Package ‘qCBA’

April 3, 2025

Title Postprocessing of Rule Classification Models Learnt on Quantized Data

Version 1.0.2

Date 2025-04-03

Maintainer Tomáš Kliegr <kliegr@gmail.com>

Description Implements the Quantitative Classification-based on Association Rules (QCBA) algorithm (<doi:10.1007/s10489-022-04370-x>). QCBA postprocesses rule classification models making them typically smaller and in some cases more accurate. Supported are 'CBA' implementations from 'rCBA', 'arulesCBA' and 'arc' packages, and 'CPAR', 'CMAR', 'FOIL2' and 'PRM' implementations from 'arulesCBA' package and 'SBRL' implementation from the 'sbrl' package. The result of the post-processing is an ordered CBA-like rule list.

Depends R (>= 2.7.0), arules (>= 1.7-4), rJava (>= 0.5-0), arulesCBA (>= 1.2.5), arc (>= 1.3), methods

Suggests rCBA (>= 0.3.0), sbrl (>= 1.4.0)

SystemRequirements Java (>= 8)

URL `https://github.com/kliegr/QCBA`

BugReports `https://github.com/kliegr/QCBA/issues`

License GPL-3

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation no

Author Tomáš Kliegr [aut, cre]

Repository CRAN

Date/Publication 2025-04-03 10:10:02 UTC

Contents

`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`	aobject 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 {
## Not run:
 classAtt <- "Species"
 discrModel <- discrNumeric(iris, classAtt)
 irisDisc <- as.data.frame(lapply(discrModel$Disc.data, as.factor))
 arulesCBAModel <- arulesCBA::CBA(Species ~ ., data = irisDisc, supp = 0.1,
  conf=0.9)
 CBAmodel <- arulesCBA2arcCBAModel(arulesCBAModel, discrModel$cutp,  iris, classAtt)
 qCBAmodel <- qcba(cbaRuleModel=CBAmodel,datadf=iris)
 print(qCBAmodel@rules)

## End(Not run)
}
```

`benchmarkQCBA` *Learn and evaluate QCBA postprocessing on multiple rule learners. This can be, for example, used to automatically select the best model for a given use case based on a combined preference for accuracy and model size.*

Description

Learn multiple rule models using base rule induction algorithms from **arulesCBA** and apply QCBA to postprocess them.

Usage

```
benchmarkQCBA(
  train,
  test,
  classAtt,
  train_disc = NULL,
  test_disc = NULL,
  cutPoints = NULL,
  algs = c("CBA", "CMAR", "CPAR", "PRM", "FOIL2"),
  iterations = 2,
  rounding_places = 3,
  return_models = FALSE,
  debug_prints = FALSE,
  seed = 1,
  ...
)
```

Arguments

`train`	data frame with training data
`test`	data frame with testing data before postprocessing
`classAtt`	the name of the class attribute
`train_disc`	prediscretized training data
`test_disc`	prediscretized tet data
`cutPoints`	specification of cutpoints applied on the data (ignored if train_disc is null)
`algs`	vector with names of baseline rule learning algorithms. Names must correspond to function names from the **arulesCBA** library
`iterations`	number of executions over base learner, which is used for obtaining a more precise estimate of build time
`rounding_places`	statistics in the resulting dataframe will be rounded to specified number of decimal places
`return_models`	boolean indicating if also learnt rule lists (baseline and postprocessed) should be included in model output
`debug_prints`	print debug information such as rule lists
`seed`	random seed value
`...`	Parameters for base learners, the name of the argument is the base learner (one of ‘algs‘ values) and value is a list of parameters to pass. To specify parameters for QCBA pass "QCBA". See also Example 3.

Value

Outputs a dataframe with evaluation metrics and if ‘return_models==TRUE‘ also the induced baseline and QCBA models (see also Example 3). Included metrics in the dataframe with statistics: **accuracy**: percentage of correct predictions in the test set **rulecount**: number of rules in the rule list. Note that for QCBA the count includes the default rule (rule with empty antecedent), while for base learners this rule may not be included (depending on the base learner) **modelsize**: total number of conditions in the antecedents of all rules in the model **buildtime**: learning time for inference of the model. In case of QCBA, this excludes time for the induction of the base learner

See Also

[qcba()] which this function wraps.

Examples

```
# EXAMPLE 1: pass train and test folds, induce multiple base rule learners,
# postprocess each with QCBA and return benchmarking results.
## Not run:
if (identical(Sys.getenv("NOT_CRAN"), "true")) {
# Define input dataset and target variable
df_all <-datasets::iris
classAtt <- "Species"
# Create train/test partition using built-in R functions
tot_rows<-nrow(df_all)
train_proportion<-2/3
df_all <- df_all[sample(tot_rows),]
trainFold <- df_all[1:(train_proportion*tot_rows),]
testFold <- df_all[(1+train_proportion*tot_rows):tot_rows,]
# learn with default metaparameter values
stats<-benchmarkQCBA(trainFold,testFold,classAtt)
print(stats)
# print relative change of QCBA results over baseline algorithms
print(stats[,6:10]/stats[,0:5]-1)
}
## End(Not run)
# EXAMPLE 2: As Example 1 but data are discretizated externally
# Discretize numerical predictors using built-in discretization
# This performs supervised, entropy-based discretization (Fayyad and Irani, 1993)
# of all numerical predictor variables with 3 or more distinct numerical values
# This example could run for more than 5 seconds
## Not run:
if (identical(Sys.getenv("NOT_CRAN"), "true")) {
  discrModel <- discrNumeric(trainFold, classAtt)
  train_disc <- as.data.frame(lapply(discrModel$Disc.data, as.factor))
  test_disc <- applyCuts(testFold, discrModel$cutp, infinite_bounds=TRUE, labels=TRUE)
  stats<-benchmarkQCBA(trainFold,testFold,classAtt,train_disc,test_disc,discrModel$cutp)
  print(stats)
}
## End(Not run)
# EXAMPLE 3: pass custom metaparameters to selected base rule learner,
# then postprocess with QCBA, evaluate, and return both models
# This example could run for more than 5 seconds
if (identical(Sys.getenv("NOT_CRAN"), "true")) {
# use only CBA as a base learner, return rule lists.
## Not run:
 output<-benchmarkQCBA(trainFold,testFold,classAtt,train_disc,test_disc,discrModel$cutp,
                    CBA=list("support"=0.05,"confidence"=0.5),algs = c("CPAR"),
                    return_models=TRUE)
  message("Evaluation statistics")
  print(output$stats)
  message("CPAR model")
  inspect(output$CPAR[[1]])
  message("QCBA model")
  print(output$CPAR_QCBA[[1]])

## End(Not run)
}
```

`customCBARuleModel-class` *customCBARuleModel*

Description

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

Slots

`rules`	dataframe with rules
`cutp`	list of cutpoints
`classAtt`	name of the target class attribute
`attTypes`	attribute types

`getConfVectorForROC` *Returns vector with confidences for the positive class (useful for ROC or AUC computation)*

Description

Methods for computing ROC curves require a vector of confidences of the positive class, while in qCBA, the confidence returned by predict.qCBARuleModel with outputProbabilies = TRUE returns confidence for the predicted class. This method converts the values to confidences for the positive class

Usage

```
getConfVectorForROC(confidences, predictedClass, positiveClass)
```

Arguments

`confidences`	Vector of confidences
`predictedClass`	Vector with predicted classes
`positiveClass`	Positive class (String)

Value

Vector of confidence values

Examples

```
predictedClass = c("setosa","virginica")
confidences = c(0.9,0.6)
baseClass="setosa"
getConfVectorForROC(confidences,predictedClass,baseClass)
```

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

```
## S3 method for class 'qCBARuleModel'
predict(
  object,
  newdata,
  testingType,
  loglevel = "WARNING",
  outputFiringRuleIDs = FALSE,
  outputConfidenceScores = FALSE,
  confScoreType = "ordered",
  positiveClass = NULL,
  ...
)
```

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).
`outputConfidenceScores`	if set to TRUE, instead of predictions, the function will return confidences of the firing rule
`confScoreType`	applicable only if 'outputConfidenceScores=TRUE', possible values 'ordered' for confidence computed only for training instances reaching this rule, or 'global' for standard rule confidence computed from the complete training data
`positiveClass`	This setting is only used if 'outputConfidenceScores=TRUE'. It should be used only for binary problems. In this case, the confidence values are recalculated so that these are not confidence values of the predicted class (default behaviour of 'outputConfidenceScores=TRUE') but rather confidence values associated with the class designated as positive
`...`	other arguments (currently not used)

Value

vector with predictions.

See Also

qcba

Examples

```
## Not run:
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")
message(rmqCBA@rules[firingRuleIDs[2],1])
message("The second instance is")
message(testFold[2,])

## End(Not run)
```

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,
  computeOrderedStats = TRUE
)
```

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=ConfImprovementAgainstLast` or `ConfImprovementAgainstSeedRule`)
`minCondImprovement`	parameter of qCBA extend procedure
`minConf`	minimum confidence to accept extension (used when extensionStrategy=MinConf)
`extensionStrategy`	possible values: `ConfImprovementAgainstLastConfirmedExtension`, `ConfImprovementAgainstSee`
`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
`computeOrderedStats`	appends orderedConf and orderedSupp quality metrics to the resulting dataframe. Setting this parameter to FALSE will reduce the training time.

Value

Object of class qCBARuleModel.

Examples

```
## Not run:
allData <- datasets::iris[sample(nrow(datasets::iris)),]
trainFold <- allData[1:100,]
rmCBA <- cba(trainFold, classAtt="Species")
rmqCBA <- qcba(cbaRuleModel=rmCBA,datadf=trainFold)
print(rmqCBA@rules)

## End(Not run)
```

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 the test QCBA workflow.*

Description

Learns a CBA classifier and performs all QCBA postprocessing steps This function uses a package-specific seed option, which can be overridden with ‘options(qcba.seed = 42)‘, where 42 is an example seed value

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. This function uses a package-specific seed option, which can be overridden with ‘options(qcba.seed = 42)‘, where 42 is an example seed value

Usage

```
qcbaIris2()
```

Value

Accuracy.

`qCBARuleModel-class` *QCBA Rule Model*

Description

This class represents a QCBA rule-based classifier.

Usage

```
## S4 method for signature 'qCBARuleModel'
show(object)
```

Arguments

`object`	An object of class `qCBARuleModel`

Details

qCBARuleModel

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

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

Arguments

`rcbaModel`	object returned by rCBA::build
`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

```
# 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")) {
## Not run:
 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=0.01))
 CBAmodel <- rcbaModel2CBARuleModel(rCBAmodel,discrModel$cutp,iris,"Species")
 qCBAmodel <- qcba(CBAmodel,iris)
 print(qCBAmodel@rules)

## End(Not run)
  }
}
```

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 if (! requireNamespace("sbrl", quietly = TRUE)) {
  message("Please install sbrl to allow for postprocessing of sbrl models")
} 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(sbrl)
  library(rCBA)
```

```
# sbrl handles only binary problems, iris has 3 target classes - remove one class
set.seed(getOption("qcba.seed"))
allData <- datasets::iris[sample(nrow(datasets::iris)),]
classToExclude<-"versicolor"
allData <- allData[allData$Species!=classToExclude, ]
# drop the removed level
allData$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)
# SBRL 1.4 crashes if features contain a space
# even if these features are converted to factors,
# to circumvent this, it is necessary to replace spaces
trainFoldDisc <- as.data.frame(lapply(trainFoldDisc, function(x) gsub(" ", "", as.character(x))))
for (name in names(trainFoldDisc)) {trainFoldDisc[name] <- as.factor(trainFoldDisc[,name])}
# learn sbrl model, rule_minlen is increased to demonstrate the effect of postprocessing
sbrl_model <- sbrl(trainFoldDisc, iters=20000, pos_sign="0",
 neg_sign="1", rule_minlen=3, rule_maxlen=5, minsupport_pos=0.05, minsupport_neg=0.05,
 lambda=20.0, eta=5.0, nchain=25)
# 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")
message(sbrl_model)
rm_sbrl<-sbrlModel2arcCBARuleModel(sbrl_model,trainFoldDiscCutpoints,trainFold,sbrlFixedLabel)
message(paste("sbrl acc=",sbrl_acc,", sbrl rule count=",nrow(sbrl_model$rs), ",
avg condition count (incl. default rule)",
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]])
    avg_rule_length <- rmQCBA_sbrl@rules$condition_count/nrow(rmQCBA_sbrl@rules)
    message("RESULT of QCBA postprocessing of SBRL")
    message(rmQCBA_sbrl@rules)
    message(paste("QCBA after SBRL acc=",acc_qcba_sbrl,", rule count=",
    rmQCBA_sbrl@ruleCount, ", avg condition count (incl. default rule)",  avg_rule_length))
    unlink("tdata_R.label") # delete temp files created by SBRL
    unlink("tdata_R.out")
  }
}
```

Index