Package ‘qCBA’

January 15, 2020

Title Quantitative Classification by Association Rules
Version 0.4
Date 2020-01-15
Author Tomas Kliegr
Maintainer Tomas Kliegr <kliegr@gmail.com>
Description CBA postprocessing algorithm that creates smaller models for datasets containing quantitative (numerical) attributes. Article describing QCBA is published in Tomas Kliegr (2017) <arXiv:1711.10166>.
Depends R (>= 2.7.0), arules, rJava (>= 0.5-0), arc (>= 1.1.2), methods
Suggests arulesCBA (>= 1.1.2), rCBA (>= 0.3.0), sbrl, stringr
SystemRequirements Java (>= 8)
URL https://github.com/kliegr/QCBA
BugReports https://github.com/kliegr/QCBA/issues
License GPL-3
RoxygenNote 6.1.1
NeedsCompilation no
Repository CRAN
Date/Publication 2020-01-15 16:10:02 UTC

R topics documented:

arulesCBA2arcCBAModel ........................................... 2
customCBARuleModel-class ..................................... 3
mapDataTypes ....................................................... 3
predict.qCBARuleModel .......................................... 4
qcba ................................................................. 5
qcbaHumTemp ....................................................... 6
qcbaIris ............................................................. 7
qcbaIris2 ............................................................ 7
arulesCBA2arcCBAModel  arulesCBA2arcCBAModel Converts a model created by arulesCBA so that it can be passed to qCBA

Description

Creates instance of arc CBAmodel class from the arc package. Instance of CBAmodel can then be passed to qCBA.

Usage

arulesCBA2arcCBAModel(arulesCBAModel, cutPoints, rawDataset, classAtt, attTypes)

Arguments

arulesCBAModel  a object returned by arulesCBA::CBA()
cutPoints  specification of cutpoints applied on the data before they were passed to rCBA::build
rawDataset  the raw data (before discretization). This dataset is used to guess attribute types if attTypes is not passed
classAtt  the name of the class attribute
attTypes  vector of attribute types of the original data. If set to null, you need to pass rawDataset.

Examples

if (! requireNamespace("arulesCBA", quietly = TRUE)) {
  message("Please install arulesCBA: install.packages('arulesCBA')")
} else {
  if (identical(Sys.getenv("NOT_CRAN"), "true")) {
    classAtt <- "Species"
    discrModel <- discrNumeric(iris, classAtt)
    irisDisc <- as.data.frame(lapply(discrModel$Disc.data, as.factor))
    arulesCBAModel <- arulesCBA::CBA(Species ~ ., data = irisDisc, supp = 0.05, conf=0.9, lhs.support=TRUE)
    CBAmodel <- arulesCBA2arcCBAModel(arulesCBAModel, discrModel$cutp, iris, classAtt)
    qCBAmodel <- qcba(cbaRuleModel=CBAmodel, datadf=iris)
    print(qCBAmodel@rules)
  }
}
customCBARuleModel-class

rCBARuleModel

Description

This class represents a CBA rule-based classifier, where rules are represented as string vectors in a data frame.

Slots

- rules: dataframe output by rCBA
- cutp: list of cutpoints
- classAtt: name of the target class attribute
- attTypes: attribute types

mapDataTypes

Map R types to qCBA

Description

The QCBA Java implementation uses different names of some data types than are used in this R wrapper.

Usage

mapDataTypes(Rtypes)

Arguments

- Rtypes: Vector with R data types

Value

Vector with qCBA data types

Examples

mapDataTypes(unname(sapply(iris, class)))
predict.qCBARuleModel  

Aplies qCBARuleModel

Description

Applies qcba rule model on provided data. Automatically detects whether one-rule or multi-rule classification is used.

Usage

```R
## S3 method for class 'qCBARuleModel'
predict(object, newdata, testingType,
         loglevel = "WARNING", outputFiringRuleIDs = FALSE, ...)
```

Arguments

- `object`: qCBARuleModel class instance
- `newdata`: data frame with data
- `testingType`: either `mixture` for multi-rule classification or `firstRule` for one-rule classification. Applicable only when model is loaded from file.
- `loglevel`: logger level from java.util.logging
- `outputFiringRuleIDs`: if set to TRUE, instead of predictions, the function will return one-based IDs of rules used to classify each instance (one rule per instance).
- `...`: other arguments (currently not used)

Value

vector with predictions.

See Also

qcba

Examples

```R
allData <- datasets::iris[sample(nrow(datasets::iris)),]
trainFold <- allData[1:100,]
testFold <- allData[101:nrow(datasets::iris),]
rmCBA <- cba(trainFold, classAtt="Species")
rmqCBA <- qcba(cbaRuleModel=rmCBA, datadf=trainFold)
print(rmqCBA@rules)
prediction <- predict(rmqCBA, testFold)
acc <- CBARuleModelAccuracy(prediction, testFold[[rmqCBA@classAtt]])
message(acc)
firingRuleIDs <- predict(rmqCBA, testFold, outputFiringRuleIDs=TRUE)
message("The second instance in testFold was classified by the following rule")
```
qcba

message(rmqCBA@rules[firingRuleIDs[2],1])
messagethe second instance is"
messagetestFold[2,]

qcba qCBA Quantitative CBA

Description

Creates QCBA model by from a CBA rule model. The default values are set so that the function postprocesses CBA models, reducing their size. The resulting model has the same structure as CBA model: it is composed of an ordered list of crisp conjunctive rules, intended to be applied for one-rule classification. The experimental annotate and fuzzification parameters will trigger more complex postprocessing of CBA models: rules will be annotated with probability distributions and optionally fuzzy borders. The intended use of such models is multi-rule classification. The `predict` function automatically determines whether the input model is a CBA model or an annotated model.

Usage

qcba(cbaRuleModel, datadf, extendType = "numericOnly",
     defaultRuleOverlapPruning = "transactionBased",
     attributePruning = TRUE, trim_literal_boundaries = TRUE,
     continuousPruning = FALSE, postpruning = "cba",
     fuzzification = FALSE, annotate = FALSE, ruleOutputPath,
     minImprovement = 0, minCondImprovement = -1, minConf = 0.5,
     extensionStrategy = "ConfImprovementAgainstLastConfirmedExtension",
     loglevel = "WARNING", createHistorySlot = FALSE,
     timeExecution = FALSE)

Arguments

cbaRuleModel a CBARuleModel
datadf data frame with training data
extendType possible extend types - numericOnly or noExtend
defaultRuleOverlapPruning pruning removing rules made redundant by the default rule; possible values: noPruning, transactionBased, rangeBased, transactionBasedAsFirstStep
attributePruning remove redundant attributes
trim_literal_boundaries trimming of literal boundaries enabled
continuousPruning indicating continuous pruning is enabled
postpruning type of postpruning (none, cba - data coverage pruning, greedy - data coverage pruning stopping on first rule with total error worse than default)
fuzzification boolean indicating if fuzzification is enabled. Multi-rule classification model is produced if enabled. Fuzzification without annotation is not supported.

annotate boolean indicating if annotation with probability distributions is enabled, multi-rule classification model is produced if enabled.

ruleOutputPath path of file to which model will be saved. Must be set if multi rule classification is produced.

minImprovement parameter of QCBA extend procedure (used when extensionStrategy=ConfImprovementAgainstLastConfirmedExtension or ConfImprovementAgainstSeedRule)

minCondImprovement parameter of QCBA extend procedure

minConf minimum confidence to accept extension (used when extensionStrategy=MinConf)

extensionStrategy possible values: ConfImprovementAgainstLastConfirmedExtension, ConfImprovementAgainstSeedRule

loglevel logger level from java.util.logging

createHistorySlot creates a history slot on the resulting QCBARuleModel model, which contains an ordered list of extensions that were created on input rules during the extension process

timeExecution reports execution time of the extend step

Value

Object of class QCBARuleModel.

Examples

allData <- datasets::iris[sample(nrow(datasets::iris)),]
trainFold <- allData[1:100,]
testFold <- allData[101:nrow(datasets::iris),]
rmCBA <- cba(trainFold, classAtt="Species")
rmqCBA <- qcba(cbaRuleModel=rmCBA, datadf=trainFold)
print(rmqCBA@rules)

---

qcbaHumTemp Use the HumTemp dataset to test the one rule classification QCBA workflow.

Description

Learns a CBA classifier and performs all QCBA postprocessing steps.

Usage

qcbaHumTemp()

Value

QCBA model
qcbaIris

Use the iris dataset to test QCBA workflow.

Description

Learns a CBA classifier and performs all QCBA postprocessing steps.

Usage

qcbaIris()

Value

Accuracy.

qcbaIris2

Use the Iris dataset to test the experimental multi-rule QCBA workflow.

Description

Learns a CBA classifier, and then transforms it to a multirule classifier, including rule annotation and fuzzification. Applies the learnt model with rule mixture classification. The model is saved to a temporary file.

Usage

qcbaIris2()

Value

Accuracy.
Description

This class represents a QCBA rule-based classifier.

Slots

- **rules** object of class rules from arules package postprocessed by **qCBA**
- **history** extension history
- **classAtt** name of the target class attribute
- **attTypes** attribute types
- **rulePath** path to file with rules, has priority over the rules slot
- **ruleCount** number of rules

rcbaModel2CBARuleModel

**rcbaModel2arcCBARuleModel** Converts a model created by **rCBA** so that it can be passed to **qCBA**

Description

Creates instance of CBAmodel class from the **arc** package. Instance of CBAmodel can then be passed to **qcba**

Usage

```r
rcbaModel2CBARuleModel(rcbaModel, cutPoints, classAtt, rawDataset, attTypes)
```

Arguments

- **rcbaModel** object returned by **rCBA::build**
- **cutPoints** specification of cutpoints applied on the data before they were passed to **rCBA::build**
- **classAtt** the name of the class attribute
- **rawDataset** the raw data (before discretization). This dataset is used to guess attribute types if **attTypes** is not passed
- **attTypes** vector of attribute types of the original data. If set to null, you need to pass **rawDataset**. 
# this example takes about 10 seconds
if (!requireNamespace("rCBA", quietly = TRUE)) {
    message("Please install rCBA: install.packages('rCBA')")
} else {
    # This will run only outside a CRAN test, if the environment variable NOT_CRAN is set to true
    # This environment variable is set by devtools
    if (identical(Sys.getenv("NOT_CRAN"), "true")) {
        library(rCBA)
        message(packageVersion("rCBA"))
        discrModel <- discrNumeric(iris, "Species")
        irisDisc <- as.data.frame(lapply(discrModel$Disc.data, as.factor))
        rCBAModel <- rCBA::build(irisDisc, parallel=FALSE, sa=list(timeout=0.1))
        CBAmodel <- rcbaModel2CBARuleModel(rCBAModel, discrModel$cutp, "Species", iris)
        qCBAmodel <- qCBA(CBAmodel, iris)
        print(qCBAmodel@rules)
    }
}

sbrlModel2arcCBARuleModel

`sbrlModel2arcCBARuleModel` Converts a model created by `sbrl` so that it can be passed to `qCBA`

## Description

Creates instance of CBAmodel class from the `arc` package. Instance of CBAmodel can then be passed to `qCBA`.

## Usage

`sbrlModel2arcCBARuleModel(sbrl_model, cutPoints, rawDataset, classAtt, attTypes)`

## Arguments

- `sbrl_model`: object returned by `arulesCBA::CBA()`
- `cutPoints`: specification of cutpoints applied on the data before they were passed to `rCBA::build`
- `rawDataset`: the raw data (before discretization). This dataset is used to guess attribute types if `attTypes` is not passed
- `classAtt`: the name of the class attribute
- `attTypes`: vector of attribute types of the original data. If set to null, you need to pass `rawDataset`. 
Examples

if (! requireNamespace("rCBA", quietly = TRUE)) {
  message("Please install rCBA to allow for sbrl model conversion")
  return()
} else {
  library(rCBA)
}
if (! requireNamespace("sbrl", quietly = TRUE)) {
  message("Please install sbrl to allow for postprocessing of sbrl models")
  return()
} else {
  library(sbrl)
}
# sbrl handles only binary problems, iris has 3 target classes - remove one class
set.seed(111)
allData <- datasets::iris[sample(nrow(datasets::iris)),]
classToExclude<="versicolor"
allData <- allData[allData$Species!=classToExclude,]
# drop virginica level
allData$Species <- allData$Species[, drop=TRUE]
trainFold <- allData[1:50,]
testFold <- allData[51:nrow(allData),]
sbrlFixedLabel<="label"
origLabel<="Species"

orignames<-colnames(trainFold)
orignames[which(orignames == origLabel)]<-sbrlFixedLabel
colnames(trainFold)<-orignames
colnames(testFold)<-orignames

# to recode label to binary values:
# first create dict mapping from original distinct class values to 0,1
origval<-levels(as.factor(trainFold$label))
newval<-range(0,1)
dict<-data.frame(origval,newval)
# then apply dict to train and test fold
trainFold$label<-dict[match(trainFold$label, dict$origval), 2]
testFold$label<-dict[match(testFold$label, dict$origval), 2]

# discretize training data
trainFoldDiscTemp <- discrNumeric(trainFold, sbrlFixedLabel)
trainFoldDiscCutpoints <- trainFoldDiscTemp$cutp
trainFoldDisc <- as.data.frame(lapply(trainFoldDiscTemp$Disc.data, as.factor))

# discretize test data
testFoldDisc <- applyCuts(testFold, trainFoldDiscCutpoints, infinite_bounds=TRUE, labels=TRUE)

# learn sbrl model
sbrl_model <- sbrl(trainFoldDisc, iters=30000, pos_sign="0", neg_sign="1", rule_minlen=1, rule_maxlen=10,
minsupport_pos=0.10, minsupport_neg=0.10, 
lambda=10.0, eta=1.0, alpha=c(1,1), nchain=10)

# apply sbrl model on a test fold
yhat <- predict(sbrl_model, testFoldDisc)
yvals<- as.integer(yhat$V1>0.5)
sbrl_acc<-mean(as.integer(yvals == testFoldDisc$label))
message("SBRL RESULT")

rm_sbrl<-sbrlModel2arcCBARuleModel(sbrl_model,trainFoldDiscCutpoints,trainFold,sbrlFixedLabel)
message(paste("sbrl acc=",sbrl_acc,"sbrl rule count=",nrow(sbrl_model$rs), ",avg rule length", 
sum(rm_sbrl@rules@lhs@data)/length(rm_sbrl@rules)))

rmQCBA_sbrl <- qcba(cbaRuleModel=rm_sbrl,datadf=trainFold)
prediction <- predict(rmQCBA_sbrl,testFold)
acc_qcba_sbrl <- CBARuleModelAccuracy(prediction, testFold[[rmQCBA_sbrl@classAtt]])
if (! requireNamespace("stringr", quietly = TRUE)) {
  message("Please install stringr to compute average rule length for QCBA")
  avg_rule_length <- NA
} else {
  library(stringr)
  avg_rule_length <- (sum(unlist(lapply(rmQCBA_sbrl@rules[1],str_count,pattern=",")))+ 
  # assuming the last rule has antecedent length zero
  nrow(rmQCBA_sbrl@rules)-1)/nrow(rmQCBA_sbrl@rules)
}
message("QCBA RESULT")
rmQCBA_sbrl@rules

message(paste("QCBA after SBRL acc="acc_qcba_sbrl,"rule count=",
rmQCBA_sbrl@ruleCount, "avg rule length", avg_rule_length))
unlink("tdata_R.label") # delete temp files created by SBRL
unlink("tdata_R.out")
Index

arulesCBA2arcCBAModel, 2
CBARuleModel, 5
customCBARuleModel
   (customCBARuleModel-class), 3
customCBARuleModel-class, 3
iris, 7
mapDataTypes, 3
predict, 5
predict.qCBARuleModel, 4
qcba, 2, 4, 5, 8, 9
qcbaHumTemp, 6
qcbaIris, 7
qcbaIris2, 7
qCBARuleModel, 4, 6
qCBARuleModel (qCBARuleModel-class), 8
qCBARuleModel-class, 8
rcbaModel2CBARuleModel, 8
sbrlModel2arcCBARuleModel, 9