Package 'inTrees'

April 22, 2024

Title Interpret Tree Ensembles

Version 1.4

Date 2024-04-22

Imports RRF, arules, gbm, xtable, xgboost, data.table, methods

Description For tree ensembles such as random forests, regularized random forests and gradient boosted trees, this package provides functions for: extracting, measuring and pruning rules; selecting a compact rule set; summarizing rules into a learner; calculating frequent variable interactions; formatting rules in latex code. Reference: Interpreting tree ensembles with inTrees (Houtao Deng, 2019, <doi:10.1007/s41060-018-0144-8>).

Maintainer Houtao Deng <softwaredeng@gmail.com>

BugReports https://github.com/softwaredeng/inTrees/issues

License GPL (>= 3)

NeedsCompilation no

Repository CRAN

Date/Publication 2024-04-22 21:26:06

Author Houtao Deng [aut, cre],
Xin Guan [aut],
Vadim Khotilovich [aut]

Contents

`applyLearner` *apply a simplified tree ensemble learner (STEL) to data*

Description

apply STEL to data and get predictions

Usage

```
applyLearner(learner, X)
```

Arguments

`learner`	a matrix with rules ordered by priority
`X`	predictor variable matrix

Value

predictions for the data

See Also

`buildLearner`

`buildLearner` *build a simplified tree ensemble learner (STEL)*

Description

Build a simplified tree ensemble learner (STEL). Currently works only for classification problems.

Usage

```
buildLearner(ruleMetric, X, target, minFreq = 0.01)
```

Arguments

ruleMetric	a matrix including the conditions, predictions, and and metrics
X	predictor variable matrix
target	target variable
minFreq	minimum frequency of a rule condition in order to be included in STEL.

Value

a matrix including the conditions, prediction, and metrics, ordered by priority.

Author(s)

Houtao Deng

References

Houtao Deng, Interpreting Tree Ensembles with inTrees, technical report, 2014

Examples

```
data(iris)
library(RRF)
X <- iris[,1:(ncol(iris)-1)]
target <- iris[,"Species"]
rf <- RRF(X,as.factor(target),ntree=100) # build an ordinary RF
treeList <- RF2List(rf)
ruleExec <- extractRules(treeList,X)
ruleExec <- unique(ruleExec)
ruleMetric <- getRuleMetric(ruleExec,X,target) # measure rules
ruleMetric <- pruneRule(ruleMetric,X,target) # prune each rule
#ruleMetric <- selectRuleRRF(ruleMetric,X,target) # rule selection
learner <- buildLearner(ruleMetric,X,target)
pred <- applyLearner(learner,X)
read <- presentRules(learner,colnames(X)) # more readable format

# format the rule and metrics as a table in latex code
library(xtable)
print(xtable(read), include.rownames=FALSE)
print(xtable(ruleMetric[1:2,]), include.rownames=FALSE)
```

---

computeRuleInfor *compute rule information*

Description

compute rule information

Usage

```
computeRuleInfor(instIx, pred, target)
```

Arguments

instIx	indices of the intances
pred	prediction from a rule
target	target values for the instances

Value

return error and frequency

Examples

```
# this is an internal function.
```

---

dataSimulate *Simulate data*

Description

Simulate data

Usage

```
dataSimulate(flag = 1, nCol = 20, nRow = 1000)
```

Arguments

flag	1 (default): team optimization; 2: non-linear; 3: linear.
nCol	the number of columns in the data set. must >= 2.
nRow	the number of rows in the data set.

Value

predictor variable matrix and target variable

Examples

```
res <- dataSimulate(flag=1)
X <- res$X;
target <- res$target
```

dicretizeVector *discretize a variable*

Description

discretize a variable

Usage

```
dicretizeVector(v, K = 3)
```

Arguments

v	vector
K	discretize into up to K levels with equal frequency

Value

discretized levels for v

Examples

```
data(iris)
dicretizeVector(iris[,1],3)
```

extractRules *Extract rules from a list of trees*

Description

Extract rule conditions from a list of trees. Use functions RF2List/GBM2List to transform RF/GBM objects to list of trees.

Usage

```
extractRules(treeList, X, ntree = 100, maxdepth = 6, random = FALSE, digits = NULL)
```

Arguments

treeList	tree list
X	predictor variable matrix
ntree	conditions are extracted from the first ntree trees
maxdepth	conditions are extracted from the top maxdepth levels from each tree
random	the max depth for each tree is an integer randomly chosen between 1 and maxdepth
digits	digits for rounding

Value

a set of rule conditions

Examples

```
library(RRF)
data(iris)
X <- iris[,1:(ncol(iris)-1)]
target <- iris[,"Species"]
rf <- RRF(X,as.factor(target),ntree=100) # build an ordinary RF
treeList <- RF2List(rf)
ruleExec <- extractRules(treeList,X,digits=4) # transform to R-executable rules
ruleExec <- unique(ruleExec)
```

`formatGBM` *internal*

Description

internal

Usage

```
formatGBM(gbmList, splitBin,X)
```

Arguments

`gbmList`	
`splitBin`	
`X`	predictor variable matrix

Value

No return value

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (gbmList, splitBin,X)
{
    for (j in 1:length(gbmList$list)) {
        a <- gbmList$list[[j]]
        rownames(a) <- 1:nrow(a)
        a$status <- a$SplitVar
        a <- a[, c("LeftNode", "RightNode", "MissingNode", "SplitVar",
            "SplitCodePred", "status")]
        a[which(a[, "SplitVar"] >= 0), c("SplitVar", "LeftNode",
            "RightNode", "MissingNode")] <- a[which(a[, "SplitVar"] >=
            0), c("SplitVar", "LeftNode", "RightNode", "MissingNode")] +
            1
        ix <- a$MissingNode[which(a$MissingNode > 0)]
        if (length(ix) > 0)
            a$status[ix] <- 10
```

```
        a <- a[, c("LeftNode", "RightNode", "SplitVar", "SplitCodePred",
            "status")]
        cat <- which(sapply(X, is.factor) & !sapply(X, is.ordered))
        ix <- which(a[, "SplitVar"] %in% cat)
        for (i in ix) a[i, "SplitCodePred"] <- splitBin[a[i,
            "SplitCodePred"] + 1]
        colnames(a) <- c("left daughter", "right daughter", "split var",
            "split point", "status")
        gbmList$list[[j]] <- a
    }
    return(gbmList)
}
```

GBM2List *Transform gbm object to a list of trees*

Description

Transform gbm object to a list of trees that can be used for rule condition extraction

Usage

```
GBM2List(gbm1,X)
```

Arguments

gbm1	gbm object
X	predictor variable matrix

Value

a list of trees in an inTrees-required format

See Also

RF2List

Examples

```
library(gbm)
data(iris)
X <- iris[,1:(ncol(iris)-1)]
target <- iris[,"Species"]
gbmFit <- gbm(Species~ ., data=iris, n.tree = 400,
               interaction.depth = 10,distribution="multinomial")
treeList <- GBM2List(gbmFit,X)
ruleExec = extractRules(treeList,X)
ruleExec <- unique(ruleExec)
#ruleExec <- ruleExec[1:min(2000,length(ruleExec)),,drop=FALSE]
ruleMetric <- getRuleMetric(ruleExec,X,target)
ruleMetric <- pruneRule(ruleMetric,X,target)
ruleMetric <- unique(ruleMetric)
learner <- buildLearner(ruleMetric,X,target)
pred <- applyLearner(learner,X)
```

```
readableLearner <- presentRules(learner,colnames(X)) # more readable format
err <- 1-sum(pred==target)/length(pred);
```

`getFreqPattern` *calculate frequent variable interactions*

Description

calculate frequent variable interactions

Usage

```
getFreqPattern(ruleMetric, minsup = 0.01, minconf = 0.5, minlen = 1, maxlen = 4)
```

Arguments

`ruleMetric`	a matrix including conditions, predictions, and the metrics
`minsup`	minimum support of conditions in a tree ensemble
`minconf`	minimum confidence of the rules
`minlen`	minimum length of the conditions
`maxlen`	max length of the conditions

Value

a matrix including frequent variable interations (in a form of conditions), predictions, length, support, and confidence.

Examples

```
library(RRF)
library(arules)
data(iris)
X <- iris[,1:(ncol(iris)-1)]
target <- iris[,"Species"]
rf <- RRF(X,as.factor(target),ntree=100) # build an ordinary RF
treeList <- RF2List(rf)
ruleExec <- extractRules(treeList,X) # transform to R-executable rules
ruleMetric <- getRuleMetric(ruleExec,X,target)
freqPattern <- getFreqPattern(ruleMetric)
freqPatternMetric <- getRuleMetric(freqPattern,X,target)
```

getRuleMetric *Assign outcomes to a conditions, and measure the rules*

Description

Assign outcomes to a conditions, and measure the rules

Usage

```
getRuleMetric(ruleExec, X, target)
```

Arguments

ruleExec	a set of rule conditions
X	predictor variable matrix
target	target variable

Value

a matrix including the condictions, predictions, and metrics

References

Houtao Deng, Interpreting Tree Ensembles with inTrees, technical report, 2014

Examples

```
library(RRF)
data(iris)
X <- iris[,1:(ncol(iris)-1)]
target <- iris[,"Species"]
rf <- RRF(X,as.factor(target),ntree=100) # build an ordinary RF
treeList <- RF2List(rf)
ruleExec <- extractRules(treeList,X) # transform to R-executable rules
ruleExec <- unique(ruleExec)
ruleMetric <- getRuleMetric(ruleExec,X,target) # measure rules
```

getTypeX *get type of each variable*

Description

get type of each variable: numeric or categorical

Usage

```
getTypeX(X)
```

Arguments

X

Value

A vector indicating each variable's type: numeric: 1; categorical: 2

`lookupRule` *internal*

Description

internal

Usage

```
lookupRule(rules, strList)
```

Arguments

`rules`

`strList`

Value

rules that matched to strList

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (rules, strList)
{
    ix <- grep(strList[1], rules[, "condition"])
    if (length(strList) >= 2) {
        for (i in 2:length(strList)) {
            ix2 <- grep(strList[i], rules[, "condition"])
            ix <- intersect(ix, ix2)
        }
    }
    if (length(ix) >= 1)
        return(rules[ix, , drop = FALSE])
    if (length(ix) == 0)
        return(NULL)
  }
```

measureRule *internal*

Description

internal

Usage

```
measureRule(ruleExec, X, target, pred = NULL, regMethod = "mean")
```

Arguments

ruleExec

X

target

pred

regMethod

Value

data frame including rule's length, frequency, error, rule condition and prediction

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ruleExec, X, target, pred = NULL)
{
    len <- length(unlist(strsplit(ruleExec, split = " & ")))
    origRule <- ruleExec
    ruleExec <- paste("which(", ruleExec, ")")
    ixMatch <- eval(parse(text = ruleExec))
    if (length(ixMatch) == 0) {
        v <- c("-1", "-1", "-1", "", "")
        names(v) <- c("len", "freq", "err", "condition", "pred")
        return(v)
    }
    ys <- target[ixMatch]
    freq <- round(length(ys)/nrow(X), digits = 3)
    if (is.numeric(target)) {
        ysMost <- mean(ys)
        err <- sum((ysMost - ys)^2)/length(ys)
    }
    else {
        if (length(pred) > 0) {
            ysMost = pred
        }
        else {
            ysMost <- names(which.max(table(ys)))
```

```
        }
        conf <- round(table(ys)[ysMost]/sum(table(ys)), digits = 3)
        err <- 1 - conf
    }
    rule <- origRule
    v <- c(len, freq, err, rule, ysMost)
    names(v) <- c("len", "freq", "err", "condition", "pred")
    return(v)
  }
```

Num2Level *internal function*

Description

internal function

Usage

```
Num2Level(rfList, splitV)
```

Arguments

rfList	
splitV	

Value

data frame with numeric variables converted to categorical variables.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (rfList, splitV)
{
    for (i in 1:rfList$ntree) {
        rfList$list[[i]] <- data.frame(rfList$list[[i]])
       rfList$list[[i]][, "prediction"] <- data.frame(dicretizeVector(rfList$list[[i]][,
            "prediction"], splitV))
        colnames(rfList$list[[i]]) <- c("left daughter", "right daughter",
            "split var", "split point", "status", "prediction")
    }
    return(rfList)
  }
```

presentRules *Present a learner using column names instead of X[i,]*

Description

Present a learner using column names instead of X[i,]

Usage

```
presentRules(rules, colN, digits)
```

Arguments

rules	a set of rules
colN	a vector including the column names
digits	digits for rounding

Value

a matrix including the conditions (with column names), etc.

See Also

buildLearner

Examples

```
# See function "buildLearner"
```

pruneRule *Prune irrevant variable-value pair from a rule condition*

Description

Prune irrevant variable-value pair from a rule condition

Usage

```
pruneRule(rules, X, target, maxDecay = 0.05, typeDecay = 2)
```

Arguments

rules	A metrix including the rules and metrics
X	predictor variable matrix
target	target variable vector
maxDecay	threshold of decay
typeDecay	1: relative error; 2: error; default :2

Value

A matrix including the rules each being pruned, and metrics

Author(s)

Houtao Deng

References

Houtao Deng, Interpreting Tree Ensembles with inTrees, technical report, 2014

See Also

buildLearner

Examples

```
# see function "buildLearner"
```

---

pruneSingleRule *internal*

Description

internal

Usage

```
pruneSingleRule(rule, X, target, maxDecay, typeDecay)
```

Arguments

rule

X

target

maxDecay

typeDecay

Value

a pruned rule and its metrics.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (rule, X, target, maxDecay, typeDecay)
{
    newRuleMetric <- measureRule(rule["condition"], X, target)
    errOrig <- as.numeric(newRuleMetric["err"])
    ruleV <- unlist(strsplit(rule["condition"], split = " & "))
    pred <- rule["pred"]
    if (length(ruleV) == 1)
        return(newRuleMetric)
    for (i in length(ruleV):1) {
        restRule <- ruleV[-i]
        restRule <- paste(restRule, collapse = " & ")
        metricTmp <- measureRule(restRule, X, target, pred)
        errNew <- as.numeric(metricTmp["err"])
        if (typeDecay == 1) {
            decay <- (errNew - errOrig)/max(errOrig, 1e-06)
        }
        else {
            decay <- (errNew - errOrig)
        }
        if (decay <= maxDecay) {
            ruleV <- ruleV[-i]
            newRuleMetric <- metricTmp
            if (length(ruleV) <= 1)
                break
        }
    }
    return(newRuleMetric)
  }
```

RF2List *Transform a random forest object to a list of trees*

Description

Transform a random forest object to a list of trees

Usage

```
RF2List(rf)
```

Arguments

rf	random forest object

Value

a list of trees

See Also

GBM2List

Examples

```
library(RRF)
data(iris)
X <- iris[,1:(ncol(iris)-1)]
target <- iris[,"Species"]
rf <- RRF(X,as.factor(target),ntree=100) # build an ordinary RF
treeList <- RF2List(rf)
ruleExec <- extractRules(treeList,X) # transform to R-executable rules
```

---

rule2Table *internal function*

Description

internal function

Usage

```
rule2Table(ruleExec, X, target)
```

Arguments

ruleExec

X

target

Value

a matrix of indicators matching each rule condition and each row of data

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ruleExec, X, target)
{
    I <- rep(0, nrow(X))
    ruleExec <- paste("which(", ruleExec, ")")
    ixMatch <- eval(parse(text = ruleExec))
    if (length(ixMatch) > 0)
        I[ixMatch] <- 1
    names(I) = NULL
    return(I)
  }
```

ruleList2Exec *internal*

Description

internal

Usage

```
ruleList2Exec(X, allRulesList)
```

Arguments

X

allRulesList

Value

data frame containing rule conditions

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (X, allRulesList)
{
    typeX = getTypeX(X)
    ruleExec <- unique(t(sapply(allRulesList, singleRuleList2Exec,
        typeX = typeX)))
    ruleExec <- t(ruleExec)
    colnames(ruleExec) <- "condition"
    return(ruleExec)
  }
```

selectRuleRRF *select a set of relevant and non-redundant rules*

Description

select a set of relevant and non-redundant rules using regularized random forests

Usage

```
selectRuleRRF(ruleMetric, X, target)
```

Arguments

ruleMetric	a matrix including the rules and metrics
X	predictor variable matrix
target	response variable

Value

a matrix including a set of relevant and non-redundant rules, and their metrics

Author(s)

Houtao Deng

See Also

buildLearner

Examples

```
 # See function "buildLearner:
```

---

singleRuleList2Exec *internal*

Description

internal

Usage

```
singleRuleList2Exec(ruleList, typeX)
```

Arguments

ruleList	
typeX	

Value

data frame containing rule conditions

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ruleList, typeX)
{
    ruleExec <- ""
    vars <- ls(ruleList)
```

```
    vars <- vars[order(as.numeric(vars))]
    for (i in 1:length(vars)) {
        if (typeX[as.numeric(vars[i])] == 2) {
            values <- paste("c(", paste(paste("'", ruleList[[vars[i]]],
                "'", sep = ""), collapse = ","), ")", sep = "")
            tmp = paste("X[,", vars[i], "] %in% ", values, sep = "")
        }
        else {
            tmp = ruleList[[vars[i]]]
        }
        if (i == 1)
            ruleExec <- paste(ruleExec, tmp, sep = "")
        if (i > 1)
            ruleExec <- paste(ruleExec, " & ", tmp, sep = "")
    }
    return(c(ruleExec))
  }
```

---

sortRule *internal*

Description

internal

Usage

```
sortRule(M, decreasing = TRUE)
```

Arguments

M

decreasing

Value

sorted rule conditions

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (M, decreasing = TRUE)
{
    qIx = order((1 - as.numeric(ruleMetric[, "err"])), as.numeric(ruleMetric[,
        "freq"]), -as.numeric(ruleMetric[, "len"]), decreasing = decreasing)
    return(M[qIx, ])
  }
```

treeVisit *internal function*

Description

internal function

Usage

```
treeVisit(tree, rowIx, count, ruleSet, rule, levelX, length, max_length, digits)
```

Arguments

tree

rowIx

count

ruleSet

rule

levelX

length

max_length

digits

Value

a list containing rules and the count

voteAllRules *internal*

Description

Predictions from a rule set

Usage

```
voteAllRules(ruleMetric, X, type = "r", method = "median")
```

Arguments

ruleMetric	rules and metrics
X	predictor variable matrix
type	regression or classification
method	for regression, use median or average

Value

predictions from the rule set

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ruleMetric, X, type = "r", method = "median")
{
    xVoteList = vector("list", nrow(X))
    predY <- rep("", nrow(X))
    for (i in 1:nrow(ruleMetric)) {
        ixMatch <- eval(parse(text = paste("which(", ruleMetric[i,
            "condition"], ")")))
        if (length(ixMatch) == 0)
            next
        for (ii in ixMatch) {
            xVoteList[[ii]] = c(xVoteList[[ii]], ruleMetric[i,
                "pred"])
        }
    }
    for (i in 1:length(xVoteList)) {
        thisV <- xVoteList[[i]]
        if (length(thisV) == 0)
            next
        if (type == "c")
            predY[i] <- names(table(thisV)[which.max(table(thisV))])
        if (type == "r") {
            thisV = as.numeric(thisV)
            if (method == "median") {
                predY[i] <- median(thisV)
            }
            else {
                predY[i] <- mean(thisV)
            }
        }
    }
    if (type == "r")
        predY <- as.numeric(predY)
    return(predY)
  }
```

XGB2List *Transform an xgboost object to a list of trees*

Description

Transform an xgboost object to a list of trees

Usage

```
XGB2List(xgb, X)
```

Arguments

xgb	xgboost object
X	predictor variable matrix

Value

a list of trees in an inTrees-required format

See Also

XGB2List

Examples

```
library(data.table)
library(xgboost)
# test data set 1: iris
X <- within(iris,rm("Species")); Y <- iris[,"Species"]
X <- within(iris,rm("Species")); Y <- iris[,"Species"]
model_mat <- model.matrix(~. -1, data=X)
xgb <- xgboost(model_mat, label = as.numeric(Y) - 1, nrounds = 20,
 objective = "multi:softprob", num_class = 3 )
tree_list <- XGB2List(xgb,model_mat)
```

Index