimated by model_trt1. This is what `computeTwin2()` does.

Fields

`model_trt1` a caret/RandomForest/randomForest object for treatment T = 1

`model_trt0` a caret/RandomForest/randomForest object for treatment T = 0

`...` field from parent class : `VT.forest`

Methods

`computeTwin1()` Compute twin1 with OOB predictions from double forests. See details.

`computeTwin2()` Compute twin2 by the other part of data in the other forest. See details.

See Also

`VT.difft`, `VT.forest`, `VT.forest.one`

`VT.forest.fold` *Difft via k random forests*

Description

A reference class to compute twins via k random forest

Details

`VT.forest.fold` extends `VT.forest`

Twins are estimated by k-fold cross validation. A forest is computed on k-1/k of the data and then used to estimate twin1 and twin2 on 1/k of the left data.

Fields

`interactions` logical set TRUE if model has been computed with interactions

`fold` numeric, number of fold, i.e. number of forest (k)

`ratio` numeric experimental, use to balance sampsize. Defaut to 1.

`groups` vector Define which observations belong to which group

`...` field from parent class : `VT.forest`

Methods

`run()` Compute twin1 and twin2 estimation. Switch treatment if necessary.

See Also

`VT.difft`, `VT.forest`, `VT.forest.one`, `VT.forest.double`

VT.forest.one *Difft by one random forest*

Description

A reference class to compute twins via one random forest

Details

`VT.forest.one` extends `VT.forest`.

OOB predictions are used to estimate $E(Y|T = realtreatment)$. Then, treatement is switched, it means that 1 becomes 0 and 0 becomes 1. We use again `model` to estimate $E(Y|T = theothertreatment)$. This is what `computeTwin1()` and `computeTwin2()` functions do.

Fields

`model` is a caret/RandomForest/randomForest class object

`interactions` logical set TRUE if model has been computed with interactions

`...` field from parent class : `VT.forest`

Methods

`computeTwin1()` Compute twin1 with OOB predictions

`computeTwin2()` Compute twin2 by switching treatment and applying random forest model

See Also

`VT.difft`, `VT.forest`, `VT.forest.double`

VT.object *VT.object*

Description

A Reference Class to deal with RCT dataset

Details

Currently working with binary response only. Continous will come, one day. Two-levels treatment only as well.

`data` field should be as described, however if virtual twins won't used interactions, there is no need to transform factors. See formatRCTDataset for more details.

Fields

`data` Data.frame with format: $Y, T, X_1, \ldots, X_p$. Y must be two levels factor if type is binary. T must be numeric or integer.

`screening` Logical, set to `FALSE` Set to `TRUE` to use `varimp` in trees computation.

`varimp` Character vector of important variables to use in trees computation.

`delta` Numeric representing the difference of incidence between treatments.

`type` Character : binary or continous. Only binary is currently available.

Methods

`computeDelta()` Compute delta value.

`getData(interactions = F)` Return dataset. If interactions is set to T, return data with treatement interactions

`getFormula()` Return formula : Y~T+X1+...+Xp. Usefull for cforest function.

`getIncidences(rule = NULL)` Return incidence table of data if rule set to NULL. Otherwise return incidence for the rule.

`getX(interactions = T, trt = NULL)` Return predictors (T,X,X*T,X*(1-T)). Or (T,X) if interactions is FALSE. If trt is not NULL, return predictors for T = trt

`getXwithInt()` Return predictors with interactions. Use VT.object::getX(interactions = T) instead.

`getY(trt = NULL)` Return outcome. If trt is not NULL, return outcome for T = trt.

`switchTreatment()` Switch treatment value.

See Also

`VT.difft`

Examples

```
## Not run:
# Default use :
vt.o <- VT.object$new(data = my.rct.dataset)

# Getting data
head(vt.o$data)

# or getting predictor with interactions
vt.o$getX(interactions = T)

# or getting X|T = 1
vt.o$getX(trt = 1)

# or getting Y|T = 0
vt.o$getY(0)

# Print incidences
vt.o$getIncidences()

## End(Not run)
```

`VT.predict` *VT.predict generic function*

Description

VT.predict generic function

Usage

```
VT.predict(rfor, newdata, type)

## S4 method for signature 'RandomForest,missing,character'
VT.predict(rfor, type = "binary")

## S4 method for signature 'RandomForest,data.frame,character'
VT.predict(rfor, newdata,
  type = "binary")

## S4 method for signature 'randomForest,missing,character'
VT.predict(rfor, type = "binary")

## S4 method for signature 'randomForest,data.frame,character'
VT.predict(rfor, newdata,
  type = "binary")
```

```
## S4 method for signature 'train,ANY,character'
VT.predict(rfor, newdata, type = "binary")

## S4 method for signature 'train,missing,character'
VT.predict(rfor, type = "binary")
```

Arguments

rfor	random forest model. Can be train, randomForest or RandomForest class.
newdata	Newdata to predict by the random forest model. If missing, OOB predictions are returned.
type	Must be binary or continous, depending on the outcome. Only binary is really available.

Value

vector $E(Y = 1)$

Methods (by class)

- rfor = RandomForest,newdata = missing,type = character: rfor(RandomForest) newdata (missing) type (character)
- rfor = RandomForest,newdata = data.frame,type = character: rfor(RandomForest) newdata (data.frame) type (character)
- rfor = randomForest,newdata = missing,type = character: rfor(randomForest) newdata (missing) type (character)
- rfor = randomForest,newdata = data.frame,type = character: rfor(randomForest) newdata (data.frame) type (character)
- rfor = train,newdata = ANY,type = character: rfor(train) newdata (ANY) type (character)
- rfor = train,newdata = missing,type = character: rfor(train) newdata (missing) type (character)

---

vt.subgroups *Visualize subgroups*

Description

Function which uses VT.tree intern functions. Package rpart.plot must be loaded. See VT.tree for details.

Usage

```
vt.subgroups(vt.trees, only.leaf = T, only.fav = T, tables = F,
  verbose = F, compete = F)
```

Arguments

vt.trees	VT.tree object. Or return of vt.tree function. Can be a list.
only.leaf	logical to select only leaf of trees. TRUE is default.
only.fav	logical select only favorable subgroups (meaning with favorable label of the tree). TRUE is default.
tables	set to TRUE if tables of incidence must be shown. FALSE is default.
verbose	print infos during computation. FALSE is default.
compete	print competitors rules thanks to competitors computation of the tree

Value

data.frame of rules

Examples

```
data(sepsis)
vt.o <- vt.data(sepsis, "survival", "THERAPY", TRUE)
# inside model :
vt.f <- vt.forest("one", vt.o)
# use classification tree
vt.tr <- vt.tree("class", vt.f, threshold = c(0.01, 0.05))
# show subgroups
subgroups <- vt.subgroups(vt.tr)
# change options you'll be surprised !
subgroups <- vt.subgroups(vt.tr, verbose = TRUE, tables = TRUE)
```

VT.tree *Tree to find subgroup*

Description

An abstract reference class to compute tree

Details

VT.tree.class and VT.tree.reg are children of VT.tree. VT.tree.class and VT.tree.reg try to find a strong association between difft (in VT.difft object) and RCT variables.

In VT.tree.reg, a regression tree is computed on difft values. Then, thanks to the threshold it flags leafs of the tree which are above the threshold (when sens is ">"). Or it flags leafs which are below the threshold (when sens = "<").

In VT.tree.class, it first flags difft above or below (depending on the sens) the given threshold. Then a classification tree is computed to find which variables explain flagged difft.

To sum up, VT.tree try to understand which variables are associated with a big change of difft.

Results are shown with `getRules()` function. `only.leaf` parameter allows to obtain only the leaf of the `tree`. `only.fav` parameter select only favorable nodes. `tables` shows incidence table of the rule. `verbose` allow `getRules()` to be quiet. And `compete` show also rules with `maxcompete` competitors from the `tree`.

Fields

`vt.difft` `VT.difft` object

`outcome` outcome vector from `rpart` function

`threshold` numeric Threshold for difft calculation (c)

`screening` Logical. TRUE if using varimp. Default is VT.object screening field

`sens` character Sens can be ">" (default) or "<". Meaning : `difft > threshold` or `difft < threshold`

`name` character Names of the tree

`tree` rpart Rpart object to construct the tree

`Ahat` vector Indicator of beglonging to Ahat

Methods

`computeNameOfTree(type)` return label of response variable of the tree

`createCompetitors()` Create competitors table

`getAhatIncidence()` Return Ahat incidence

`getAhatQuality()` Return Ahat quality

`getData()` Return data used for tree computation

`getIncidences(rule, rr.snd = T)` Return incidence of the rule

`getInfos()` Return infos about tree

`getRules(only.leaf = F, only.fav = F, tables = T, verbose = T, compete = F)` Return subgroups discovered by the tree. See details.

`run(...)` Compute tree with rpart parameters

See Also

`VT.tree.reg`, `VT.tree.class`

vt.tree *Trees to find Subgroups*

Description

vt.tree is a wrapper of VT.tree.class and VT.tree.reg. With parameter tree.type, any of these two class can be used with its own parameter.

Usage

```
vt.tree(tree.type = "class", vt.difft, sens = ">", threshold = seq(0.5,
  0.8, 0.1), screening = NULL, ...)
```

Arguments

tree.type	must be a character. "class" for classification tree, "reg" for regression tree.
vt.difft	VT.difft object. Or return of vt.forest function.
sens	must be a character c(">","<"). See VT.tree for details.
threshold	must be numeric. It can be a unique value or a vector. If numeric vector, a list is returned. See VT.tree for details.
screening	must be logical. If TRUE, only varimp variables of VT.object is used to create the tree.
...	rpart() function parameters. Can be used for any tree.type.

Details

See VT.tree, VT.tree.class and VT.tree.reg classes.

Value

VT.tree or a list of VT.tree depending on threshold dimension. See examples.

Examples

```
data(sepsis)
vt.o <- vt.data(sepsis, "survival", "THERAPY", T)
# inside model :
vt.f <- vt.forest("one", vt.o)
# use classification tree
vt.tr <- vt.tree("class", vt.f, threshold = c(0.01, 0.05))
# return a list
class(vt.tr)
# access one of the tree
tree1 <- vt.tr$tree1
# return infos
# vt.tr$tree1$getInfos()
# vt.tr$tree1$getRules()
```

```
# use vt.subgroups tool:
subgroups <- vt.subgroups(vt.tr)
```

VT.tree.class *Classification tree to find subgroups*

Description

See VT.tree

Methods

run(...) Compute tree with rpart parameters

VT.tree.reg *Regression tree to find subgroups*

Description

See VT.tree

Methods

run(...) Compute tree with rpart parameters

Index