Associative Classification in R: arc, arulesCBA, and rCBA
by Michael Hahsler, Ian Johnson, Tomáš Kliegr and Jaroslav Kuchař

Abstract Several methods for creating classifiers based on rules discovered via association rule mining have been proposed in the literature. These classifiers are called associative classifiers and the best-known algorithm is Classification Based on Associations (CBA). Interestingly, only very few implementations are available and, until recently, no implementation was available for R. Now, three packages provide CBA. This paper introduces associative classification, the CBA algorithm, and how it can be used in R. A comparison of the three packages is provided to give the potential user an idea about the advantages of each of the implementations. We also show how the packages are related to the existing infrastructure for association rule mining already available in R.

Introduction

Association rule learning (Agrawal et al., 1993) was initially designed for data exploration to discover interesting patterns in very large and sparse datasets. Several years after its inception, association rule learning was also adapted to create rule-based classification models. The first algorithm called CBA (Classification Based on Associations) was introduced by Liu et al. (1998). While there were multiple follow-up algorithms providing some improvements in classification performance (e.g., CPAR (Yin and Han, 2003) and FARC-HD-OVO (Elkano et al., 2015)), these performance gains are offset by deterioration of comprehensibility of the produced set of rules. For some practical applications, CBA still provides a very good balance between accuracy, speed, and model comprehensibility. Unlike many more recent approaches, CBA classifiers are easy to interpret and apply: the ruleset is relatively small, rules are crisp (i.e., not fuzzy rules), and rules are sorted according to strength. CBA uses a simple first-match strategy for classification, where the first matching rule determines the predicted class.

With the exception of fuzzy approaches such as FARC-HD, associative classification approaches require a dataset in the form of transactions, i.e., all attributes need to be binary indicators and thus numeric attributes in the input data need to be discretized. This puts additional demands on the user and may deteriorate model fit on datasets with numerical attributes. Another disadvantage relating to CBA and most other associative classification approaches is that these algorithms require the user to specify a minimum support and a minimum confidence threshold for association rule mining. The performance (accuracy and speed) is typically very sensitive to proper selection of these threshold values. Setting these thresholds too high can result in the classifier underfitting the dataset or even an empty rule list. Too low values can lead to combinatorial explosion with an excessive number of rules generated, leading to speed and memory issues. Another limitation that applies specifically to CBA is that even when the user specifies reasonable thresholds, CBA typically produces more rules than other related approaches (Alcala-Fdez et al., 2011). These limitations may be the reason why CBA implementations have not been available in many computational environments for machine learning and statistics. However, in the last two years, three packages with CBA implementations appeared on CRAN (listed by date of the first release): rCBA (Kuchar, 2018), arc (Kliegr, 2018) and arulesCBA (Johnson and Hahsler, 2019). Each of these packages offers some enhancements over the original CBA algorithm to address some shortcomings of association rule-based classification.

The goal of this paper is to introduce prospective users to the concepts used in associative classification and the CBA algorithm in particular. We provide detailed information on the three available R packages and the enhancements they provide, followed by hands-on examples.

The paper is organized as follows. We first introduce association rule mining followed by a discussion of the CBA algorithm. We present existing CBA implementations and focus on the features and the use of the three new R implementations. We conclude with a short comparison of features and a run-time comparison on a typical dataset.

Background: Association rule mining

Associative classifiers like CBA are based on association rules. Mining association rules was first introduced by Agrawal et al. (1993) and, following the notation used by Agrawal et al. (1993), Hahsler et al. (2005) and Tan et al. (2006), can formally be defined as:

Let $D = \{t_1, t_2, \ldots, t_m\}$ be a set of transactions called the database, and let $I = \{i_1, i_2, \ldots, i_n\}$ be
the set of all items considered in the database. Each transaction in D has a unique transaction ID and contains a subset of the items in I. To illustrate the concepts, we use a small example from the supermarket domain introduced by Hahsler et al. (2005). The set of items is I = {milk, bread, butter, beer} and a small database containing five transactions with these items is shown in Figure 1. An example rule for the supermarket could be {milk, bread} ⇒ {butter} meaning that if milk and bread is bought, customers also may buy butter.

A rule is defined as an expression X ⇒ Y where X, Y ⊆ I and X ∩ Y = ∅. The sets of items (for short itemsets) X and Y are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule. Often rules are restricted to only a single item in the consequent. Association rules are rules which surpass user-specified minimum support and minimum confidence thresholds. The support, supp(X), of an itemset X is a measure of importance defined as the proportion of transactions in the dataset which contain the itemset. The confidence of a rule is defined as conf(X ⇒ Y) = supp(X ∪ Y)/supp(X), measuring how likely it is to see Y in a transaction containing X.

An association rule X ⇒ Y needs to satisfy
\[
supp(X ∪ Y) ≥ σ \quad \text{and} \quad conf(X ⇒ Y) ≥ δ,
\]
where σ and δ are the minimum support and minimum confidence thresholds, respectively. For example, the rule {milk, bread} ⇒ {butter} has a support of 1/5 = 0.2 and a confidence of 0.2/0.4 = 0.5 in the database in Figure 1, which means that for 50% of the transactions containing milk and bread, the rule is correct. Confidence can be interpreted as an estimate of the probability P(Y|X), the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS (see, e.g., Hipp et al., 2000).

Another popular measure for the importance of a rule is lift (Brin et al., 1997). The lift of a rule is defined as lift(X ⇒ Y) = supp(X ∪ Y)/(supp(X) supp(Y)), and can be interpreted as the deviation of the support of the whole rule from the support expected under independence given the supports of the LHS and the RHS. Lift values greater than one indicate positive associations between the rule’s LHS and RHS.

Because associative classification is based on association rules, transaction data is required as the input. Here each object (or instance) needs to be converted into a transaction containing only binary items. Discrete variables can be converted into items using a set of 0-1 dummy variables, one for each possible value. Continuous variables need to be first discretized and then converted. Typically, discretization for associative classifiers is performed using a class-based (also called supervised) discretization strategy, which identifies ranges for several intervals using information from the class variable (Yin and Han, 2003). The most popular method for class-based discretization is Minimum Description Length Principle (MDLP) discretization (Fayyad and Irani, 1993), which uses a greedy procedure to find cut points based on the entropy of the induced partition of the data with respect to the class variable. MDLP was also used in the initial paper on CBA (Liu et al., 1998). The procedure uses a stopping rule to find the appropriate number of cut points. One of the advantages of MDLP is that there are no external parameters to be set; the number of bins is determined automatically.

### The CBA algorithm

Liu et al. (1998) proposed the first approach to associative classification called CBA. In CBA, a special type of association rules called Class Association Rules (CARs) are used for classification. A CAR is an association rule that conforms to the additional constraint that the consequent (RHS) of the rule is a single item that is associated with a class label for the classification problem. CBA proposed the following steps to perform associative classification (Vanhoof and Depaire, 2010):

1. Mine a set of class association rules (CARs),
2. prune and sort the rules,
3. classify new objects using the RHS of the first matching rule.

<table>
<thead>
<tr>
<th>transaction ID</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>milk, bread</td>
</tr>
<tr>
<td>2</td>
<td>bread, butter</td>
</tr>
<tr>
<td>3</td>
<td>beer</td>
</tr>
<tr>
<td>4</td>
<td>milk, bread, butter</td>
</tr>
<tr>
<td>5</td>
<td>bread, butter</td>
</tr>
</tbody>
</table>

Figure 1: An example supermarket database with five transactions.
Within the original paper, the first step is handled by a modification of the popular APRIORI algorithm (Agrawal and Srikant, 1994) for mining CARs. The modification includes an optional pruning step based on the rule's pessimistic error rate with the goal to reduce the size of the set of considered CARs. According to results reported in Liu et al. (1998), the absence of pessimistic pruning does not affect classifier accuracy and a regular implementation of APRIORI can be used. The output of association rule learning algorithms is determined by two parameters, the minimum confidence and the support thresholds. In light of classification, confidence gives the proportion of objects correctly classified by the rule in the training set. Therefore it can be seen as an optimistic estimate of the accuracy of the rule.

The main obstacles for straightforward use of the discovered CARs as a classifier are the excessive number of rules discovered even on small datasets, the fact that contradicting rules are generated, and the absence of a default rule. To address these issues, CBA employs rule sorting and a data coverage pruning procedure to reduce the number of rules. Two variants were proposed in the original paper (Liu et al., 1998): the direct M1 version, and the M2 version which reduces data access. Accessing data fewer times is especially useful if the data is too large to be stored in main memory. The amount of available main memory has increased substantially since the original paper was published, and all the discussed implementations store all data in main memory, making the improvements of M2 less relevant. For pruning, the rules are first ranked in the order of their strength:

1. Rule A is ranked higher if confidence of rule A is greater than that of rule B.
2. For rules tied for 1, rule A is ranked higher if support of rule A is greater than that of rule B.
3. For rules tied for 1 and 2, rule A is ranked higher if rule A is produced before rule B in the mining process. Since APRIORI applies breath-first search, rule A is ranked higher if rule A has fewer conditions (i.e., a smaller antecedent set) than rule B.

Rules are processed in ranking order. After each rule is processed, the matching (covered) transactions are removed. If a rule does not correctly cover at least one instance, it is deleted (pruned). In CBA, data coverage pruning is combined with default rule pruning. A default rule is a rule added to the end of the rule set with the majority class in the uncovered transactions in the RHS and an empty LHS. This rule ensures that a query instance is always classified even if it is not matched by any other rule in the classifier. The algorithm prunes all rules below the current rule, if a default rule inserted at that place reduces the total number of errors on the training set.

Other algorithms. Since CBA was introduced, several competing associative classification approaches have been proposed to optimize accuracy, training time, and ruleset size. Two popular extensions of CBA are CMAR (Li et al., 2001) and CPAR (Yin and Han, 2003). A multiclass-focused approach called Multiclass Associative Classification (MAC) (Abdelhamid et al., 2012) has been proposed for expanding CBA with the goal of more accurately addressing classification problems with many different class labels.

Separate from CBA is a small class of associative classifiers called rule-induction classifiers which generate excessively large rulesets and then use greedy pruning strategies to reduce the size while maintaining classification accuracy. Common examples of this technique are RIPPER (Cohen, 1995) and SLIPPER (Cohen and Singer, 1999).

Recently, instead of relying on heuristics, several optimization approaches have been proposed for selecting the rules used by the classifier. Scalable Bayesian Rule Lists Model (Yang et al., 2017) tries to identify a small subset of mined CARs by optimizing the posterior of a Bayesian hierarchical model over rule lists. The method is implemented in the R package sbrl (Yang et al., 2019). Azmi et al. (2019) propose to view the items in CARs as binary predictor variables and use logistic regression with L1 regularization to find a small subset of association rules for classification. The approach is available in arulesCBA as function RCAR().

While several alternative approaches have been introduced, CBA still acts as a strong contender in associative classification and is typically used as the benchmark against which new methods are assessed (Alcala-Fdez et al., 2011). A comparison between CBA and selected successors is performed in Kliegr (2017).

Implementations

There are only a few implementations of CBA available. Table 1 shows them ordered by the first release date and summarizes the used licenses and programming languages.

In the following, we discuss the three currently available implementations of CBA in R. We will first present each package individually and then compare the packages by providing code for the same
Table 1: Review of existing CBA implementations

<table>
<thead>
<tr>
<th>Software</th>
<th>1st release</th>
<th>License</th>
<th>Language</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUCS-KDD</td>
<td>2004</td>
<td>not stated</td>
<td>Java</td>
<td>Endorsed by Bing Liu, the main CBA author. See <a href="http://cgi.csc.liv.ac.uk/~frans/KDD/Software/CBA/cba.html">http://cgi.csc.liv.ac.uk/~frans/KDD/Software/CBA/cba.html</a></td>
</tr>
<tr>
<td>KEEL</td>
<td>2010</td>
<td>GPL-3</td>
<td>Java</td>
<td>At the writing of this paper not available via RKEEL. See <a href="http://sci2s.ugr.es/keel/">http://sci2s.ugr.es/keel/</a></td>
</tr>
<tr>
<td>rCBA</td>
<td>2015</td>
<td>Apache 2.0</td>
<td>R</td>
<td>See <a href="https://CRAN.R-project.org/package=rCBA">https://CRAN.R-project.org/package=rCBA</a></td>
</tr>
<tr>
<td>arc</td>
<td>2016</td>
<td>AGPL-3</td>
<td>R with Java</td>
<td>From the authors of the arules R package. See <a href="https://CRAN.R-project.org/package=arulesCBA">https://CRAN.R-project.org/package=arulesCBA</a></td>
</tr>
<tr>
<td>arulesCBA</td>
<td>2016</td>
<td>GPL-3</td>
<td>R with C</td>
<td></td>
</tr>
</tbody>
</table>

classification problem implemented with each of the packages. We will use as the example dataset the well-known iris dataset (Fisher, 1936) and split it into 80% for training and 20% for testing.

data("iris")
iris <- iris[sample(seq(nrow(iris))), ]
iris_train <- iris[1:(nrow(iris)*.8), ]
iris_test <- iris[-(1:(nrow(iris)*.8)), ]

The data contains 150 flowers described by four quantitative variables representing different measurements and a categorical variable indicating one of three different species.

head(iris_train)

```
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
18     5.1     3.5     1.4     0.3  setosa
93     5.8     2.6     4.0     1.2 versicolor
91     5.5     2.6     4.4     1.2 versicolor
92     6.1     3.0     4.6     1.4 versicolor
126    7.2     3.2     6.0     1.8 virginica
149    6.2     3.4     5.4     2.3 virginica
```

The classification problem we use for the examples is to predict species using the four measurements.

All three packages integrate with the infrastructure for association rule mining in R implemented in package arules (Hahsler et al., 2005) and the ecosystem of related packages (Hahsler et al., 2011). While the presented packages can perform discretization, the conversion of a dataset with continuous variable to a set of transactions with binary items, and mining class association rules (CARs) internally and transparent to the user, we will give here a short example of how the packages arules and arulesCBA can be used to perform these tasks. First, we discretize the data using supervised discretization based on minimum description length principle (MDLP) offered by packages like discretization (Kim, 2012). Here we use the discretizeDF.supervised function provided in arulesCBA.

library("arules")
library("arulesCBA")
iris_train_disc <- discretizeDF.supervised(Species ~ ., data = iris_train, method = "mdlp")
head(iris_train_disc)

```
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
18 [-Inf,5.55) [3.35, Inf) [-Inf,2.6) [-Inf,0.75) setosa
93 [5.55, Inf) [-Inf,2.95) [2.6,4.75) [0.75,1.75) versicolor
91 [-Inf,5.55) [-Inf,2.95) [2.6,4.75) [0.75,1.75) versicolor
92 [5.55, Inf) [2.95,3.35) [2.6,4.75) [0.75,1.75) versicolor
126 [5.55, Inf) [2.95,3.35) [5.05, Inf) [1.75, Inf) virginica
149 [5.55, Inf) [3.35, Inf) [5.05, Inf) [1.75, Inf) virginica
```

Now we can convert the discretized data into transactions which automatically converts factors into binary items with labels composed of variable name and factor labels.
trans_train <- as(iris_train_disc, "transactions")
inspect(head(trans_train, n = 3))

[1] {Sepal.Length=[-Inf,5.55),
   Sepal.Width=[3.35, Inf],
   Petal.Length=[-Inf,2.6),
   Petal.Width=[-Inf,0.75),
   Species=setosa} 18

[2] {Sepal.Length=[5.55, Inf],
   Sepal.Width=[-Inf,2.95),
   Petal.Length=[2.6,4.75),
   Petal.Width=[0.75,1.75),
   Species=versicolor} 93

[3] {Sepal.Length=[-Inf,5.55),
   Sepal.Width=[-Inf,2.95),
   Petal.Length=[2.6,4.75),
   Petal.Width=[0.75,1.75),
   Species=versicolor} 91

Note that the class variable is translated into several items, all starting with Species=. From these
transactions, CARs can be mined by restricting the items which can appear in the right-hand-side
of the rules. This can be done with the APRIORI implementation available in arules by specifying
appearance restrictions.

rules <- apriori(trans_train, parameter = list(support = 0.01, confidence = 0.8),
appearance = list(rhs = grep("Species="., itemLabels(trans_train), value = TRUE),
default = "lhs"))
arulesCBA contains a convenience function called mineCARs to make setting the appropriate
appearance easier using the standard formula interface.

rules <- mineCARs(Species ~ ., data = trans_train, support = 0.01, confidence = 0.8)
rules

set of 78 rules

inspect(head(rules, n = 3))

lhs rhs support confidence lift count
[1] {Sepal.Width=[3.35, Inf]} => {Species=setosa} 0.19 0.85 2.6 23
[2] {Petal.Length=[5.05, Inf]} => {Species=virginica} 0.27 1.00 3.0 32
[3] {Petal.Length=[2.6,4.75)} => {Species=versicolor} 0.29 0.97 2.9 35

Test data can be discretized consistently with the training data using discretizeDF, which applies
the discretization used in the second argument to the data in the first argument. Followed by a
conversion to transactions.

iris_test_disc <- discretizeDF(iris_test, iris_train_disc)
trans_test <- as(iris_test_disc, "transactions")

While these steps are performed in most cases by the discussed packages internally, it is still
helpful to understand the process. One of the advantages of associative classifiers is that the rule base
can be inspected and, therefore, it is important to understand the transformations used to create items.
Next, we will discuss the packages in alphabetical order.

Package arc

The R package arc (Kliegr, 2018) provides a pure R implementation of the rule pruning step of CBA.
The association rule learning step is handled by the implementation of APRIORI in package arules. arc
implements the M1 version of the CBA pruning step (Liu et al., 1998) and offers in addition automatic
discretization and threshold tuning. A CBA model can be learned for the iris dataset as follows.

library("arc")
classifier <- arc::cba(iris_train, "Species")
The function `cba()` will create an instance of the `CBARuleModel` class for the iris dataset using `Species` as the class variable. Note that discretization is performed and that the support and confidence thresholds are automatically found.

The resulting S4 object holds a list of rules, a list of cut points (if discretization was automatically performed), the name of the class attribute, and a list of attribute types. The slot rules of the `CBARuleModel` object contains the rule base, which can be inspected by:

```r
inspect(classifier@rules)
```

<table>
<thead>
<tr>
<th>lhs</th>
<th>rhs</th>
<th>support</th>
<th>confidence</th>
<th>lift</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{Petal.Length=[-Inf;2.45],</td>
<td>`{Species=setosa}</td>
<td>0.333</td>
<td>1.00</td>
<td>3.0</td>
<td>40</td>
</tr>
<tr>
<td>Petal.Width=[-Inf;0.75]}`</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`{Sepal.Length=(5.75; Inf],</td>
<td>`{Species=virginica}</td>
<td>0.258</td>
<td>1.00</td>
<td>3.2</td>
<td>31</td>
</tr>
<tr>
<td>Sepal.Width=[-Inf;3.15],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Petal.Width=[1.75; Inf]}`</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`{Sepal.Length=(5.75; Inf],</td>
<td>`{Species=virginica}</td>
<td>0.200</td>
<td>1.00</td>
<td>3.2</td>
<td>24</td>
</tr>
<tr>
<td>Sepal.Width=[-Inf;3.15],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Petal.Width=[1.75; Inf]}`</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`{Sepal.Length=(5.75; Inf],</td>
<td>`{Species=versicolor}</td>
<td>0.200</td>
<td>1.00</td>
<td>2.8</td>
<td>24</td>
</tr>
<tr>
<td>Petal.Width=[2.45;4.95],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Petal.Width=[0.75;1.75]}`</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`{Sepal.Length=(5.45;5.75],</td>
<td>`{Species=versicolor}</td>
<td>0.092</td>
<td>1.00</td>
<td>2.8</td>
<td>11</td>
</tr>
<tr>
<td>Sepal.Width=[-Inf;3.15],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Petal.Length=[2.45;4.95],</td>
<td>`{Species=versicolor}</td>
<td>0.042</td>
<td>1.00</td>
<td>2.8</td>
<td>5</td>
</tr>
<tr>
<td>Petal.Width=[0.75;1.75]}`</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`{Sepal.Length=(5.75; Inf],</td>
<td>`{Species=versicolor}</td>
<td>0.333</td>
<td>0.98</td>
<td>2.7</td>
<td>40</td>
</tr>
<tr>
<td>Sepal.Width=[3.15; Inf],</td>
<td>`{Species=versicolor}</td>
<td>0.308</td>
<td>0.31</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Petal.Length=[2.45;4.95],</td>
<td>`{Species=virginica}</td>
<td>0.308</td>
<td>0.31</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Petal.Width=[0.75;1.75]}`</td>
<td>`{Species=virginica}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Predictions for new data can be obtained using `predict()`. The new data is discretized automatically to match the rules.

```r
predict(classifier, head(iris_test))
```

```
[1] virginica setosa versicolor virginica setosa versicolor virginica
Levels: setosa versicolor virginica
```

Next, we discuss the new features of automatic discretization and threshold tuning.

**Automatic discretization.** Since association rule classification is a supervised task, the discretization can take advantage of using the class label. In the `arc` package, automatic discretization with MDLP is enabled by default. All numeric explanatory attributes with three or more distinct values are by default subject to discretization. The package relies on the `discretization` package (Kim, 2012). The `arc` package provides several convenience functions that allow to perform discretization of all attributes at once, addressing some of the shortcomings of the `mdlp` function from the `discretization` package, such as the inability to handle missing values, or skip non-numeric attributes. Only attributes containing at least a preset number of distinct values are discretized. The package is also capable of discretizing the target attribute if necessary. For this purpose, unsupervised discretization (clustering) is used.

**Automatic threshold tuning.** Association rule learning is notorious for how difficult it is to set the minimum support and minimum confidence thresholds. The necessity to set these thresholds applies also to CBA. The `arc` package contains an optional procedure for automatic setting of these thresholds detailed in (Kliegr and Kuchar, 2019). The package contains a wrapper for the `apriori` function from the `arules` package that iterative changes mining parameters (maximum antecedent length, minimum support threshold and minimum confidence threshold) until a desired number of rules is obtained, all options are exhausted or a preset time limit is reached. The desired number of rules can be specified by the `target_rule_count` parameter.

The `arc` package also supports manual specification of thresholds:

```r
classifier <- arc::cba(iris_train, "Species",
  rulelearning_options = list(minsupp = 0.05, minconf = 0.9,
```
minlen = 1, maxlen = 5, maxtime = 1000, target_rule_count = 50000,
trim=TRUE, find_conf_supp_thresholds = FALSE))

classifier@rules

set of 3 rules

Unlike other implementations of CBA, which also implement the M2 version of CBA described by
Liu et al. (1998), the arc package relies solely on the M1 version. However, the implementation does not
follow the originally proposed way relying on iterative processing of rules in the sort order. Instead,
the pruning steps in M1 are implemented using more efficient multiplication of sparse matrices
exposed by the arules package, which relies on the optimized C code from the Matrix package (Bates
and Maechler, 2017).

Package arulesCBA

The arulesCBA package (Johnson and Hahsler, 2019) is an extension of the arules package and strives
to integrate seamlessly with its association rule mining infrastructure. The packages allows the user to
set a time limit for rule mining, exposed by the arules package. The core operations of arulesCBA are
implemented in a mixture of R and C to speed up processing. arulesCBA implements both versions
of the pruning step, M1 and the optimized M2 version. The code for the pruning algorithm is heavily
optimized by using rule-indexed sparse matrix representation, sparse matrix operations via package
Matrix (Bates and Maechler, 2017) and prefix trees.

The arulesCBA interface. In arulesCBA, classifiers are created using the CBA() function. An advant-
age of this package for R users is that it it consistently uses the well-known formula interface for
building classifier models and for supervised discretization. Users can provide a number of options
to the function to tune discretization, rule mining, and model building. The following is the list of
available parameters to the CBA function.

• formula: A symbolic description of the model to be fitted using a standard formula object of
the from:

class ~ explanatory variables

The class is the variable name (part of the item label before =). Explanatory variables are
separated using + and the special dot symbol . for all variables is also allowed.

• data: A data.frame containing the training data. If necessary, discretization is automatically
applied. Alternatively, also a transaction set can be supplied.

• support, confidence: Minimum support and confidence thresholds for mining CARs with
APRIORI.

• parameter, control: Parameter and control lists passed on to the apriori() function from the
arules package.

• disc.method: Discretization method for factorizing numeric input (default: "mdlp"). One of
('mdlp', 'caim', 'chi2', 'caac', 'ameva', 'chimerge', 'extendedchi2', 'modchi2').

A classifier for the iris dataset can be learned as follows.

library("arulesCBA")
classifier <- arulesCBA::CBA(Species ~ ., data = iris_train,
supp = 0.05, confidence = 0.9)

classifier

CBA Classifier Object
Class: Species (labels: setosa, versicolor, virginica )
Default Class: Species=setosa
Number of rules: 2
Classification method: first
Description: CBA algorithm by Liu, et al. 1998 with support=0.05 and
confidence=0.9

CBA() returns an object of class CBA which contains all needed information for classification. A
print method shows the settings used for the classifier. Prediction follows the usual approach in R.
predict(classifier, head(iris_test))

[1] virginica setosa versicolor virginica setosa setosa
Levels: setosa versicolor virginica

The rule base is stored as a rules object from arules and can be extracted for inspection using the rules() function.

inspect(rules(classifier))

<table>
<thead>
<tr>
<th>lhs</th>
<th>rhs</th>
<th>support</th>
<th>confidence</th>
<th>lift</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1] {Petal.Width=(1.75, Inf]} =&gt; {Species=virginica}</td>
<td>0.29</td>
<td>1.00</td>
<td>3.1</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>[2] {Sepal.Length=(5.55, Inf], Petal.Width=(0.8,1.75]} =&gt; {Species=versicolor}</td>
<td>0.26</td>
<td>0.91</td>
<td>2.7</td>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>

Note that only two rules are shown, while ar above produced three rules. The reason is that arulesCBA stores the default class Species=setosa separate from the rule base while ar includes it.

Advanced use of arulesCBA. arulesCBA is implemented with flexibility and future extensions in mind. For example, to have optimal control over the discretization process, the user can discretize the data manually before learning the classifier. The discretization functions in arules and arulesCBA retain enough information so that predict() can later automatically discretize the new data.

Another extension implemented in CBA_ruleset() allows the user to create an associative classifier by providing a custom rule base in the form of a rules object. For example, we can easily create a classifier from a set of CARs using, for example, majority voting instead of CBA’s first-match strategy for classification.

rules <- mineCARs(Species ~ ., trans_train, parameter = list(support = 0.01, confidence = 0.8))

classifier <- arulesCBA::CBA_ruleset(Species ~ ., rules, method = "majority")

classifier

CBA Classifier Object
Class: Species (labels: setosa, versicolor, virginica )
Default Class: Species=versicolor
Number of rules: 78
Classification method: majority
Description: Custom rule set

This allows the user to experiment with different pruning methods and classification strategies.

Package rCBA

The rCBA package (Kuchar, 2018) was the first available implementation of the CBA algorithm on CRAN. The main algorithms are implemented in Java and it is the only R implementation that supports the use of multiple CPU cores during pruning. The package provides wrapper functions for pruning, prediction, and the FPGrowth association rule mining algorithm (Han et al., 2004). rCBA includes both, the M1 and the M2 version of the CBA algorithm. It also includes data coverage pruning and automatic threshold tuning.

Model building with automatic tuning of parameters and APRIORI is done as follows.

library("rCBA")
classifier <- rCBA::build(iris_train)

inspect(classifier$model)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>support</th>
<th>confidence</th>
<th>lift</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {Petal.Width=0.2} =&gt; {Species=setosa}</td>
<td>0.183</td>
<td>1.00</td>
<td>2.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 {Petal.Width=1.3} =&gt; {Species=versicolor}</td>
<td>0.108</td>
<td>1.00</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 {Petal.Length=1.4} =&gt; {Species=setosa}</td>
<td>0.100</td>
<td>1.00</td>
<td>2.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 {Petal.Length=1.5} =&gt; {Species=setosa}</td>
<td>0.092</td>
<td>1.00</td>
<td>2.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 {Petal.Length=1.8} =&gt; {Species=virginica}</td>
<td>0.083</td>
<td>1.00</td>
<td>3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 {Petal.Width=2.3} =&gt; {Species=virginica}</td>
<td>0.058</td>
<td>1.00</td>
<td>3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 {Petal.Width=0.4} =&gt; {Species=setosa}</td>
<td>0.058</td>
<td>1.00</td>
<td>2.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
rCBA::classification(head(iris_test), classifier$model)

[1] versicolor versicolor versicolor versicolor setosa versicolor
Levels: setosa versicolor

Pruning methods. rCBA implements both version of the proposed pruning algorithms (Liu et al., 1998) the direct M1 version, and the optimized M2 version. It also offers an option to only use data coverage pruning, called data coverage for business rule (dcbr) (Kliegr et al., 2014).

Selection of algorithms for rule learning. The CBA algorithm can generally rely on any rule learning algorithm (Liu et al., 1998). By default it uses the APRIORI implementation in arules, but it can also use rCBA’s own implementation of the FP-Growth algorithm Han et al. (2004) for the association learning step.

rulebase <- rCBA::fpgrowth(iris_train, support = 0.05, confidence = 0.9, consequent = "Species")
rulebase <- rCBA::pruning(iris_train, rulebase, method = "m2cba")
rCBA::classification(head(iris_test), rulebase)

[1] versicolor versicolor versicolor versicolor setosa versicolor
Levels: setosa versicolor

Automatic threshold tuning. Since pure random or grid search do not use any background knowledge of the algorithm, these approaches are unsuitable for optimizing the parameters of association rule learning. The implementation for the parameter optimization in rCBA is based on the simulated annealing (SA) algorithm which addresses this problems. The objective criterion, which is optimized against, is the accuracy of the model. A detailed description of the approach can be found in Kliegr and Kuchar (2019).

Comparison of R implementations

In order to help the user to decide which package addresses best the particular use case, Table 2 presents a comparison of the features and limitations of the packages. Since all three packages implement the same algorithm, we did not compare classification accuracy between implementations, but performed a small run-time comparison instead.

We compare the different implementations on some standard classification problems. The results are shown in Table 3. The datasets are available in the packages mlbench, datasets, arules, and the Lymphography dataset (Lymph) (Mickalski et al., 1986) was obtained from the UCI repository1. The number of transactions ranges from 101 to 48842 and the number of items (after discretization) from 15 to 147. We used for the comparison a minimum confidence threshold of 0.5, a maximal rule length of 10 and set the minimum support so a reasonable number of classification association rules (CARs)

1https://archive.ics.uci.edu/ml/datasets.html
Table 2: Comparison of features in CBA implementations in R

<table>
<thead>
<tr>
<th>Feature</th>
<th>arc</th>
<th>arulesCBA</th>
<th>rCBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBA pruning</td>
<td>M1</td>
<td>M1/M2</td>
<td>M1/M2</td>
</tr>
<tr>
<td>Language</td>
<td>R</td>
<td>R + C</td>
<td>R + Java</td>
</tr>
<tr>
<td>Built-in discretization</td>
<td>MDLP</td>
<td>MDLP and others</td>
<td>No</td>
</tr>
<tr>
<td>Automatic threshold tuning</td>
<td>Unsupervised</td>
<td>Many rules, many rules and instances</td>
<td>Supervised</td>
</tr>
<tr>
<td>Recommended problem size</td>
<td>Small number of rules and instances</td>
<td>Medium number of rules and instances</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Run time comparison for different datasets (* indicates that the algorithm ran out of memory).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Transactions</th>
<th>Items</th>
<th>Support</th>
<th>Confidence</th>
<th>CARs</th>
<th>Rule base</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoo</td>
<td>101</td>
<td>26</td>
<td>0.01</td>
<td>0.50</td>
<td>3607</td>
<td>8</td>
<td>0.97</td>
</tr>
<tr>
<td>Lymph</td>
<td>147</td>
<td>63</td>
<td>0.10</td>
<td>0.50</td>
<td>16087</td>
<td>40</td>
<td>0.90</td>
</tr>
<tr>
<td>Iris</td>
<td>150</td>
<td>15</td>
<td>0.01</td>
<td>0.50</td>
<td>119</td>
<td>8</td>
<td>0.96</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>351</td>
<td>147</td>
<td>0.40</td>
<td>0.50</td>
<td>16321</td>
<td>10</td>
<td>0.89</td>
</tr>
<tr>
<td>BreastCancer</td>
<td>699</td>
<td>91</td>
<td>0.01</td>
<td>0.50</td>
<td>5541</td>
<td>64</td>
<td>0.99</td>
</tr>
<tr>
<td>Pima</td>
<td>768</td>
<td>19</td>
<td>0.01</td>
<td>0.50</td>
<td>3536</td>
<td>76</td>
<td>0.80</td>
</tr>
<tr>
<td>Vehicle</td>
<td>846</td>
<td>77</td>
<td>0.08</td>
<td>0.50</td>
<td>13987</td>
<td>143</td>
<td>0.60</td>
</tr>
<tr>
<td>Adult</td>
<td>48842</td>
<td>115</td>
<td>0.10</td>
<td>0.50</td>
<td>932</td>
<td>6</td>
<td>0.77</td>
</tr>
</tbody>
</table>

was produced. CBA pruned the CARs to between 6 and 143 rules and achieves an accuracy (in sample testing) of typically around 90%. Only difficult datasets like Pima, Vehicle and Adult have worse results.

To compare run time, we conducted experiments on a standard laptop with an Intel Core i5-8250U CPU @ 1.60GHz with 4 cores and 8GB of RAM running R version 3.6.1 on Ubuntu 19.10. The package versions used for the comparison are: arc: 1.2, rCBA: 0.4.3, arulesCBA: 1.1.5. We disabled automatic threshold tuning. To remove the effect of random system load, we executed each algorithm ten times on each dataset and report the average execution time. The results are summarized in Table 3. arc produces the longest run times due to its pure R implementation. For Adult, the largest dataset arc ran out of memory. rCBA executes faster than arc. Both M2 and parallel execution using multi-core support in Java only improve the run time for the largest dataset. However, there the improvement is quite significant, reducing the run time to a third. arulesCBA’s M2 implementation is on average the fastest while the M2 implementation’s performance deteriorates on larger datasets.

Since many datasets of interest are typically larger then the standard datasets, we perform additional experiments to assess run time sensitivity for the number of input rules and the dataset size. For the experiments, we use the Lymph dataset. For assessing sensitivity to ruleset size, we oversample the dataset to 500 transactions and mine CARs with a minimum support of 0.05, a minimum confidence of 0.5 and a maximal rule length of 10. This results in more than 100000 rules. We then evaluate run time for building classifiers from the first 100, 1000, 10000, and 100000 mined rules. The results are shown in Figure 2(a). We see that M2 is generally slower then the corresponding M1 implementations. This might be due to the fact that the tested implementation hold all data in main memory, while M2 was designed for situations where the data does not reside in main memory. However, parallel execution helps rCBA’s M2 implementation. arulesCBA’s M1 implementation is on average the fastest.

To assess the sensitivity to dataset size, we fix the ruleset size to 500 and increased the dataset size by oversampling every round by a factor of 2. In Figure 2(b), we see a similar result to the sensitivity to the number of rules. Parallel execution in rCBA helps both algorithms and arulesCBA’s
Figure 2: Comparison of the run time of different implementations on an oversampled Lymphography dataset

M1 implementation is the fastest. All packages are integrated with the arules infrastructure, where arulesCBA has the most consistent integration. arc and rCBA offer automatic threshold tuning, which will help users with applying associative classification for practical applications.

Conclusion

In this paper, we reviewed associative classifiers based on the CBA algorithm. While the algorithm is cited in many papers about classifiers based on association rule mining, there are only very few implementations available. This paper discussed three recent implementations in R packages. Due to the differences in implementation language (R, C, and Java) and additional implemented features, each of the packages has its strengths. We hope that this review and the provided examples help users to experiment with associative classifiers and that the packages will be used by the research community to develop new methods.

Acknowledgments

Tomas Kliegr was supported by long term institutional support of research activities by Faculty of Informatics and Statistics, University of Economics, Prague.

Ian Johnson was supported by the Goldwater Foundation and the President’s Scholars program at Southern Methodist University, Dallas, TX, USA.
Bibliography


J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery, 8(1):53–87, Jan. 2004. ISSN 1384-5810. [p8, 9]


J. Kuchar. *rCBA: CBA Classifier for R*, 2018. URL https://CRAN.R-project.org/package=rCBA. R package version 0.4.3. [p1, 8]


Michael Hahsler
Office of Information Technology and Department of Engineering Management, Information, and Systems
Bobby B. Lyle School of Engineering
Southern Methodist University
P. O. Box 750123, Dallas, TX 75275, USA
mhahsler@lyle.smu.edu

Ian Johnson
Google,
Boulder, CO, USA
ianjjohnson@icloud.com

Tomáš Kliegr
Department of Information and Knowledge Engineering
Faculty of Informatics and Statistics
University of Economics, Prague
Winston Churchill Sq. 4, Prague, Czech Republic
ORCiD https://orcid.org/0000-0002-7261-0380
first.last@vse.cz

Jaroslav Kuchař
Web Intelligence Research Group