Package ‘forestRK’

July 19, 2019

Version 0.0-5

Encoding UTF-8

Title Implements the Forest-R.K. Algorithm for Classification Problems

Description Provides functions that calculates common types of splitting criteria used in random forests for classification problems, as well as functions that make predictions based on a single tree or a Forest-R.K. model; the package also provides functions to generate importance plot for a Forest-R.K. model, as well as the 2D multidimensional-scaling plot of data points that are colour coded by their predicted class types by the Forest-R.K. model. This package is based on:
Bernard, S., Heutte, L., Adam, S., (2008, ISBN:978-3-540-85983-3)
``Forest-R.K.: A New Random Forest Induction Method",
Fourth International Conference on Intelligent Computing,
September 2008, Shanghai, China, pp.430-437.

Author Hyunjin Cho [aut, cre],
Rebecca Su [ctb]

Maintainer Hyunjin Cho `<h56cho@uwaterloo.ca>`

Depends R (>= 3.6.0)

Imports igraph, ggplot2, rapportools, partykit, stats, graphics, pkgKitten, knitr, mlbench

License GPL (>= 3) | file LICENSE

Repository CRAN

Note The package is also based on the discussion
https://stats.stackexchange.com/questions/168964/building-a-regression-tree-with-r-from-scratch/168967#168967

RoxygenNote 6.1.1

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation no

Date/Publication 2019-07-19 10:50:02 UTC

R topics documented:

bstrap *Performs bootstrap sampling of the (training) dataset*

Description

Performs bootstrap sampling of our (training) dataset; this function is used inside of the `forestRK` function.

Usage

```
bstrap(dat = data.frame(), nbags, samp.size)
```

Arguments

`dat`	a numericized data frame that stores both the covariates of the observations and their numericized class types `y`; `dat` should contain no `NA` or `NaN`'s.
`nbags`	the number of bags or the number of bootstrap samples that we want to generate.
`samp.size`	the number of samples that each bag (individual bootstrap sample) should contain.

Value

A list containing a data frames of bootstrap samples generated from `dat`.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`forestRK`

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

# covariates of training data set
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new
# combine the covariates x with class types y
b <- data.frame(cbind(x.train, y.train))

## bstrp function example
bootstrap.sample <- bstrap(dat = b, nbags = 20, samp.size = 30)
```

`construct.treeRK` *Constructs a classification tree on the (training) dataset, by implementing the RK (Random 'K') algorithm*

Description

Constructs a classification tree based on the dataset of interest by implementing the RK (Random 'K') algorithm.

The package `rapportools` is loaded internally when this function is called; this is to use the method `is.boolean` to check one of the stopping criteria in the beginning of the function. The functions specifically from the `forestRK` package that are being used inside `construct.treeRK` are `criteria.calculator` and `cutoff.node.and.covariate.index.finder`.

The `construct.treeRK` output is one of the arguments that is used to call the `pred.treeRK` function.

DESCRIPTIONS OF THE RETURNED VALUES:

The hirarchical flag of a rktree (`construct.treeRK()$flag`) is constructed in the following way:

(1) the first entry of the flag, "r" denotes for "root"; (2) the subsequent strings of the flag is constructed in the way that last "x" denotes for the left child node of the node represented by the series of characters that are before the last "x", and the last "y" denotes for the right child node of the node represented by the series of characters that are before the last "y".

For example, the flag "rxyx" is the left child node of the node represented by "rxy".

`x.node.list` and `y.node.list` are the lists of children nodes (for `x` and `y`, respectively) of the `rktree`, listed in the order consistent to the order of the nodes represented in the `rktree`'s hirarchical flag.

`covariate.split` is a matrix that lists the numericized covariate names that were used for the splits to construct the rktree. The first entry of `covariate.split` is `NA`, which stands for the condition at the root. The number immediately underneath NA is the numericized covariate name that was used for the first split in the rktree, and the number below that is the numericized covariate name that was used for the second split, etc. If the numericized covariate name listed under covariate.split is the number "n", this corresponds to the "n"-th covariate or the name of the "n"-th column of the data frame x.train.

value.at.split is a vector that lists the actual values of the covariates at which the split had occured while constructing the rktree. The first entry of value.at.split is NA, which denotes for the root prior to any splits. To give an example of how to interpret the value.at.split, if the second entry appear in the covariate.split is 4, and the second entry appear under value.at.split is 0.5, this indicates that the first split of the rktree had occured on the covariate corresponds to the 4th column of the data frame x.train, and the exact criteria for that first split was (4th covariate value) <= 0.5 vs. (4th covariate value) > 0.5.

amount.decrease.criteria is a matrix that lists the amount of decrease in splitting criteria (Entropy or Gini Index) after each split had occurred. The first entry of amount.decrease.criteria is NA, which denotes for the condition at the root (no split). To give an example, if the second entry appear in the amount.decrease.criteria is 0.91, and if entropy was set to TRUE, this means that after the first split, the Entropy of the original node had decreased by 0.91.

num.obs is a matrix that stores the number of observations contained within a parent node prior to the split; the matrix starts with the entry "NA", in order to reflect the condition at "root". The 2nd entry of num.obs would inform us on the number of observations contained within the parent node on which the 1st split had took place while the rktree was built; the 3rd entry of the num.obs would inform us on the number of observations contained within the parent node on which the 2nd split had took place, and so on.

Usage

```
construct.treeRK(x.train = data.frame(), y.new.train = c(),
                 min.num.obs.end.node.tree = 5, entropy = TRUE)
```

Arguments

x.train
: a numericized data frame of covariates of the data on which we want to build our rktree models (typically the training data); this data frame can be obtained by applying the x.organizer function. x.train should contain no NA or NaN's.

y.new.train
: a numericized class types of the observations from the dataset on which we want to build our rktree models (typically the training data). y.new.train should contain no NA or NaN's.

min.num.obs.end.node.tree
: the minimum number of observations that we want each end node of our rktree to contain. Default is set to '5'.

entropy
: TRUE if Entropy is used as the splitting criteria; FALSE if Gini Index is used as the splitting criteria. Default is set to TRUE.

Value

A list containing the following items:

`covariate.names`
a vector of the names of all covariates that we consider in our model.

`l` length of the hierarchical flag.

`x.node.list` a list containing a series of children nodes produced from the numericized data frame `x.train` as the `rktree` model was building up.

`y.new.node.list`
a list containing a series of children nodes produced from the numericized vector of class type `y.new.train` as the `rktree` model was building up.

`flag` hierchical flag that characterizes each split in the `rktree`.

`covariate.split`
a matrix that lists numericized covariates used for each split as the `rktree` was built.

`value.at.split`
a vector that lists the values at which each node of the `rktree` was split.

`amt.decrease.criteria`
a matrix that lists the amount of decrease in splitting criteria after each split as the `rktree` was built.

`num.obs` a matrix that stores the number of observations contained in each parent node right before each split.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`pred.treeRK`

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

## numericize the data
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

# Construct a tree
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
tree.entropy <- construct.treeRK(x.train, y.train)
tree.gini <- construct.treeRK(x.train, y.train,
                              min.num.obs.end.node.tree = 6, entropy = FALSE)
tree.entropy$covariate.names
tree.gini$flag # ...etc...
```

criteria.after.split.calculator

Calculates Entropy or Gini Index of a node after a given split

Description

Calculates Entropy or Gini Index of a particular node after a particular split; this function is called within `construct.treeRK` function.

The argument `split.record` is a `kidids_split` object from the package `partykit`; the method `kidids_split` splits the data according to the criteria specified by an user ahead of time, and returns a vector storing the index of the split group (group "1" or "2") that each observation from the original data in question belongs to after the split has occurred.

For more information about the function, please see the `partykit` documentation.

Usage

```
criteria.after.split.calculator(x.node = data.frame(), y.new.node = c(),
                                split.record = kidids_split(),
                                entropy = TRUE)
```

Arguments

`x.node`	numericized data frame of covariates (obtained via `x.organizer()`) from a particular node that is to be split; `x.node` should contain no `NA` or `NaN`'s.
`y.new.node`	numericized class type of each observation from a particular node that is to be split; `y.new.node` should contain no `NA` or`NaN`'s.
`split.record`	output of the `kidids_split` function from the `partykit` package that describes a particular split.
`entropy`	`TRUE` if Entropy is used as the splitting criteria; `FALSE` if Gini Index is used instead. Default is set to `TRUE`.

Value

The value of Entropy or Gini Index of a particular node after a particular split.

Author(s)

Hyunjin Cho, <h56cho@uwaterloo.ca> Rebecca Su, <y57su@uwaterloo.ca>

See Also

`criteria.calculator`

Examples

```
## example: iris dataset
library(forestRK) # load the package forestRK
library(partykit)

# covariates of training data set
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
# numericized class types of observations of training dataset
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new
## criteria.after.split.calculator() example in the implementation
## of the forestRK algorithm

ent.status <- TRUE

# number.of.columns.of.x.node
# = total number of covariates that we consider
number.of.columns.of.x.node <- dim(x.train)[2]
# m.try = the randomly chosen number of covariates that we consider
# at the time of split
m.try <- sample(1:(number.of.columns.of.x.node),1)
## sample m.try number of covariates from the list of all covariates
K <- sample(1:(number.of.columns.of.x.node), m.try)

# split the data
# (the choice of the type of split used here is only arbitrary)
# for more information about kidids_split,
# please refer to the documentation for the package 'partykit'
sp <- partysplit(varid=K[1], breaks = x.train[1,K[1]], index = NULL,
                 right = TRUE, prob = NULL, info = NULL)
split.record <- kidids_split(sp, data=x.train)

# implement critera.after.split function based on kidids_split object
criteria.after.split <- criteria.after.split.calculator(x.train,
                                   y.train, split.record, ent.status)
criteria.after.split
```

---

criteria.calculator

Calculates Entropy or Gini Index of a particular node before (or without) a split

---

Description

Calculates the Entropy or Gini Index of a particular node before (or without) a split. This function is used inside the `criteria.after.split.calculator` method.

Usage

```
criteria.calculator(x.node = data.frame(), y.new.node = c(),
                    entropy = TRUE)
```

Arguments

`x.node` numericized data frame of covariates of a particular node (can be obtained by applying `x.organizer`) before or without a split; `x.node` should contain no `NA` or `NaN`'s.

`y.new.node` numericized vector of class type (`y`) of a particular node (can be obtained by applying `y.organizer`) before or without split; `y.new.node` should contain no `NA` or `NaN`'s.

`entropy` `TRUE` if Entropy is used as the splitting criteria; `FALSE` if Gini Index is used as the splitting criteria. Default is set to `TRUE`.

Value

A list containing the following items:

`criteria` the value of the Entropy or the Gini Index of a particular node.

`ent.status` logical value (`TRUE` or `FALSE`) of the parameter `entropy`.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`criteria.after.split.calculator`

Examples

```
## example: iris dataset
library(forestRK) # load the package forestRK

# covariates of training data set
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
# numericized class types of observations of training dataset
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

## criteria.calculator() example
## calculate the Entropy of the original training dataset
criteria.calculator(x.node = x.train, y.new.node = y.train)
## calculate the Gini Index of the original training dataset
criteria.calculator(x.node = x.train, y.new.node = y.train, entropy = FALSE)
```

`cutoff.node.and.covariate.index.finder` *Identifies optimal cutoff point of an impure node for splitting after applying the `rk` (Random K) algorithm.*

Description

Identifies optimal cutoff point of an impure dataset for splitting after applying the `rk` (Random K) algoritm, in terms of Entropy or Gini Index.

To give an example, if the function gives `cutoff.value` of 2.5, `covariate.ind` of 4, and `cutoff.node` of 23, this would inform the user that if a split is to be performed on the particular node that the user is considering, the split should occur on the 4th covariate (the actual name of this covariate would be the name of the 4th column from the original dataset), at the value of 2.5 (left child node in this case would be the group of observations that have their 4th covariate value less than or equal to 2.5, and for the right child node would be the group of observations that have their 4th covariate value greater than 2.5), and that this splitting point corresponds to the 23rd observation point of the node.

This function internally loads the packages `partykit` and `rapportools`; the package `partykit` is internally loaded to generate the object `split.record.optimal`, and the package `rapportools` is loaded to allow the validation of one of the stopping criteria that uses `is.boolean` method.

This function is ran internally in the `construct.treeRK` function.

Usage

```
cutoff.node.and.covariate.index.finder(x.node = data.frame(),
                                        y.new.node = c(), entropy = TRUE)
```

Arguments

`x.node` a numericized data frame of covariates of the observations from a particular node prior to the split (can be obtained after applying `x.organizer()`); `x.node` should contain no `NA` or `NaN`'s.

`y.new.node` a vector storing numericized class type of the observations from a particular node before the split (can be obtained after applying `y.organizer()`); `y.new.node` should contain no `NA` or `NaN`'s.

`entropy` `TRUE` if Entropy is used as the splitting criteria; `FALSE` if Gini Index is used as the splitting criteria. Default is set to `TRUE`.

Value

A list containing the following items:

`cutoff.value` the value at which the optimal split should take place.

`cutoff.node` the index of the observation (observation number) at which optimal split should occur.

`covariate.ind`
: numeric index of the covariate at which the optimal split should occur.

`split.record.optimal`
: the `kidid_split` output of the optimal split.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`construct.treeRK`

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

## numericize the data
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

# implementation of cutoff.node.and.covariate.index.finder()
res <- cutoff.node.and.covariate.index.finder(x.train, y.train,
                                              entropy=FALSE)
res$cutoff.value
res$cutoff.node
res$covariate.ind
res$split.record.optimal
```

`draw.treeRK` *Creates a* `igraph` *plot of a* `rktree`

Description

Creates a plot of a `rktree` that was built from the (training) dataset.

The package `igraph` is loaded internally when this function is called, to aid in generating the plot of a `rktree`.

DESCRIPTIONS OF THE `rktree` PLOT:

The resulting plot is a classical decision tree.

The rectangular nodes (or vertices) that contain "=<" symbol are used to describe the splitting criteria applied to that very node while constructing the `rktree`; for example, in the `rktree` plot generated by the code shown in the "examples" section below, the node with the label "Petal.Width =< 1.6" indicates that this node was split into a chunk that contains observations with Petal.Width <= 1.6 and a chunk that contains observations with Petal.Width greater than 1.6, in order to construct the `rktree`.

Any other rectangular nodes (or vertices) that do not contain the "=<" symbol indicate that we have reached an end node, and the text displayed in such node is the actual name of the class type that the rktree model assigns to the observations belonging to that node; for example, in the `rktree` plot generated by the code shown in the "examples" section below, the vertex with the label "setosa" indicates that the `rktree` assigns the class type "setosa" to all observations that belong to that particular node.

Usage

```
draw.treeRK(tr = construct.treeRK(), y.factor.levels,
font = "Times", node.colour = "white", text.colour = "dark blue",
text.size = 0.67, tree.vertex.size = 75, tree.title = "Diagram of a Tree",
title.colour = "dark blue")
```

Arguments

`tr` a `construct.treeRK()` object (a tree).

`y.factor.levels` a `y.organizer()$y.factor.levels` output.

`font` font type used in the `rktree` plot; default is "Times".

`node.colour` colour of the node used in the `rktree` plot; default is "White".

`text.colour` colour of the text used in the `rktree` plot; default is "Dark Blue".

`text.size` size of the text in the `rktree` plot; default is 0.67.

`tree.vertex.size` size of the `rktree` plot vertices; default is 75.

`tree.title` title of the `rktree` plot; default title is "Diagram of a Tree".

`title.colour` colour for the title of the `rktree` plot; default title colour is "Dark Blue".

Value

An `igraph` plot of a `rktree`.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`mds.plot.forestRK` `importance.plot.forestRK`

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

## Numericize the data
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
```

```
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new
y.factor.levels <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.factor.levels

## Construct a tree
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
tree.entropy <- construct.treeRK(x.train, y.train)

# Plot the tree
draw.treeRK(tree.entropy, y.factor.levels, font="Times",
            node.colour = "black", text.colour = "white", text.size = 0.7,
            tree.vertex.size = 100, tree.title = "Decision Tree",
            title.colour = "dark green")
```

ends.index.finder *Identifies numerical indices of the end nodes of a* rktree *from the matrix of hierarchical flags.*

Description

Identifies numerical indices of the end nodes of a rktree by closely examining the structure of the rktree flag (obtained via construct.treeRK()$flag); the precise algorithm used is the following:

if m-th string in the list of rktree flag is a substring of one or more of (m + 1),...,n-th strings in the list of flag, then the node represented by the m-th string of the flag is not an end node; otherwise, the node represented by the m-th string of the flag is the end node.

Usage

```
ends.index.finder(tr.flag = matrix())
```

Arguments

tr.flag	a construct.treeRK()$flag object or a similar flag matrix.

Value

A vector that lists the indices of the end nodes of a given rktree (indices that are consistent to the indices in x.node.list, y.new.node.list, and flag that are returned by the construct.treeRK function).

Author(s)

Hyunjin Cho, <h56cho@uwaterloo.ca> Rebecca Su, <y57su@uwaterloo.ca>

See Also

construct.treeRK

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

# covariates of training data set
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

# Construct a tree
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
tree.entropy <- construct.treeRK(x.train, y.train)

# Find indices of end nodes of tree.entropy
end.node.index <- ends.index.finder(tree.entropy$flag)
```

`forestRK` *Builds up a random forest RK model based on the given (training) dataset*

Description

Builds up a random forest RK model onto the given (training) dataset.

The functions `bstrap` and `construct.treeRK` are used inside this function. Once the call for `bstrap` generates bootstrap samples of the training dataset, then the function `construct.treeRK` is called in order to build a tree on each of those bootstrap dataset, to form a bigger forest.

Calling of this function internally loads the package `rapportools`; this is to allow the use of `is.boolean` method to check one of the stopping criteria.

Usage

```
forestRK(X = data.frame(), Y.new = c(),
         min.num.obs.end.node.tree = 5, nbags, samp.size, entropy = TRUE)
```

Arguments

`X` a numericized data frame storing covariates of each observation contained in the given (training) dataset (obtained via `x.organizer()`); `X` should contain no `NA` or `NaN`'s.

`Y.new` a vector storing the numericized class types of each observation contained in the given (training) dataset `X`; `Y.new` should contain no `NA` or `NaN`'s.

`min.num.obs.end.node.tree`
the minimum number of observations that we want each end node of our `rktree` to contain. Default is set to 5.

`nbags` number of bootstrap samples that we want to generate to generate a forest.

`samp.size` number of observations that we want each of our bootstrap samples to contain.

`entropy` `TRUE` if we use Entropy as the splitting criteria; `FALSE` if we use the Gini Index for the splitting criteria. Default is set to `TRUE`.

Value

A list containing the following items:

`X` The original (training) dataset that was used to construct the random forest RK model.

`forest.rk.tree.list` A list of trees (`construct.treeRK` objects) contained in the `forestRK` model.

`bootsamp.list` A list containing data frames of bootstrap samples that were generated from the given (training) dataset `X`.

`ent.status` The value of the parameter `entropy`.

Author(s)

Hyunjin Cho, <h56cho@uwaterloo.ca> Rebecca Su, <y57su@uwaterloo.ca>

See Also

`bstrap` `construct.treeRK`

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

# covariates of training data set
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

# Implement forestRK function
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
# normally nbags and samp.size has to be much larger than 30 and 50
forestRK.1 <- forestRK(x.train, y.train, nbags = 30, samp.size = 50)

# extract the first tree in the forestRK.1 model
forestRK.1$forest.rk.tree.list[[1]]
```

get.tree.forestRK *Extracts the structure of one or more trees in a forestRK object*

Description

Extracts structure of one or more trees from a forestRK object.

Each tree in the list are named by the exact indices of the tree; for example, if the code obj <-get.tree.forestRK(forestRK.1,tree.index=c(4,5,6)) was used to extract the structure of the 4th, 5th, and 6th trees in the forest, the user can retrieve the information pertains explicitly to the 4th tree in the forest by doing obj["4"]].

Usage

```
get.tree.forestRK(forestRK.object = forestRK(), tree.index=c())
```

Arguments

forestRK.object
a forestRK object.

tree.index
a vector of indices of the trees that we want to extract from the forestRK object.

Value

A list containing forestRK trees that have their indices specified in the function argument tree.index.

Author(s)

Hyunjin Cho, <h56cho@uwaterloo.ca> Rebecca Su, <y57su@uwaterloo.ca>

See Also

forestRK

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

# random forest
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
# normally nbags and samp.size have to be much larger than 30 and 50
forestRK.1 <- forestRK(x.train, y.train, nbags = 30, samp.size = 50)
```

```
# get tree
tree.index.ex <- c(1,3,8)
get.tree <- get.tree.forestRK(forestRK.1, tree.index = tree.index.ex)
get.tree[["8"]] # display the 8th tree of the random forest
```

`importance.forestRK` *Calculates Gini Importance or Mean Decrease Impurity (same algorithm is used in 'scikit-learn') of each covariate that we consider in the `forestRK` model*

Description

Calculates Gini Importance of each covariate considered in the forestRK model, and list them in the order of most important to the least important.

The Gini Importance or Mean Decrease in Impurity algorithm is also used in 'scikit-learn'. Gini Importance is defined as the total decrease in node impurity averaged over all trees of the ensemble, where the decrease in node impurity is obtained after weighting by the probability for an observation to reach that node (which is approximated by the proportion of samples reaching that node).

Usage

```
importance.forestRK(forestRK.object = forestRK())
```

Arguments

`forestRK.object`
a `forestRK` object.

Value

A list containing the following items:

`importance.covariate.names`
a vector of names of the covariates of our dataset ordered from the most important to the least important.

`average.decrease.in.criteria.vec`
a vector storing the average decrease in the weighted splitting criteria by each covariate that was calculated across all trees in the forestRK object; the numbers are ordered from the highest average decrease in criteria (importance) to the lowest, so the i-th importance number from this vector pertains to the i-th covariate listed in the vector output `importance.covariate.names`.

`ent.status` status of the parameter `entropy`; `TRUE` if Entropy is used for splitting critera, `FALSE` if Gini Index is used instead.

`x.original` a numericized data frame storing covariates of each observation from the given (training) dataset that was used to construct the `forestRK` object in the beginning of the `forestRK` function call.

Author(s)

Hyunjin Cho, <h56cho@uwaterloo.ca> Rebecca Su, <y57su@uwaterloo.ca>

See Also

forestRK

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

## numericize the data
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

# random forest
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
# typically the nbags and samp.size has to be much larger than 30 and 50
forestRK.1 <- forestRK(x.train, y.train, nbags=30, samp.size=50)
# execute importance.forestRK function
imp <- importance.forestRK(forestRK.1)
```

---

importance.plot.forestRK
Generates importance ggplot *of the covariates considered in the* forestRK *model*

Description

Generates importance ggplot of the covariates considered in the forestRK model.

When the number of covariates under consideration is huge, it can be pretty difficult to read the covariate name from this plot. In this case, user can identify the name of the covariate that he or she is interested in by extracting importance.covariate.names from the importance.forestRK.object that was used in the function call. importance.covariate.names lists the original names of the covariates after ordering them from the most important to the least important. So for example, the exact name of the covariate that has the second highest importance would be the second element of the vector importance.covariate.names, and so on.

Usage

```
importance.plot.forestRK(importance.forestRK.object = importance.forestRK(),
                         colour.used = "dark green", fill.colour = "dark green",
                         label.size = 10)
```

Arguments

`importance.forestRK.object`
an `importance.forestRK` object.

`colour.used` colour used for the border of the importance plot; default is "dark green".

`fill.colour` colour used to fill the bars of the importance plot; default is "dark green" (yes, I like dark green).

`label.size` size of the labels; default is set to 10.

Value

An importance plot of the covariates considered in the `forestRK` model, ordered from the most important covariate to the least important covariate.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`forestRK`

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

## numericize the data
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

# random forest
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
# typically the nbags and samp.size has to be much larger than 30 and 50
forestRK.1 <- forestRK(x.train, y.train, nbags = 30, samp.size = 50)
# execute forestRK.importance function
imp <- importance.forestRK(forestRK.1)

# generate importance plot
importance.plot.forestRK(imp)
```

mds.plot.forestRK *Makes 2D MDS (multidimensional scaling) `ggplot` of the test observations based on the predictions from a `forestRK` model.*

Description

Plots 2D MDS (Multi-Dimensional Scaling) `ggplot` of the test observations based on the provided `forestRK` model, and each test observation is colour coded by their predicted class types.

The plot also has legends that tells user which colour pertains to which predicted class type.

The existing R functions `dist` and `cmdscale` were used in this function to compute the Multi-Dimensional Scales of the test data.

Usage

```
mds.plot.forestRK(pred.forestRK.object = pred.forestRK(),
 plot.title ="MDS Plot of Test Data Colour Coded by Forest RK Model Predictions",
 xlab ="First Coordinate", ylab = "Second Coordinate",
 colour.lab = "Predictions By The Random Forest RK Model")
```

Arguments

`pred.forestRK.object`
a `pred.forestRK()` object.

`plot.title` an user specified title for the mds plot; the default is "MDS Plot of Test Data Colour Coded by Forest RK Model Predictions".

`xlab` label for the x-axis of the plot; the default is "First Coordinate".

`ylab` label for the y-axis of the plot; the default is "Second Coordinate".

`colour.lab` label title for the legend that specifies categories for each colour; the default is "Predictions By The Random Forest RK Model".

Value

A multidimensional scaling ggplot (2D) of the test observations, colour coded by their predicted class types.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`forestRK`

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
x.test <- x.organizer(iris[,1:4], encoding = "num")[c(26:50,76:100,126:150),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new
y.factor.levels <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.factor.levels

# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
# typically the nbags and samp.size has to be much larger than 30 and 50
pred.forest.rk <- pred.forestRK(x.test = x.test,
                                x.training = x.train, y.training = y.train,
                                nbags = 30, samp.size = 50,
                                y.factor.levels = y.factor.levels)

# generate a classical mds plot of test observations
# and colour code them by the predicted class
mds.plot.forestRK(pred.forest.rk)
```

`pred.forestRK` *Make predictions on the test data based on the forestRK model constructed from the training data*

Description

Makes predictions on the test dataset based on the `forestRK` model constructed from the training dataset.

Please be aware that, the test data points in `test.prediction.df.list`,`pred.for.obs.forest.rk`, and `num.pred.for.obs.forest.rk` are re-ordered by the increasing original index number (the original rownames) of those test observations. So if you shuffled the data before seperating them into a training and a test set, the order of the data points in which they are presented under the attribute `test.prediction.df.list`,`pred.for.obs.forest.rk`, and `num.pred.for.obs.forest.rk` may not be same as the shuffled order of your original test set.

Calling of this function internally loads the package `rapportools`; this is to allow the use of `is.boolean` method to check one of the stopping criteria in the beginning.

The basic mechanism behind `pred.forestRK` function is the following:

When the function is called, it calls `forestRK` function after passing the user-specified training data as an argument, in order to first generate the `forestRK` object. After that, the function uses `pred.treeRK` function to make predictions on the test observations based on each individual tree in the `forestRK` object. Once the individual prediction from each tree are obtained for all of the test observations, the function stores those individual predictions under a big dataframe. Once that data frame is complete, then the function collapses the results by the rule of the majority votes. For example, for the m-th observation from the test set, if the most frequently predicted class type for that m-th test observation by all of the rkTrees in the forest is class type 'A', then by the rule of the majority votes, the `pred.forestRK` function will assign class 'A' as the predicted class type for that m-th test observation based on the `forestRK` model.

Usage

```
pred.forestRK(x.test = data.frame(), x.training = data.frame(),
              y.training = c(), y.factor.levels,
              min.num.obs.end.node.tree = 5,
              nbags, samp.size, entropy = TRUE)
```

Arguments

`x.test` a numericized data frame of covariates of the data points on which we want to make our predictions (typically the test observations); `x.test` can be obtained by applying the `x.organizer()` function. `x.test` should contain no `NA` or `NaN`'s.

`x.training` a numericized data frame of covariates of data points from which we build our `forestRK` model (typically the training observations); `x.training` can be obtained by applying the `x.organizer()` function. `x.training`should contain no `NA` or `NaN`'s.

`y.training` a vector that stores numericized class types of the training data points; `y.training` should contain no `NA` or `NaN`'s.

`min.num.obs.end.node.tree`
the minimum number of observations that we want each end node of our `rktree` to contain. Default is set to 5.

`nbags` the number of bootstrap samples that we want to generate to form a `forest-RK`.

`samp.size` the number of data points that we want each of our bootstrap sample to contain.

`y.factor.levels`
a vector of original names of all class types that the user considers in his or her study (can be obtained via `y.organizer()$y.factor.levels`)

`entropy` `TRUE` if we use Entropy as the splitting criteria; `FALSE` if we use the Gini Index for the splitting criteria. Default is set to `TRUE`.

Value

A list containing the following items:

`x.test` the original test dataset that we used to make predictions.

`df.of.predictions.for.all.observations`
a data frame storing predicted class types for all test observations from each tree in the forest; each row of this data frame pertains to individual test observation, and each column pertain to a specific tree from the `forestRK` model. This data frame stores predicted (numericized) class type of each test observation from each tree in the `forestRK` model.

`forest.rk` a `forestRK` object that was generated in the beginning of the function call.

`test.prediction.df.list`
a list of data frames storing the `prediction.df`'s (the data frame that can be obtained via `pred.treeRK()$prediction.df`) of the test observations that were generated from each tree in the `forestRK` model. Note that the test data points in `test.prediction.df.list` are re-ordered by the increasing original observation index number.

`pred.for.obs.forest.rk`
a vector that stores the actual predicted class labels of the test observations instead of their numericized (integer) class types. Note that the test data points in `pred.for.obs.forest.rk` are re-ordered by the increasing original observation index number.

`num.pred.for.obs.forest.rk`
the numericized version of `pred.for.obs.forest.rk`. Note that the test data points in `num.pred.for.obs.forest.rk` are re-ordered by the increasing original observation index number.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`pred.treeRK` `forestRK`

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

## numericize the data
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
x.test <- x.organizer(iris[,1:4], encoding = "num")[c(26:50,76:100,126:150),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

y.factor.levels <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.factor.levels

## make prediction from a random forest RK model
## typically the nbags and samp.size has to be much larger than 30 and 50
pred.forest.rk <- pred.forestRK(x.test = x.test, x.training = x.train,
                                y.training = y.train,
                                y.factor.levels,
                                min.num.obs.end.node.tree = 6,
                                nbags = 30, samp.size = 50, entropy = FALSE)
pred.forest.rk$test.prediction.df.list[[10]]
pred.forest.rk$pred.for.obs.forest.rk # etc....
```

pred.treeRK *Make predictions on the test observations based on a rktree model*

Description

Makes predictions on the observations in the test dataset based on the `rktree` model constructed from the training dataset.

Please be aware that, at the end of the `pred.treeRK` function, the test data points in `prediction.df` are re-ordered by the increasing original index number (the original rownames) of those test observations. So if you shuffled the data before seperating them into a training and a test set, the order of the data points in which they are presented under the data frame `prediction.df` may not be same as the shuffled order in your original test set.

Users of this function may be interested in identifying the original name of the numericized predicted class type shown in the last column of data frame `prediction.df`. This can easily be done by extracting the attribute `y.factor.levels` from the `y.organizer` object. For example, if the data frame `prediction.df` indicates that the predicted class type of the 1st test observation is "2", that means the actual name of the predicted class type for that 1st test observation is indicated as the 2nd element of the vector `y.organizer.object$y.factor.levels` that we can obtain during the data cleaning phase.

The `pred.treeRK` function makes a use of the list of hierarchical flags generated by the `construct.treeRK` function; the function uses the list of hierarchical flag as a guide to how it should split the test set to make predictions. The function `pred.treeRK` itself actually generates a list of hierarchical flag of its own as it splits the test set, and at the end of the function `pred.treeRK` tries to match the list of hierarchical flag it generated with the list of hierarchical flag from the `construct.treeRK` function. If the two flags match exactly, then it is a good sign since this would imply that the splitting on the test set was done in the manner consistent with how the training set was split when the rkTree in question was built. If there is any difference in the two flags, however, this is not a good sign since it would signal that the splitting on the test set has done in a different manner than how the splitting was done on the training set; if the mismatch occurs, the `pred.treeRK` function will stop and throw an error. For more information about the hierarchical flags of a `rkTree`, please see the `construct.treeRK` section of this documentation.

Usage

```
pred.treeRK(X = data.frame(), rktree = construct.treeRK())
```

Arguments

`X`	a numericized data frame of covariates of the test observations or the observations that we want to make predictions for (obtained via `x.organizer()`). `X` should contain no `NA` or `NaN`'s.
`rktree`	a `construct.treeRK` object.

Value

A list containing the following items:

prediction.df
: a data frame of test observations. If prediction.df has n columns, the first n-1 columns will contain the numericized covariates of the test observations, and the very last n-th column will contain the predicted numericized class type for each of those test observations. Note that, at the end of the pred.treeRK function, the test data points in prediction.df are re-ordered by theincreasing original observation index number.

flag.pred
: the hierarchical flag of splits performed on the test set by applying the rktree model in question.

Author(s)

Hyunjin Cho, <h56cho@uwaterloo.ca> Rebecca Su, <y57su@uwaterloo.ca>

See Also

pred.forestRK

Examples

```
## example: iris dataset
## load the forestRK package
library(forestRK)

## numericize the data
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
x.test <- x.organizer(iris[,1:4], encoding = "num")[c(26:50,76:100,126:150),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

## Construct a tree
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
tree.entropy <- construct.treeRK(x.train, y.train)
tree.gini <- construct.treeRK(x.train, y.train,
                              min.num.obs.end.node.tree = 6, entropy = FALSE)

## Make predictions on the test set based on the constructed rktree model
# last column of prediction.df stores predicted class on the test observations
# based on a given rktree
prediction.df <- pred.treeRK(X = x.test, tree.entropy)$prediction.df
flag.pred <- pred.treeRK(X = x.test, tree.entropy)$flag.pred
```

`var.used.forestRK` *Extract the list of covariates used to perform the splits to generate a particular tree(s) in a `forestRK` object*

Description

Spits out the list of covariates used to perform the splits to generate a particular tree(s) in a `forestRK` object that the user provided.

The function extracts the list of names of covariates used in splits to construct a single or a multiple numbers of trees from a `forestRK` object. The `var.used.forestRK` displays the actual name of the covariate used for each split (not their numericized ones), consistent to the exact order of the split; for instance, the 1st element of the vector `covariate.used.for.split.tree[["6"]]` from the example below is the covariate on which the 1st split had occured while the 6th tree in the `forestRK.1` object was built.

Each vector in the list are named by the exact indices of the tree; for example, if the code `obj <-var.used.forestRK(forestRK.1,tree.index=c(4,5,6))` is used to extract the list of covariates used for splitting to construct 4th, 5th, and 6th trees in the forest, and the user can retrieve the information pertains explicitly to the 6th tree in the forest by doing `obj[["6"]]`.

Usage

```
var.used.forestRK(forestRK.object = forestRK(), tree.index = c())
```

Arguments

`forestRK.object`
: a `forestRK` object.

`tree.index`
: a vector storing the indices of the trees that we are interested to examine.

Value

A list of vectors that stores the names of covariates on which each split was performed to construct the specific tree(s) in a `forestRK` model that the user provided.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`forestRK`

Examples

```
library(forestRK)

x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
y.train <- y.organizer(iris[c(1:25,51:75,101:125),5])$y.new

# random forest
# min.num.obs.end.node.tree is set to 5 by default;
# entropy is set to TRUE by default
# normally nbags and samp.size have to be much larger than 30 and 50
forestRK.1 <- forestRK(x.train, y.train, nbags = 30, samp.size = 50)

# prediction from a random forest RK
covariate.used.for.split.tree <- var.used.forestRK(forestRK.1,
                                                   tree.index=c(4,5,6))

# retrieve the list of covariates used for splitting for the 'tree #6'
covariate.used.for.split.tree[["6"]]
```

`x.organizer` *Numericizing a data frame of covariates from the original dataset via Binary or Numeric Encoding*

Description

Takes the original data frame of covariates as an input (which may or may not be numeric), and converts it into a numericized data frame by applying either Binary or Numeric Encoding.

Binary Encoding for categorical features are recommended for tree ensembles when the cardinality of categorical feature is >= 1000; Numeric Encoding for categorical features are recommended for tree ensembles when the cardinality of categorical features is < 1000.

For more information about the Binary and Numeric Encoding and their effectiveness under different cardinality, please visit: https://medium.com/data-design/ visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931

NOTE: In order to use other functions within the forestRK package, you must ensure that the numericized data frame of covariates (the `x.organizer` object) contains no missing record, that is, you have to remove any record containing `NA` or `NaN` prior to applying the `x.organizer` function.

Following is the summary of the data cleaning process with `x.organizer()`:

1. remove all `NA` or `NaN`'s from the data in hand. 2. split the data into a data frame that contains covariates of ALL data points, (BOTH training and test observations), and a vector that contains class types of the training observations; 3. apply the `x.organizer` to the big data frame of covariates of all observations. 4. split the `x.organizer` output into a training and a test set, as needed.

PROPER DATA CLEANING IS ABSOLUTELY NECESSARY FOR forestRK FUNCTIONS TO WORK!

Usage

```
x.organizer(x.dat = data.frame(), encoding = c("num","bin"))
```

Arguments

`x.dat` a data frame storing covariates of each observation (can be either numeric or non-numeric) from the original data; `x.dat` should contain no `NA` or `NaN`. The rownames of `x.dat` should be numerical index for each observations.

`encoding` type of encoding done for the categorical features; "`num`" stands for Numeric Encoding, and "`bin`" stands for Binary Encoding. When the data in question only has numeric features, then the user can select either one of "`num`" or "`bin`", and the `x.organizer` function will just return the original numeric dataset.

Value

A numericized data frame of the covariates from the original data obtained via either Numeric or Binary Encoding.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`y.organizer`

Examples

```
## example: iris dataset
library(forestRK) # load the package forestRK

## Basic Procedures
## 1. Apply x.organizer to a data frame that stores covariates of
## ALL observations (BOTH training and test observations)
## 2. Split the output from 1 into a training and a test set, as needed

# note: iris[,1:4] are the columns of the iris dataset that stores
# covariate values

# covariates of training data set
x.train <- x.organizer(iris[,1:4], encoding = "num")[c(1:25,51:75,101:125),]
```

y.organizer *Numericize the vector containing categorical class type(y) of the original data*

Description

Numericizes a vector of categorical class type of each (training) data point.

NOTE: In order to use other functions within the forestRK package, you must ensure that the original vector of class type `y` contains no missing record (`NA`, `NaN`), that is, you have to remove any record containing `NA` or `NaN` prior to applying the `y.organizer` function.

Following is the summary of the data cleaning process with `y.organizer()`: 1. remove all `NA` or `NaN`'s from the dataset in hand. 2. split the training dataset into a data frame that contains covariates of ALL observations (BOTH training and test observations), and a vector that contains class types of the training observations; 3. apply the `y.organizer` to the vector that contains class type of each training observation.

PROPER DATA CLEANING IS NECESSARY FOR THE forestRK FUNCTIONS TO WORK!

Usage

```
y.organizer(y = c())
```

Arguments

`y`	a vector containing the class type of each observation from the dataset on which we want to build our rktree models (the training dataset); `y` should contain no `NA` or `NaN`.

Value

A list containing the following items:

`y.new`	a vector containing numericized class type of each observation from the dataset from which our rktree models are generated from. (these are typically the observations from the training set)
`y.factor.levels`	a vector storing original names of the numericized class types.

Author(s)

Hyunjin Cho, `<h56cho@uwaterloo.ca>` Rebecca Su, `<y57su@uwaterloo.ca>`

See Also

`x.organizer`

Examples

```
## example: iris dataset
## load the package forestRK
library(forestRK)

## Basic Procedures:
## 1. Extract the portion of the data that stores class type of each
##    TRAINING observation, and make it as a vector
## 2. apply y.organizer function to the vector obtained from 1

y.train <- y.organizer(as.vector(iris[c(1:25,51:75,101:125),5]))
## retrieves the original names of each class type, if the class names
## were originally non-numeric
y.train$y.factor.levels
## retrieves the numericized vector that stores classification category
y.train$y.new
```