qCBA: An R Package for Postoptimization of Rule Models Learnt on Quantized Data

Tomas Kliegr

Abstract A popular approach to building rule models is association rule classification. However, these often produce larger models than most other rule learners, impeding the comprehensibility of the created classifiers. Also, these algorithms decouple discretization from model learning, often leading to a loss of predictive performance. This package presents an implementation of Quantitative Classification based on Associations (QCBA), which is a collection of postprocessing algorithms for rule models built over discretized data. The QCBA method improves the fit of the bins originally produced by discretization and performs additional pruning, resulting in models that are typically smaller and often more accurate. The qCBA package supports models created with multiple packages for rule-based classification available in CRAN, including arc, arulesCBA, rCBA and sbrl.

1 Introduction

There is a resurgence of interest in interpretable machine learning models, with rule learning providing an appealing combination of intrinsic comprehensibility, as humans are naturally used to working with rules, with well-documented predictive performance and scalability. Association rule classification (ARC) is a subclass of rule-learning algorithms that can quickly generate a large number of candidate rules, a subset of which is subsequently chosen for the final classifier. The first, and with at least three packages (Hahsler et al., 2019), the most popular such algorithm was Classification Based on Associations (CBA) (Liu et al., 1998). There are also multiple newer approaches, such as SBRL (Yang et al., 2017) or RCAR (Azmi and Berrado, 2020), both available in R (Yang et al., 2024; Michael Hahsler, 2024).

A major limitation of ARC approaches is that they typically trade off the ability to process numerical data for the speed of generation. On the input, these approaches require categorical data. If there are numerical attributes, these need to be converted to categories, typically through some discretization (quantization) approach, such as MDLP (Fayyad and Irani, 1993). Association rule learning then operates on prediscretized datasets, which results in the loss of predictive performance and larger rule sets.

The *Quantitative Classification Based on Associations* method (QCBA) (Kliegr and Izquierdo, 2023) is a collection of several algorithms that postoptimize rule-based classifiers learnt on prediscretized data with respect to the original raw dataset with numerical attributes. As was experimentally shown in Kliegr and Izquierdo (2023), this makes the models often more accurate and consistently smaller and thus more interpretable.

This paper presents the **qCBA** R package, which implements QCBA and is available on CRAN. The **qCBA** R package was initially developed to postprocess results of CBA implementations, as these were the most common rule learning systems in R, but it can now also handle results of other rule learning approaches such as SBRL. The three CBA implementations in CRAN – **rCBA** (Kuchař and Kliegr, 2019), **arc** (Kliegr, 2016) and **arulesCBA** (Michael Hahsler, 2024) introduced in Hahsler et al. (2019) rely on the fast and proven **arules** package (Hahsler et al., 2011) to mine association rules, which is also the main dependency of the **qCBA** R package.

2 Primer on building rule-based classifiers for QCBA in R

This primer will show how to use QCBA with CBA as the base rule learner. Out of the rule learners supported by QCBA, CBA is the most widely supported in CRAN and also scientifically most cited (as of writing). This primer is standalone and intends to show the main concepts through code-based examples. However, it uses the same dataset and setting as the going example in the open-access publication (Kliegr and Izquierdo, 2023), which contains graphical illustrations as well as formal definitions of the algorithms (as opposed to R-code examples here).

2.1 Brief introduction to association rule classification

Before we present the details of the QCBA algorithm, we start by covering the foundational methods of association rule learning and classification.

Input data Association rules are historically mined on *transactions*, which is also the format used by the most commonly used R package **arules**. A standard data table (data frame) can be converted to a transaction matrix. This can be viewed as a binary incidence matrix, with one dummy variable for each attribute-value pair (called an *item*). In the case of numerical variables, discretization (quantization) is a standard preprocessing step. It is required to ensure that the search for rules is fast and the conditions in the discovered rules are sufficiently broad.

Association rules Algorithms such as Apriori (Agrawal and Srikant, 1994) are used for association rule mining. An association rule has the form $r : antecedent \rightarrow consequent$, where both *antecedent* and *consequent* are sets of items. The **arules** package, which is the most popular Apriori implementation in CRAN, calls the antecedent the left-hand side of the rule (*lhs*) and the consequent the right-hand side (*rhs*). Each set is interpreted as a conjunction of conditions corresponding to individual items. When all these conditions are met, then the rule predicts that its consequent is true for a given input data point. Formally, we say that a rule *covers* a transaction if all items in the rule's antecedent are contained in the transaction. A rule *correctly classifies* the transaction if the transaction is covered and, at the same time, the items in the rule consequent are contained in the transaction.

Rule interest measures Each rule is associated with some quality metrics (also called rule interest measures). The two most important ones are the confidence and support of rule r. Confidence is calculated as $conf(r) = a/(a+b)$, where a is the number of transactions matching both the antecedent and consequent and b is the number of transactions matching the antecedent but not the consequent. Support is calculated either as a (absolute support) or a divided by the total number of transactions (relative support). In the association rule generation step, confidence and support are used as constraints in association rule learning. Another important constraint to prevent a combinatorial explosion is a restriction on the length of the rule (`minLen` and `maxLen` parameters in **arules**), defined as the threshold on the minimum and maximum count of items in the rule antecedent and consequent.

Association Rule Classification models ARC algorithms process candidate association rules into a classifier. An input to the ARC algorithm is a set of pre-mined *class association rules* (CARs). A class association rule is an association rule that contains exactly one item corresponding to one of the target classes in the consequent. The classifier-building process typically amounts to a selection of a subset of CARs (Vanhoof and Depaire, 2010). Some ARC algorithms, such as CBA, use rule interest measures for this: rules are first ordered and then some of them are removed using *data coverage pruning* and *default rule pruning* (step 2 and step 3 of the CBA-CB algorithm as described in Liu et al. (1998). Some ARC algorithms also include a rule with an empty antecedent (called *default rule*) at the end of the rule list to ensure that the classifier can always provide a prediction even if there is no specific rule matching that particular transaction.

Types of ARC models CBA produces *rule lists*: the rule with the highest priority in the pruned rule list is used for classification. Other recent algorithms producing rule lists include SBRL (Yang et al., 2024). The other common ARC approach is based on *rule sets*, where rules are not ordered, and multiple rules can be used for classification, e.g., through voting. This second group includes algorithms such as CMAR (Li et al., 2001) or CPAR (Yin and Han, 2003).

Postprocessing with Quantitative CBA QCBA is a postprocessing algorithm for ARC models built over quantized data. On the input, it can take both rule lists or rule sets.

However, it always outputs a rule list. While association rule learning often faces issues with a combinatorial explosion (Kliegr and Kuchar, 2019), the postprocessing by QCBA is performed on a relatively small number of rules that passed the pruning steps within the specific ARC algorithm. QCBA is a modular approach with six steps that can be performed independently of each other. The only exception is the initial refit tuning step, which processes all items in a given rule that are the result of quantization. QCBA adjusts the item boundaries so that they correspond to actual values appearing in the original training data before discretization. Benchmarks in Kliegr and Izquierdo (2023) have shown that, on average, the postprocessing with QCBA took a similar time as building the input CBA model. The most expensive step can be extension, with its complexity depending on the number of unique numerical values.

Comparison with decision tree induction A common question is the relationship between association rule classifiers and decision trees. Trees can be decomposed into rules that are similar to association rules, as each path from the root to the leaf corresponds to one rule. However, individual trees in algorithms such as C4.5 (Quinlan, 1993) are built in such a way that the resulting rules are non-overlapping, while association rule learning outputs overlapping rules. Algorithms such as Apriori will output all rules that are valid in the data, given the user-specified thresholds. In contrast, decision tree induction algorithms use a heuristic, such as information gain, which results in a large number of otherwise valid patterns being skipped as rules prioritize splits that maximize the chosen heuristic.

2.2 Example

Dataset

Let's look at the humtemp synthetic data set, which we will be using throughout this tutorial and is bundled with the **arcPackage**. There are two quantitative explanatory attributes (Temperature and Humidity). The target attribute is preference (subjective comfort level).

The first six rows of humtemp obtained with head(humtemp):

```
##   Temperature Humidity Class
## 1          45       33     2
## 2          27       29     3
## 3          40       48     2
## 4          40       65     1
## 5          38       82     1
## 6          37       30     3
```

Quantization

The essence of QCBA is the optimization of literals (conditions) created over numerical attributes in rules. QCBA thus needs to translate back the bins created during the discretization to the continuous space.

For clarity, we will perform the quantization manually using equidistant binning and user-defined cut points. An alternative approach using automatic supervised discretization is shown in Section 2.4.

```
library(qCBA)

temp_breaks <- seq(from = 15, to = 45, by = 5)
# Another possibility with user-defined cutpoints
hum_breaks <- c(0, 40, 60, 80, 100)

data_discr <- arc::applyCuts(
  df = humtemp,
  cutp = list(temp_breaks, hum_breaks, NULL),
  infinite_bounds = TRUE,
  labels = TRUE
)
head(data_discr)
```

The result of quantization is:

```
##   Temperature Humidity Class
## 1     (40;45]   (0;40]     2
## 2     (25;30]   (0;40]     3
## 3     (35;40]  (40;60]     2
## 4     (35;40]  (60;80]     1
## 5     (35;40] (80;100]     1
## 6     (35;40]   (0;40]     3
```

The purpose of the `applyCuts()` function is to ensure that within intervals, a semicolon is used as a separator instead of the more common comma. A semicolon is used as the standard interval separator in some countries, such as Czechia. However, the main reason is that a comma is already used within rules for another purpose – to separate conditions. The use of a different separator fosters unambiguity, for example, in situations when a rule set aimed to be optimized is read from a file.

Discovery of candidate class association rules

ARC algorithms typically first generate a large number of association rules or frequent itemsets. Typically, this step is handled internally by the ARC library (as shown in Section Demonstration of individual QCBA optimization steps).

For better clarity, in the example below, the list of CARs is generated by manually invoking the apriori algorithm.

```
txns <- as(data_discr, "transactions")
appearance <- list(rhs = c("Class=1", "Class=2", "Class=3", "Class=4"))
rules <- arules::apriori(
  data = txns,
  parameter = list(
    confidence = 0.5,
    support = 3 / nrow(data_discr),
    minlen = 1,
    maxlen = 3
  ),
  appearance = appearance
)
```

The first line converts the input data into *items*, attribute-value pairs such as `Temperature=(40;45]` or `Class=3`. The `appearance` defines which items can appear in the consequent of the rules (right-hand side, *rhs*). This data format is required for association rule learning. On the second line, there are several standard additional parameters typically used for the extraction of association rules. The confidence threshold of 0.5 and support of 1% is recommended (Liu et al., 1998). However, since the `humtemp` dataset has fewer than 100 rows, the support threshold would correspond to the support of just 1 transaction, which would miss the purpose of this threshold to address overfitting by eliminating rules that are backed by only a small number of instances. We, therefore, set the minimum support threshold to 3 transactions (expressed as a percentage in the code snippet). The `minlen` and `maxlen` express that rules must contain at most two items in the condition part (antecedent).

The discovered rules, shown with `inspect(rules)`, are shown below:

```
lhs                                       rhs       support    confidence coverage  lift     #
{Humidity=(80;100]}                    => {Class=1} 0.11111111 0.8000000  0.1388889 3.600000 4
{Temperature=(15;20]}                  => {Class=2} 0.11111111 0.5714286  0.1944444 2.057143 4
{Temperature=(30;35]}                  => {Class=4} 0.13888889 0.6250000  0.2222222 2.045455 5
{Temperature=(25;30]}                  => {Class=4} 0.13888889 0.5000000  0.2777778 1.636364 5
{Temperature=(25;30], Humidity=(40;60]} => {Class=4} 0.08333333 0.6000000  0.1388889 1.963636 3
```

In this listing, coverage and lift, as defined in the Hahsler et al. (2007) documentation, are additional rule quality measures not used by the **qCBA** package. Count (abbreviated with # in the listing) corresponds to the support of the rule represented as an integer value rather than a proportion.

Learn classifier from candidate CARs

Out of the five discovered rules, we create a CBA classifier. The following lists the conceptual steps performed by CBA to transform the input rule list into a CBA model:

1. Rule *precedence* is established: rules are sorted according to confidence, support and length.
2. Rules are subject to *pruning*
   - *data coverage pruning*: the algorithm iterates through the rules in the order of precedence removing any rule which does not correctly classify at least one instance. If the rule correctly classifies at least one instance, it is retained and the instances removed (only for the purpose of subsequent steps of data coverage pruning).
   - *default rule pruning*: the algorithm iterates through the rules in the sort order, and cuts off the list once keeping the current rule would result in worse accuracy of the model than if a default rule was inserted at the place of the current rule and the rules below it were removed.
3. *Default rule* is inserted at the bottom of the list.

To supply our own list of candidate rules instead of using one generated with cba(), we will call:

```
classAtt <- "Class"
rmCBA <- cba_manual(
  datadf_raw = humtemp,
  rules = rules,
  txns = txns,
  rhs = appearance$rhs,
  classAtt = classAtt,
  cutp = list()
)
inspect(rmCBA@rules)
```

The reason why we invoke cba_manual() from the arc package is that cba_manual() instead of cba() allows us to supply a custom list of rules from which the CBA model will be built. This function would also do the quantization, but since we already did this as part of the preprocessing, we use cutp = list() to express that no cutpoints are specified.

In this toy example, the CBA model, which could be displayed through the function call inspect(rmCBA@rules), is almost identical to the candidate list of rules shown above. The main difference is the reordering of rules by confidence and support (higher is better) and the addition of the *default rule* – a rule with an empty antecedent to the end. Note that for brevity, the conditions in the rules in the printout below were replaced by {...} as they are the same as in the printout on the previous page (although mind the different order of rules). The values of support, confidence, coverage and lift are also the same and were omitted.

```
    lhs       rhs        {...}  count lhs_length orderedConf orderedSupp cumulativeConf
[1] {...} => {Class=1}   {...}  4     1          0.8000000   4           0.8000000
[2] {...} => {Class=4}   {...}  5     1          0.7142857   5           0.7500000
[3] {...} => {Class=4}   {...}  3     2          0.6000000   3           0.7058824
[4] {...} => {Class=2}   {...}  4     1          0.5000000   3           0.6521739
[5] {...} => {Class=4}   {...}  5     1          0.5000000   2           0.6296296
[6] {}    => {Class=2}   {...}  0     0          0.5555556   5           0.6111111
```

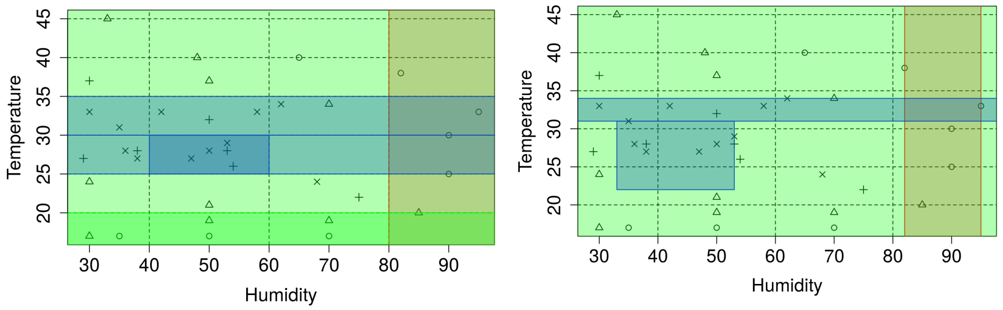

Figure 1: Illustration of postpruning algorithm (HumTemp dataset). Left: CBA model; right: QCBA model.

The default rule ensures that the rule list covers every possible instance. The rule list is visualized in Figure 1, where the green background corresponds to Class 2 predicted by the default rule. The CBA output contains several additional statistics for each rule. The *ordered* versions of confidence and support are computed only from those training instances reaching the given rule. For the first rule, the ordered confidence is identical to standard confidence. The ordered support is also semantically the same but is expressed as an absolute count rather than a proportion. To compute these values for the subsequent rules, instances covered by rules higher in the list have been removed. The cumulative confidence is an experimental measure described in **arc** documentation.

Applying the classifier

The toy example does not contain enough data for a meaningfully large train/test split. Therefore, we will evaluate the training data. The accuracy of the CBA model on training data:

```
prediction_cba <- predict(rmCBA, data_discr)
acc_cba <- CBARuleModelAccuracy(
  prediction = prediction_cba,
  groundtruth = data_discr[[classAtt]]
)
```

The accuracy is 0.61, and the contents of `prediction_cba` are the predicted values of comfort:

```
 [1] 2 4 2 2 1 2 2 4 4 4 4 4 4 1 4 1 4 4 4 4 4 4 4 4 1 2 2 2 2 1 2 2 2 2 2 2
Levels: 1 2 4
```

Explaining the prediction

The `predict()` function from the **arc** library allows for additional output to enhance the explainability of the result. By setting `outputFiringRuleIDs=TRUE` we can obtain the ID of a particular rule that was used to classify each instance in the passed dataset.

```
ruleIDs <- predict(rmCBA, data_discr, outputFiringRuleIDs = TRUE)
```

For example, we may now explain the classification of the first row of `data_discr`, which is, for convenience, reproduced below:

```
##   Temperature Humidity Class
## 1     (40;45]   (0;40]     2
```

To do so, we invoke:

```
inspect(rmCBA@rules[ruleIDs[1]])
```

This returns the default rule (number 6). The reason is that the values in this instance are out of bounds of the conditions in all other rules. Now, we have a rule list ready for postoptimization with QCBA.

Postprocessing with QCBA

By working with the original continuous data, the QCBA algorithm can improve the fit of the rules and consequently reduce their count.

We will use the rmCBA model built previously:

```
rmqCBA <- qcba(cbaRuleModel = rmCBA, datadf = humtemp)
```

This ran QCBA with the default set of optimization steps enabled, which correspond to the best-performing configuration #5 from (Kliegr and Izquierdo, 2023).

The resulting rule list is

```
   rules                                             support   confidence #
1 {Humidity=[82;95]} => {Class=1}                    0.1111111  0.8000000  1
2 {Temperature=[22;31],Humidity=[33;53]} => {Class=4} 0.1666667  0.7500000  2
3 {Temperature=[31;34]} => {Class=4}                 0.1388889  0.6250000  1
4 {} => {Class=2}                                    0.2777778  0.2777778  0
```

Note that the 'condition_count' column was abbreviated as # in the listing and the orderedConf and orderedSupp columns were omitted for brevity.

Figure 1 (right) shows the QCBA model. Compared to the CBA model in Figure 1 (left), QCBA removed two rules and refined the boundaries of the remaining rules.

Predictive performance is computed in the same way as for CBA:

```
prediction <- predict(rmqCBA, humtemp)
acc <- CBARuleModelAccuracy(prediction, humtemp[[rmqCBA@classAtt]])
```

The accuracy is unchanged at 0.61, but we got a smaller model. Similarly, as with CBA, we could use the argument *outputFiringRuleIDs*.

Note that the QCBA algorithm does not introduce any mandatory thresholds or meta-parameters for the user to set or optimize, although it does allow the user to enable or disable the individual optimizations, as shown in the next section.

3 Detailed description of package qCBA with examples in R

3.1 Overview of qcba() arguments

To build a model, qcba() needs a set of rules. As a second mandatory argument, the qcba() function takes the *raw* data frame. This can contain nominal as well as numerical columns. The remaining arguments are optional. The most important ones relating to the optimizations performed by QCBA are described in the following two subsections.

This rule model can be either the instance of customCBARuleModel or CBARuleModel class. The difference is that in the *rules* slot, in the former class, rules are represented as string objects in a data frame. This universal data frame format is convenient for loading rules from other sources or R packages that export rules as strings. The latter uses an instance of rules class from **arules**, which is more efficient, especially when the number of rules or items is larger. An important slot shared between both classes is cutp, which contains information on the cutpoints used to quantize the data on which the rules were learnt.

3.2 Optimizations on individual rules

Optimizations can be divided into two groups depending on whether they are performed on individual rules or on the entire model. We first describe the first group. Since these operations are independent of the other rules, they can also be parallelized.

Refitting rules. This step processes all items derived with quantization in the antecedent of a given rule. These items have boundaries that stick to a grid that corresponds to the result of discretization. The grid used by QCBA corresponds to all unique values appearing in the training data. *This is the only mandatory step.*

Attribute pruning (`attributePruning`). Attribute pruning is a step in QCBA that evaluates if the items are needed for each rule and item in its antecedent. The item is removed if a rule created without the item has at least the confidence of the original rule. *Enabled (`TRUE`) by default.*

Trimming (`trim_literal_boundaries`). Boundary parts of intervals into which no instances correctly classified by the rule fall are removed. *Enabled (`TRUE`) by default.*

Extension (`extendType`). The ranges of intervals in the antecedent of each rule are attempted to be enlarged. Currently, only extension on numerical attributes is supported. By default, the extension is accepted if it does not decrease rule confidence, but this behaviour can be controlled by setting the `minImprovement` parameter (default is 0). To overcome local minima, the extension process can provisionally accept a drop in confidence in the intermediate result of the extension. How much the confidence can temporarily decrease for the extension process not to stop is controlled by `minCondImprovement`. In the current version, the extension applies only to numerical attributes (set `extendType="numericOnly"` to enable, this is also the default value). In future versions, other extend types may be added.

3.3 Optimizations on rule list

The second group of optimizations aims at removing rules, considering the context of other rules in the list.

Data coverage pruning (`postpruning`). The purpose of this step is to remove rules that were made redundant by the previous QCBA optimizations. Possible values are `none`, `cba` and `greedy`. The `cba` option is identical to CBA's data coverage pruning: a rule is removed if it does not correctly classify any transaction in the training data after all transactions covered by retained rules with higher precedence were removed. The `greedy` option is an experimental modification of data coverage pruning described in the **qCBA** documentation. *Enabled by default* (the default value is `postpruning=cba`).

Default rule overlap pruning (`defaultRuleOverlapPruning`). Let R_p denote the set of rules that classify into the same class as the default rule $r_d : \{\} \rightarrow cons_1$. To determine if a pruning candidate, denoted as $r_p \in R_p : ant \rightarrow cons_1$, can be removed, all rules with lower precedence that have a nonempty antecedent and a different consequent $cons_2$ ($cons_1 \neq cons_2$) are identified. Let's denote the set of these rules as R_c. If the antecedents of all rules in R_c do not cover any of the transactions covered by r_p in the training data, r_p is removed. The removal of r_p will not affect the classification of training data, since the instances originally covered by $r_p : ant \rightarrow cons_1$ will be classified to the same class by $r_d : \{\} \rightarrow cons_1$. This is called the transaction-based version. In the alternative range-based version, the checks on rules in R_p involve checking the boundaries of intervals rather than overlap in matched transactions. This parameter has three possible values: `transactionBased`, `rangeBased` and `none`. According to analysis and benchmarks in (Kliegr and Izquierdo, 2023), the transaction-based version removes more rules than the range-based version, although it can sometimes affect predictive performance on unseen data. *Transaction-based pruning is the default.*

3.4 Effects on classification performance and model size

An overview of observed properties of the QCBA steps is present in Table 1. The entries denote the effect of applying the algorithm specified in the first column on the input rule list: $\geq$ denotes that the value of the given metric will increase or will not change, = the value will not change, $\leq$ decrease or will not change, *na* can increase, decrease or will not change. For example, applying the refit algorithm on a rule can have the following effects according to the table: the density of the rule will improve or remain the same: the (+) symbol in the table denotes that an increase in that value is considered a favourable property, and (-) as negative. Rule confidence (*conf*), rule support (*supp*), rule length (*length*) will remain unaffected. Considering the entire rule list, the refit operation will not affect the rule count or accuracy on training data (acc_{train}). However, the accuracy on unseen data (acc_{test}) may change.

algorithm	rule (local classifier)			rule list		
dataset	conf	supp	length	acc_{train}	acc_{test}	rule count
refit	=	=	=	=	*na*	=
literal pruning	$\geq$	$\geq$ (+)	$\leq$ (+)	*na*	*na*	$\leq^*$ (+)
trimming	$\geq$ (+)	=	=	$\geq$ (+)	*na*	=
extension	$\geq$ (+)	$\geq$ (+)	=	*na*	*na*	=
postpruning	na	na	na	$\geq$	*na*	$\leq$ (+)
drop - trans.	na	na	na	=	*na*	$\leq$ (+)
drop - range	na	na	na	=	=	$\leq$ (+)

Table 1: Hypothesized properties of the proposed rule tuning algorithms. *drop* denotes default rule pruning (*trans.* transaction-based; *range* range-based). * the number of rules is not directly reduced, but literal pruning can make a rule redundant (identical to another rule). The value *na* expresses that there is no unanimous effect of the preprocessing algorithm on the quality measure.

3.5 Handling missing data

Association rule learning approaches are resilient to the presence of missing data in the input data frame. The reason is that rows are viewed as transactions and combinations of column names and their values as items. A missing value (`NA`) in a given column is then interpreted as an item not present in a given transaction and skipped when a data frame is converted to the `transactions` data structure from the **arules** using the `as()` function, which will result in NA values not being present in learnt rules. Note that `qcba()` treats an empty string in the input dataframe in the same way as the `NA` value along with a dataframe containing `NA` values: the `NA` value is first converted to an empty string, represented using `.jarray()` from **rJava** and then passed to the Java-based core of **qCBA**, where it is treated as a `Float.NaN` value and also generally skipped.

3.6 Computational costs for large datasets

The individual postprocessing operations have distinct computational costs. These depend on several main factors: the number of rows and columns of the input data, the number of unique numerical values, and the number of input rules. A detailed experimental study of the influence of these factors is presented in Kliegr and Izquierdo (2023). This was performed on subsets of the KDD'99 Anomaly Detection dataset, with the maximum dataset size being processed using all QCBA optimizations, reaching about 40,000 rows and about the same number of unique numerical values. The results show that the most computationally intensive step is the extension step. For large datasets, the user may, therefore, consider disabling it or tuning its `minCI` parameter, which can influence the runtime. An important factor is also the number of input rules for optimization. This is often related to the chosen base learning algorithm. For example, CPAR tends to produce a large number of rules while SBRL will output condensed rule models. For details, please refer to Kliegr and Izquierdo (2023).

4 Demonstration of individual QCBA optimization steps

Compared to the example in Subsection Example, this part will use the larger iris dataset, which allows for the demonstration of all QCBA steps. To show QCBA with a different base rule learner than CBA from the **arc** package used in the previous section, we will use CPAR from the **arulesCBA** package.

4.1 Data

We use the `iris` dataset from the **datasets** R package. The dataset was shuffled and randomly split into a train set (100 rows) and a test set (50 rows). The data was then automatically discretized using the MDLP algorithm wrapped by `arc::discrNumeric()` and originally available from the R **discretization** library.

```
library(arulesCBA)  #version 1.2.7
set.seed(12) # Chosen for demonstration purposes
allDataShuffled <- datasets::iris[sample(nrow(datasets::iris)), ]
trainFold <- allDataShuffled[1:100, ]
testFold <- allDataShuffled[101:nrow(datasets::iris), ]
classAtt <- "Species"

discrModel <- discrNumeric(df = trainFold, classAtt = classAtt)
train_disc <- as.data.frame(lapply(discrModel$Disc.data, as.factor))
cutPoints <- discrModel$cutp
test_disc <- applyCuts(
  testFold,
  cutPoints,
  infinite_bounds = TRUE,
  labels = TRUE
)
y_true <- testFold[[classAtt]]
```

4.2 Learn base ARC model with CPAR

In the following, we learn an ARC model using the CPAR (Classification based on Predictive Association Rules) algorithm using its default settings.

```
rmBASE <- CPAR(train_disc, formula = as.formula(paste(classAtt, "~ .")))
predictionBASE <- predict(rmBASE, test_disc)
inspect(rmBASE$rules)
cat("Number of rules: ", length(rmBASE$rules))
cat("Total conditions: ", sum(rmBASE$rules@lhs@data))
cat("Accuracy on test data: ", mean(predictionBASE == y_true))
```

In this case, the rule model is composed of seven rules. Note that the last field on the output containing the Laplace statistics was omitted for brevity.

```
    lhs                             rhs                     support confidence     lift
[1] {Petal.Length=[-Inf;2.6]}   => {Species=setosa}        0.32    1.0000000      3.125000
[2] {Petal.Width=[-Inf;0.8]}    => {Species=setosa}        0.32    1.0000000      3.125000
[3] {Petal.Length=(5.15; Inf],
     Petal.Width=(1.75; Inf]}   => {Species=virginica}     0.25    1.0000000      2.777778
[4] {Petal.Width=(1.75; Inf]}   => {Species=virginica}     0.33    0.9705882      2.696078
[5] {Sepal.Length=(5.55; Inf],
     Petal.Length=(2.6;4.75],
     Petal.Width=(0.8;1.75]}    => {Species=versicolor}    0.21    1.0000000      3.125000
[6] {Petal.Length=(2.6;4.75]}   => {Species=versicolor}    0.26    0.9629630      3.009259
[7] {Petal.Width=(0.8;1.75]}    => {Species=versicolor}    0.31    0.9117647      2.849265
```

The statistics are:

```
"Number of rules: 7 Total conditions: 10 Accuracy on test data: 0.96"
```

4.3 Configuring QCBA optimizations and printing statistics

For our demonstration purposes, we will set up a generic **qCBA** configuration (variable `baseModel_arc`), which initially disables all optimizations. To avoid repeating code for passing long argument lists to `qcba()` and for printing model statistics such as model size and accuracy on test data, we will also introduce the helper function `qcba_with_summary()`.

```
baseModel_arc <- arulesCBA2arcCBAModel(
  arulesCBAModel = rmBASE,
  cutPoints = cutPoints,
  rawDataset = trainFold,
  classAtt = classAtt
)
qcbaParams <- list(
  cbaRuleModel = baseModel_arc,
  datadf = trainFold,
  extend = "noExtend",
  attributePruning = FALSE,
  continuousPruning = FALSE,
  postpruning = "none",
  trim_literal_boundaries = FALSE,
  defaultRuleOverlapPruning = "noPruning",
  minImprovement = 0,
  minCondImprovement = 0
)

qcba_with_summary <- function(params) {
  rmQCBA <- do.call(qcba, params)
  cat("Number of rules: ", nrow(rmQCBA@rules), " ")
  cat("Total conditions: ", sum(rmQCBA@rules$condition_count), " ")
  accuracy <- CBARuleModelAccuracy(predict(rmQCBA, testFold), testFold[[classAtt]])
  cat("Accuracy on test data: ", round(accuracy, 2))
  print(rmQCBA@rules)
}
```

4.4 QCBA Refit

The following code will postoptimize the previously learnt CBA model using the refit optimization:

```
qcba_with_summary(qcbaParams)
```

This will output the following list of eight rules (note that in this and the following printouts, the columns with rule measures are omitted for brevity):

```
1                                               {Petal.Length=[-Inf;1.9]} => {Species=setosa}
2                                                {Petal.Width=[-Inf;0.6]} => {Species=setosa}
3                          {Petal.Length=[5.2;Inf],Petal.Width=[1.8;Inf]} => {Species=virginica}
4     {Sepal.Length=[5.6;Inf],Petal.Length=[3.3;4.7],Petal.Width=[1;1.7]} => {Species=versicolor}
5                                                 {Petal.Width=[1.8;Inf]} => {Species=virginica}
6                                                {Petal.Length=[3.3;4.7]} => {Species=versicolor}
7                                                   {Petal.Width=[1;1.7]} => {Species=versicolor}
8                                                                      {} => {Species=virginica}
```

The intervals (except for boundaries set to infinity) were shortened. For example, for the first rule, the training data does not contain any data point with Petal.Length=2.6 (original boundary) but it does contain the value 1.9 (new boundary).

```
any(trainFold$Petal.Length == 1.9)
any(trainFold$Petal.Length == 2.6)
```

The first returns TRUE and the second FALSE.

The statistics are

```
[1] "Number of rules: 8 Total conditions: 10 Accuracy on test data: 0.96"
```

While this is one rule more than the CPAR model, this extra rule is the explicitly included default rule (rule #8). In the **arulesCBA** CPAR model, the default rule is included in a separate slot (rmBASE$default) and is the same virginica class as the one in the QCBA rule set.

Recall that rules are applied from top to bottom. A careful inspection of the rule list shows that it contains rule 4, which is a special case of rule 7 (both predicting the versicolor class). However, neither of these rules can be removed without impact on the predictions. If the more specific rule 4 was removed, some instances would be classified differently, as more instances would reach rule 5, which would classify them as virginica. Similarly, the classification would change if we replace rule 4 with rule 7 or rule 7 with rule 4.

4.5 Adjusting boundaries and attribute pruning

We will demonstrate extension, trimming and attribute pruning steps simultaneously for efficient presentation.

```
qcbaParams$attributePruning <- TRUE
qcbaParams$trim_literal_boundaries <- TRUE
qcbaParams$extend <- "numericOnly"
qcba_with_summary(qcbaParams)
```

The list of resulting rules is

```
1  {Petal.Length=[-Inf;1.9]}                    => {Species=setosa}
2  {Petal.Width=[-Inf;0.6]}                     => {Species=setosa}
3  {Petal.Length=[5.2;Inf]}                     => {Species=virginica}
4  {Sepal.Length=[5;Inf],Petal.Length=[3.3;4.7]} => {Species=versicolor}
5  {Petal.Width=[1.8;Inf]}                      => {Species=virginica}
6  {Petal.Length=[3.3;4.7]}                     => {Species=versicolor}
7  {Petal.Width=[1;1.7]}                        => {Species=versicolor}
8  {}                                           => {Species=virginica}
```

As can be seen, the adjustment of intervals resulted in a change in the boundary for Sepal.length in rule #4. The attribute pruning removed extra conditions from rules #3 and #4 resulting in a smaller model with overall improved test accuracy:

```
"Number of rules: 8 Total conditions: 8 Accuracy on test data: 1"
```

4.6 Postpruning

The postpruning is performed on a model resulting from all previous steps.

```
qcbaParams$postpruning <- "cba"
qcba_with_summary(qcbaParams)
```

The result is a substantially reduced rule list:

```
1  {Petal.Length=[1;1.9]}                       => {Species=setosa}
2  {Petal.Length=[5.2;Inf]}                     => {Species=virginica}
3  {Sepal.Length=[5;Inf],Petal.Length=[3.3;4.7]} => {Species=versicolor}
4  {Petal.Width=[1.8;Inf]}                      => {Species=virginica}
5  {}                                           => {Species=versicolor}
```

The statistics are:

```
"Number of rules: 5 Total conditions: 5 Accuracy on test data: 1.0"
```

As can be seen, postpruning reduced the model size significantly, and in this case, the model has even better accuracy than the base CPAR model. However, in some other cases, a small average decrease in accuracy as a result of pruning has been shown in benchmarks in Kliegr and Izquierdo (2023).

4.7 Default rule pruning

The following code demonstrates the standard strategy for default rule pruning. As outlined earlier, this often provides an effective way to reduce the rule count, however, sometimes at the expense of slightly lower accuracy.

```
qcbaParams$defaultRuleOverlapPruning <- "transactionBased"
qcba_with_summary(qcbaParams)
```

The result is the final reduced rule list with the removed rule #3 from the previous printout. This rule classifies to the versicolor class. Since it is the default class, instances that were covered by this rule will be covered by the default rule. The original rule #4 covers – considering its position in the rule list – different instances of training data and, therefore, will not interfere with this.

```
1  {Petal.Length=[1;1.9]}   => {Species=setosa}
2  {Petal.Length=[5.2;Inf]} => {Species=virginica}
3  {Petal.Width=[1.8;Inf]}  => {Species=virginica}
4  {}                       => {Species=versicolor}
```

The statistics for the final model are:

```
Number of rules: 4 Total conditions: 3 Accuracy on test data: 1.0
```

Compared to the original CPAR model, the number of rules dropped from 8 (including the default rule) to 4, the number of conditions dropped from 10 to 3, and the accuracy increased by 0.04 points. This is an illustrative example and actual results may vary for a particular dataset. On average, the improvement reported over CPAR as measured on 22 benchmark datasets was a 2% improvement in accuracy, a 40% reduction in the number of rules and a 29% reduction in the number of conditions Kliegr and Izquierdo (2023). More details on the benchmarks are covered in Section Built-in benchmark support.

5 Interoperability with Rule Learning Packages in CRAN

The **qCBA** package is able to process CBA models produced by all three CBA implementations in CRAN: **arc**, **arulesCBA**, **rCBA**. Additionally, it can process other rule models generated by **arulesCBA** such as CPAR and SBRL models generated by the package **sbrl**.

On the input, `qcba()` requires an instance of `CBARuleModel`, which has the following slots:

- `rules`: list of rules in the model (instance of `rules` from `arules` package.
- `cutp`: specification of cutpoints used to discretize numerical attributes,
- `classAtt`: name of the target attribute,
- `attTypes`: types of the attributes (numeric or factor).

As shown below, the instance of this class is created automatically when used with **arc** and through prepared helper functions for other libraries. The code examples in the following are built on the data preparation described in Subsection Data.

5.1 arc package

As the **arc** package was specifically designed for **qCBA**, it outputs an instance of the `CBARuleModel` class, which is accepted by the `qcba()` function. The postprocessing can thus be directly applied to the result of `cba()`.

```
rmCBA <- cba(datadf = trainFold, classAtt = "Species")
rmqCBA <- qcba(cbaRuleModel = rmCBA, datadf = trainFold)
```

By default, the function `cba()` from the **arc** package learns candidate association rules using automatic threshold detection using the heuristic algorithm from (Kliegr and Kuchar, 2019). Therefore, no support and confidence thresholds have to be passed. A more complex case involving custom discretization and thresholds was demonstrated in Section Primer on building rule-based classifiers for QCBA in R.

5.2 arulesCBA package

Compared to the previous example, there is an extra line with a call to a helper function.

```
library(arulesCBA)
arulesCBAModel <- arulesCBA::CBA(Species ~ ., data = train_disc, supp = 0.1, conf = 0.9)
CBAmodel <- arulesCBA2arcCBARuleModel(arulesCBAModel, discrModel$cutp, iris, classAtt)
qCBAmodel <- qcba(cbaRuleModel = CBAmodel, datadf = iris)
```

Note that we passed a prediscretized data in irisDisc to the package arulesCBA. While the arulesCBA package allows for supervised discretization using the arulesCBA::discretizeDF.supervised() method, the cutpoints determined during the discretization are not exposed in a machine-readable way. Therefore, when arulesCBA is used in conjunction with qCBA, the discretization should be performed using arc::discrNumeric(). Since both arc and arulesCBA internally use the MDLP method from the discretization package, this should not influence the results.

5.3 rCBA package

The logic of the use of rCBA package is similar to that of the previously covered package; it is only necessary to ensure the use of a different conversion function:

```
library(rCBA)
rCBAmodel <- rCBA::build(train_disc)
CBAmodel <- rcbaModel2CBARuleModel(rCBAmodel, discrModel$cutp, iris, "Species")
qCBAmodel <- qcba(CBAmodel, iris)
```

Confidence and support thresholds are not specified as rCBA uses automatic confidence and threshold tuning using the simulated annealing algorithm from (Kliegr and Kuchar, 2019). The rCBA package internally uses a Java implementation of CBA, which may result in faster performance on some datasets than arulesCBA (Hahsler et al., 2019).

5.4 sbrl and other packages

The logic of the use of qCBA package for sbrl is similar; it is again only necessary to use a dedicated conversion function sbrlModel2arcCBARuleModel().

An example for sbrl is contained in the qCBA package documentation, as sbrl requires additional preprocessing and postprocessing: sbrl requires a specially named target attribute, allows only for binary targets, and outputs probabilities rather than specific class predictions.

For compatibility with packages that do not use the arules data structures, there is also the customCBARuleModel class, which takes rules as a dataframe conforming to the format used in arules that can be obtained with (as(rules,"data.frame")).

6 Built-in benchmark support

The qCBA package has built-in support for benchmarking over all supported types of algorithms covered in the previous section. This includes arulesCBA implementations of CBA, CMAR, CPAR, PRM and FOIL2 (Quinlan and Cameron-Jones, 1993).

By default, an average of two runs of each algorithm is performed.

```
# learn with default metaparameter values
stats <- benchmarkQCBA(train = trainFold, test = testFold, classAtt = classAtt)
print(stats)
```

The result of the last printout is

```
           CBA   CMAR   CPAR   PRM  FOIL2 CBA_QCBA CMAR_QCBA CPAR_QCBA PRM_QCBA FOIL2_QCBA
accuracy  1.00  0.960  0.960 0.960  0.960    1.000     1.000     1.000    1.000      0.960
rulecount 5.00 25.000  7.000 6.000  8.000    4.000     4.000     4.000    4.000      5.000
modelsize 5.00 52.000 10.000 9.000 13.000    3.000     3.000     3.000    3.000      5.000
buildtime 0.05  0.535  0.215 0.205  0.215    0.058     0.138     0.059    0.059      0.081
```

The function can be easily turned into a comparison with round((stats[,6:10] / stats[,1:5] -1),3), the result is then:

```
          CBA_QCBA CMAR_QCBA CPAR_QCBA PRM_QCBA FOIL2_QCBA
accuracy     0.000     0.042     0.042    0.042      0.000
rulecount   -0.200    -0.840    -0.429   -0.333     -0.375
modelsize   -0.400    -0.942    -0.700   -0.667     -0.615
buildtime    0.204    -0.737    -0.681   -0.722     -0.665
```

This shows that on the iris, depending on the base algorithm, QCBA decreased the rule count between 20% and 84%, while accuracy remained unchanged or increased by about 4%. The last row shows that the time required by QCBA is, for four out of five studied reference algorithms, lower than what it takes to train the input model by the corresponding algorithm. The `benchmarkQCBA()` function can also accept custom metaparameters and selected base rule learners. The user can also choose the number of runs (`iterations` parameter) and obtain the resulting models from the last iteration.

Since the outputs of some learners may depend on chance, the function also allows setting the random seed through the optional `seed` argument. Note that the provided seed is not used for splitting data, which needs to be performed externally. This approach provides the most control for the user, avoids replicating code in other R packages and functions aimed at splitting data, and also improves reproducibility.

```
output <- benchmarkQCBA(
  trainFold,
  testFold,
  classAtt,
  train_disc,
  test_disc,
  discrModel$cutp,
  CBA = list(support = 0.05, confidence = 0.5),
  algs = c("CPAR"),
  iterations = 10,
  return_models = TRUE,
  seed = 1
)

message("Evaluation statistics")
print(output$stats)
message("CPAR model")
inspect(output$CPAR[[1]])
message("QCBA model")
print(output$CPAR_QCBA[[1]])
```

This will produce output with a list of rules similar to the final CPAR model presented in Section Demonstration of individual QCBA optimization steps. A more complex benchmark of computational costs is presented in Kliegr and Izquierdo (2023), which also includes the study of various data sizes and the effect of varying the number of unique values in the dataset. A brief overview of the main results was presented in Subsection Computational costs for large datasets.

A GitHub repository `https://github.com/kliegr/arcBench` contains scripts that extend this workflow into automation across multiple datasets and materialized splits in each dataset. It also includes support for benchmarking additional rule learning algorithms, including SBRL, Python packages producing IDS models (Lakkaraju et al., 2016) and Weka libraries for RIPPER (Cohen, 1995) and FURIA (Hühn and Hüllermeier, 2009). Detailed benchmarking results are included in (Kliegr and Izquierdo, 2023).

7 Conclusions

Quantitative CBA ameliorates one of the major drawbacks of association rule classification, the adherence of rules comprising the classifier to the multidimensional grid created by discretization of numerical attributes. By working with the original continuous data, the algorithm can improve the fit of the rules and consequently reduce their count. The QCBA algorithm does not introduce any mandatory thresholds or meta-parameters for the user to set or optimize, although it does allow disabling the individual optimizations. The **qCBA** package implements QCBA, allowing the postprocessing of the output of all three CBA implementations currently in CRAN. The package can also be used in conjunction with other association rule-based models, including those producing rule sets and using multi-rule classification.

The QCBA algorithm is described in detail in Kliegr and Izquierdo (2023), the package documentation is available in Kliegr (2024), and additional information is available at https://github.com/kliegr/qcba, which also features an interactive RMarkdown tutorial supplementing this paper.

8 Acknowledgment

The author thanks the Faculty of Informatics and Statistics, Prague University of Economics and Business, for long-term institutional support of research activities.

References

R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1-55860-153-8. [p157]

M. Azmi and A. Berrado. RCAR framework: building a regularized class association rules model in a categorical data space. In *Proceedings of the 13th International Conference on Intelligent Systems: Theories and Applications*, pages 1–6, 2020. [p156]

W. W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995. [p170]

U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *13th International Joint Conference on Uncertainly in Artificial Intelligence (IJCAI93)*, pages 1022–1029, 1993. [p156]

M. Hahsler, B. Grün, and K. Hornik. Introduction to arules–mining association rules and frequent item sets. *SIGKDD Explor*, 2(4):1–28, 2007. [p160]

M. Hahsler, S. Chelluboina, K. Hornik, and C. Buchta. The arules r-package ecosystem: analyzing interesting patterns from large transaction data sets. *Journal of Machine Learning Research*, 12(Jun):2021–2025, 2011. [p156]

M. Hahsler, I. Johnson, T. Kliegr, and J. Kuchař. Associative classification in R: arc, arulescba, and rcba. *R Journal*, 9(2), 2019. [p156, 169]

J. Hühn and E. Hüllermeier. FURIA: an algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, 19(3):293–319, 2009. [p170]

T. Kliegr. *Association Rule Classification*, 2016. URL https://CRAN.R-project.org/package=arc. R package version 1.1.3. [p156]

T. Kliegr. *qCBA: Quantitative Classification by Association Rules*, 2024. URL https://CRAN.R-project.org/package=qCBA. R package version 1.0. [p171]

T. Kliegr and E. Izquierdo. QCBA: improving rule classifiers learned from quantitative data by recovering information lost by discretisation. *Applied Intelligence*, 53(18):20797–20827, 2023. [p156, 157, 158, 162, 163, 164, 167, 168, 170, 171]

T. Kliegr and J. Kuchar. Tuning hyperparameters of classification based on associations (CBA). In *ITAT*, pages 9–16, 2019. [p158, 168, 169]

J. Kuchař and T. Kliegr. *rCBA: CBA Classifier for R*, 2019. URL https://CRAN.R-project.org/package=rCBA. R package version 0.4.3. [p156]

H. Lakkaraju, S. H. Bach, and J. Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1675–1684, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. [p170]

W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 369–376, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1119-8. [p157]

B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 80–86. AAAI Press, 1998. [p156, 157, 159]

T. G. Michael Hahsler, Ian Johnson. *arulesCBA: Classification Based on Association Rules*, 2024. URL https://CRAN.R-project.org/package=arulesCBA. R package version 1.2.7. [p156]

J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993. [p158]

J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *European conference on machine learning*, pages 1–20. Springer, 1993. [p169]

K. Vanhoof and B. Depaire. Structure of association rule classifiers: a review. In *2010 International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 9–12, November 2010. [p157]

H. Yang, C. Rudin, and M. Seltzer. Scalable Bayesian rule lists. In *International conference on machine learning*, pages 3921–3930. PMLR, 2017. [p156]

H. Yang, C. Rudin, and M. Seltzer. *sbrl: Scalable Bayesian Rule Lists Model*, 2024. URL https://CRAN.R-project.org/package=sbrl. R package version 1.4. [p156, 157]

X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 331–335. SIAM, 2003. [p157]

Tomas Kliegr
Prague University of Economics and Business
Winston Churchill Sq. 4, Prague
Czech Republic
https://nb.vse.cz/~klit01
ORCiD: 0000-0002-7261-0380
tomas.kliegr@vse.cztomas.kliegr@vse.cz