Quantitative CBA: Small and Comprehensible Association Rule Classification Models

Tomáš Kliegr*

School of Electronic Engineering and Computer Science, Queen Mary University of London, United Kingdom, and Faculty of Informatics and Statistics, VSE, Czech Republic.

November 29, 2017

Abstract

Quantitative CBA is a postprocessing algorithm for association rule classification algorithm CBA (Liu et al, 1998). QCBA uses original, undiscretized numerical attributes to optimize the discovered association rules, refining the boundaries of literals in the antecedent of the rules produced by CBA. Some rules as well as literals from the rules can consequently be removed, which makes the resulting classifier smaller. One-rule classification and crisp rules make CBA classification models possibly most comprehensible among all association rule classification algorithms. These viable properties are retained by QCBA. The postprocessing is conceptually fast, because it is performed on a relatively small number of rules that passed data coverage pruning in CBA. Benchmark of our QCBA approach on 22 UCI datasets shows average 53% decrease in the total size of the model as measured by the total number of conditions in all rules. Model accuracy remains on the same level as for CBA.

1 Introduction

Current rule learning approaches can be divided into two categories depending on how they learn rules and process numerical attributes: inductive rule learning, typically based on a variation of separate-and-conquer approach natively supporting numerical attributes, and association rule-based classification approaches, which work only on nominal data. Largely owing to this restriction, association rule-based algorithms can be very fast on datasets with many instances and high dimensions.

In this paper, we focus on Association Rule-based Classification (ARC) on quantitative data. Current mainstream ARC approaches can be applied on data with numerical attributes, but only if these are discretized prior to mining. The disconnection between discretization and model building is a source

*first.last@vse.cz

of inefficiencies in the resulting classifier – rule boundaries are not fit to the original continuous data, resulting in loss of accuracy and redundancies. While recently several ARC approaches that support numerical data have been proposed, these produce fuzzy association rules, containing fuzzy item sets that deteriorate comprehensibility of the model.

In this paper, we propose a postprocessing algorithm for ARC classification algorithm CBA [20], which reverts to the original attribute space to "edit" discovered association rules, refining the scope of literals (conditions, attribute-value pairs) in the antecedent of the rules. As a consequence, the fit of the individual rules to data improves, rendering some of the rules and attributes redundant. These can be removed, making the resulting classifier smaller. CBA models are ordered rule lists with several properties that make them comprehensible, such as one-rule classification and crisp rules. Our preprocessing retains these. The postprocessing is conceptually fast, because it is performed on a relatively small number of rules that passed the data coverage pruning in CBA.

This paper is organized as follows. The following related work section discusses the association rule classification and quantitative association rule learning and their current limitations. Section 3 presents the Quantitative CBA (QCBA) framework. Section 4 presents the experimental validation. The conclusions summarize the contributions of our approach, discuss ideas for future work and point to the publicly available implementation of the framework.

2 Related Work

Separate-and-conquer strategy [10] is likely the most commonly approach to rule learning. It provides a basis for the seminal RIPPER algorithm [5] as well as for the state-of-the-art FURIA algorithm [15]. Association rule learning is algorithmically different approach, which was originally designed to discover interesting patterns in very large and sparse instance spaces [2]. It yields a set of conjunctive rules that correspond to high density regions in the data. Unlike in the typical separate-and-conquer approach, cardinal features need to be discretized prior to the execution of association rule learning and converted along with all nominal attributes to binary-valued features. The resulting rules correspond to hypercubes with boundaries aligned to the discretization breakpoints. This impaired precision is offset by computationally efficiency on high-dimensional data, allowing association rule learning to succeed where other approaches fail [9, p. 505].

2.1 Steps to Build an ARC Classifier

Association rule learning was adopted also for classification several years after its conception in the early 90's. The first Association Rule Classification (ARC) algorithm dubbed CBA (Classification based on Associations) was introduced in 1998 [20]. While there were multiple follow-up algorithms providing marginal improvements in classification performance (e.g. CPAR [29] or CMAR

[18]), the structure of most ARC algorithms follows that of CBA [28]: 1. learn classification association rules, 2. prune the set of rules, 3. classify new objects.

2.1.1 Rule Learning

In the first step of an ARC framework, standard association rule learning algorithms such as Apriori [2] or FP-growth [13] are used to learn conjunctive classification rules from data. The mining setup is constrained or the algorithms adapted so that only the target class can occur in the consequent of the rules.

The output of association rule learning algorithms is determined typically by two parameters: minimum confidence and support thresholds. Let us briefly remind the definition of these two metrics. The confidence of a rule is defined as $a/(a+b)$, where a is the number of correctly classified objects, i.e. those matching rule antecedent as well rule consequent, and b is the number of misclassified objects, i.e. those matching the antecedent, but not the consequent. The support of a rule is defined as a/n, where n is the number of all objects (relative support), or simply as a (absolute support).

The main obstacles for a straightforward use of the discovered association rules as a classifier is the excessive number of rules discovered even on small datasets, and the fact that contradicting rules are generated. ARC algorithms contain a rule pruning step, which significantly reduces the number of rules, and define how the situation when one object is matched by multiple rules is handled.

2.1.2 Pruning

A qualitative review of rule pruning algorithms used in ARC is presented in [27, 28]. The most commonly used method according to these survey papers is *data coverage pruning*. This type of pruning processes the rules in the order of their strength, removing transactions (instances, objects) that the rule matches from the database. 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": the algorithm replaces all rules below the current rule if a default rule inserted at that place would reduce the number of errors. Default rule is a rule with empty antecedent, which ensures that a query instance is always classified even if it is not matched by any other rule in the classifier.

The effect of pruning on the size of the rule list is reported in [20], which presents evaluation on 26 UCI datasets. To illustrate the effect data coverage pruning in the CBA algorithm: the average number of rules per dataset without pruning was 35,140; with pruning the average number of rules was reduced to 69 without effectively impacting accuracy.

2.1.3 Classification

The original CBA algorithm performs "one rule" classification. First, rules are sorted according to the following criteria: 1. confidence, 2. support, 3. rule length

(shorter rule is placed higher). Instance is assigned the class in the consequent of the first rule with antecedent matching the instance in the ordered list of rules. The advantage of one rule classification is that it is easily understandable, which provides advantages in some applications [17]. In the effort to improve classifier accuracy, the successors of CBA such as CPAR combine multiple rules to perform classification.

2.2 Why CBA as Base Algorithm for Quantitative ARC?

The main benefit of using a rule-based classifier, as opposed to state-of-the-art sub-symbolic method such as a deep neural network should be the comprehensibility of the rule-based model, combined with fast execution on large and sparse datasets and accuracy comparable to state-of-the-art "black-box" classification models. Individual ARC algorithms meet these aspirations to a different degree. Table 1 presents a comparison between ten most well known ARC algorithms (and closely related approaches) in terms of key comprehensibility metrics, accuracy and performance.

We selected CBA as a basis for our work, since as follows from Table 1 it produces more comprehensible models than any of its successors while maintaining high accuracy and fast execution times. The difference between accuracy of CBA model and accuracy of state-of-the-art ARC algorithms such as FARC-HD is very small. In terms of accuracy, CBA is outperformed only by FARC-HD (by 4%) and CPAR (by 2%). However, CPAR has 4x times more rules on the output and less comprehensible multi-rule classification. While FARC-HD outperforms CBA in terms of accuracy, and even more so in its evolved version FARC-HD-OVO [7]. However, this algorithm is more than 100x slower than CBA and produces less comprehensible fuzzy rules.

In addition to criteria in Table 1, CBA has also the advantage that it uses standard (constrained) association rule learning in the first step. This makes work on postprocessing CBA output "future-proof", since the performance of CBA can be improved by replacing apriori with another association rule learning algorithm such as FP-Growth, which [11] report to be faster on most problems.

2.3 Quantitative Association Rule Learning

The proposed QCBA algorithm focuses on support for quantitative attributes in association rule classification. It does not follow the fuzzy approach, because, as noted earlier, it results in impaired comprehensibility of the model. To our knowledge, there is no prior work on creating ARC algorithms composed of crisp rules that support numerical attributes. However, there has been work on learning standard association rules (as a nugget discovery rather than classification task) from numerical data.[1]

[1]Note that there is a difference between (quantitative) association rule mining, which is exploratory data mining task aimed at discovering interesting patterns in data, and (quantitative) association rule classification, which aims at building understandable classification (predictive) models.

Table 1: Comparison between CBA and other association rule (or closely related) classifiers. *single* refers to single rule classification, *crisp* to whether the rules comprising the classifier are crisp (as opposed to fuzzy), *det.* to whether the algorithm is deterministic with no random element such as genetic optimization, *assoc* corresponds to whether the method is based on association rules, *acc*, *rules* and *time* is average accuracy, average rule count and average time across 26 datasets as reported by [3]. * indicates that the algorithm did not process all datasets

algorithm	year	single	crisp	det	assoc	acc	rules	time
CBA [20]	1998	yes	yes	yes	yes	.80	185	35s
CBA 2 [21]	2001	yes	yes	yes	yes	.79	184	2 m
2SLAVE [12]	2001	no?	no	no	no	.77	16	22m
CMAR [19]	2001	no	yes	yes	yes	.79	1419	6m
CPAR [29]	2003	no	yes	yes	yes	.82	788	11s
LAFAR [14]	2003	no	no	no	yes	.75*	47*	5h*
FH-GBML [16]	2005	no	no	no	no	.77	11	3h
CFAR [4]	2008	yes	no	yes	yes	.71*	47*	17m*
SGERD [22]	2008	no?	no	no	no	.74	7	3s
FARC-HD [3]	2011	no?	no	no	yes	.84	39	1h 20m

Multiple quantitative association rule learning algorithms have been proposed (cf. [1] for a recent review). Two representative and widely referenced approaches include the QuantMiner [25] and NAR-Discovery [26].

The earlier proposed QuantMiner is an evolutionary algorithm, which optimizes a multi-objective fitness function that combines support and confidence. The essence of QuantMiner is that a number of seed rules is evolved using standard evolutionary operators, where mutation corresponds to increase or decrease of lower/upper bound of a rule.

NAR-Discovery takes a different, two stage approach. Similarly to QCBA, set of "coarse" association rules is generated on prediscretized data with standard association rule generation algorithms in the first stage. In the second stage, for each coarse grained rule, a number of refined-rules is generated using fine bins. The granularity of the fine as well as coarse bins is a parameter of the algorithm. One feature of NAR-Discovery is that it produces at least one order of magnitude more rules than QuantMiner.

Comparison between our QCBA framework to NAR-Discovery and QuantMiner is summarized in Table 2. Additional justifications for individual values in the table: 1. classification models: neither QuantMiner or NAR-Discovery were designed for classification, 2. deterministic: QuantMiner is an evolutionary algorithm, 3. number of rules: too many generated rules is one of the biggest issues facing association rule generation algorithms, neither NAR-Discovery nor QuantMiner contain procedures to limit number of rules, while QCBA contains several iterations of rule pruning, 4. precision of intervals: for QuantMiner the precision of the intervals depends on the setting of the evolutionary process

Table 2: Comparison with Quantitative Association Rule Generation Approaches

property	QCBA	NAR-D	QuantMiner
classification models	y	n	n
deterministic	y	y	n
number of rules	+++	+	++
precision of intervals	+++	+	++
externally set parameters	+++	++	+

and for NAR-Discovery on the discretization setting, QCBA generates interval boundaries exactly corresponding to values in the input continuous data. 5. externally set parameters[2]: NAR-Discovery requires two granularity settings, (fine/coarse), and QuantMiner requires number of parameters such as population size, mutation and crossover rate for the evolutionary process. QCBA does not require any externally set parameters (there are several optional parameters).

Finally, it should be noted that the comparison described above is not completely fair to the quantitative association rule learning algorithms, since they address different task than QCBA. Unlike QCBA these are unsupervised algorithms that do not have the class information available. This is exploited by QCBA, among others, to perform rule pruning, which allows to reduce the number of rules in an informed way.

3 QCBA Framework

Quantitative CBA (QCBA) is an association rule classification framework, which is designed as an extension of CBA. Classification workflow involving QCBA consists of two main components. The first component is a CBA implementation. The core of the framework is the second component, which post-processes the discovered rules comprising the CBA classifier in two phases. In the first phase, the fit of the individual rules to the training data is improved. This increases coverage of individual rules and reduces their length by removing redundant attributes. In the second phase, three types of rule pruning are performed to reduce the number of rules.

[2]In addition to confidence and support thresholds.

1. **Standard CBA classifier building**.
   (a) **Discretization of numeric fields in the dataset**. Any discretization technique can be used. CBA is typically used with discretization based on the Minimum Description Length Principle (MDLP) [8], which selects number of cut-points with the highest entropy gains.
   (b) **Discovery of candidate association rules.** Class association rules can be discovered using any base association rule learner (apriori, FP-Growth, etc.)
   (c) **Rules are sorted to create initial rule list**. The sort criteria are confidence, support and rule length.
   (d) **Data coverage pruning**. Any of the CBA variants (M1 or M2 introduced in [20]) can be used.
2. **Tuning of individual rules.**
   (a) **Refitting rules to value grid.** Literals originally aligned to borders of the discretized regions are refit to finer grid with steps corresponding to all unique attribute values appearing in the training data.
   (b) **Literal pruning**: Remove redundant literals from rules. Literal is considered redundant if its removal does not decrease rule confidence.
   (c) **Trimming.** Boundaries of literals in discovered rules are trimmed so that their borders do not contain regions covering zero training instances.
   (d) **Extension.** Ranges of literals in the body of each rule are extended. The extension is accepted only if it does not deteriorate rule confidence.
3. **Pruning of the optimized rule list.** Rules are resorted and since the regions they cover could change, another iteration of pruning is performed to remove rules made newly redundant:
   (a) **Data coverage pruning.** Once the rules have been extended, they match more objects, which can make some of the rules redundant, therefore data coverage pruning can be performed to remove some of the newly redundant rules.
   (b) **Default rule pruning.** All rules below the current rule are replaced by default rule if this reduces the number of errors on training data. Note that to preserve more rules for QCBA to work with, this pruning is skipped in CBA and is performed after QCBA tuned the rule list.
   (c) **Default rule overlap pruning.** Rules that classify into the same class as the default rule in the end of the classifier can be removed, if there is no other rule between the removed rule and the default rule that would change the classification of instances originally classified by the removed rule.

Algorithm 1 depicts the succession of tuning steps in QCBA and provides pointers to algorithms described in detail in the following subsections. Please note that an interactive tutorial visually demonstrating all QCBA tuning steps is referenced from referenced from `https://github.com/kliegr/qcba`.

Algorithm 1 QCBA *qcba()*

Require: *rules* – input rule list generated by CBA
Ensure: optimized *rules*

1: *rules* ← remove any rules with empty antecedent from *rules*. {CBA includes at least one such rule as default rule in the end of the list.}
2: **for** $rule \in rules$ **do** {process rules in the CBA sort order (Fig. 1)}
3: $rule \leftarrow refit(rule)$ {cf. Alg. 2}
4: $rule \leftarrow pruneAttributes(rule)$ {cf. Alg. 3}
5: $rule \leftarrow trim(rule)$ {cf. Alg. 4}
6: $rule \leftarrow extendRule(rule)$ {cf. Alg. 5}
7: **end for**
8: $rules \leftarrow postprune(rules)$ {cf. Alg. 8, postpruning adds a new default rule, if postpruning is disabled, QCBA ensures that default rule is added at this point.}
9: $rules \leftarrow drop(rules)$ {cf. Alg. 9 (transaction-based, Alg. 10 (range-based) version of default rule overlap pruning}
10: **return** *rules*

3.1 Refit

As a first step, QCBA *refits* the boundaries of the rule to a finer grid, which corresponds to unique attribute values actually appearing in the data (Algorithm 2). The refit operation is inspired by the way the C4.5 decision tree learning algorithm selects splitting points for numerical attributes [23].

Algorithm 2 Refit rule *refit()*

Require: r – input rule learnt on discretized training data
Ensure: rule r with refit literals

1: **for** $literal = (A, V) \in antecedent(r)$ **do** {V is a value range, e.g. interval [30;30) and A is attribute, e.g. humidity}
2: $\mathcal{A} \leftarrow$ all unique values appearing in training data in attribute A.
3: $left \leftarrow min(\mathcal{A} \cap V)$
4: $right \leftarrow max(\mathcal{A} \cap V)$
5: $r \leftarrow$ replace *literal* in r with new *literal = (A,[left,right])*
6: **end for**
7: **return** r

3.2 Literal pruning

The literal pruning step removes redundant literals (attribute-value pairs) from rules. Literal is considered redundant if its removal does not decrease rule confidence. Literal pruning is depicted in Algorithm 3.

Algorithm 3 Literal pruning *pruneLiterals()*

Require: r – input rule
Ensure: rule r with redundant attributes (literals) removed

```
1: attrRemoved ← FALSE
2: repeat
3:    for literal ∈ antecedent(r) do {Literals are iterated in arbitrary order}
4:       r' ← remove literal from r
5:       if confidence(r') ≥ confidence(r) then
6:          r ← r'
7:          attrRemoved ← TRUE
8:          break
9:       else
10:         attrRemoved ← FALSE
11:      end if
12:   end for
13: until attrRemoved = FALSE
14: return r
```

3.3 Trimming

Through the trimming operation, literal boundaries are shaved of any values that belong solely to instances that are misclassified by r (Algorithm 4).

Algorithm 4 Rule trimming *trim()*

Require: r – input rule
Ensure: rule r with trimmed literals

```
1: corrCovByR ← training instances covered and correctly classified by r
2: for literal = (A, V) ∈ antecedent(r) do {Literals are iterated in arbitrary order}
3:    corrCovByL ← training instances covered by literal
4:    distValsL ← distinct values training instances have in attribute A
5:    if size(distValsL) ≤ 1 then {Intervals restricted to single value are not permitted}
6:       continue
7:    end if
8:    distValsLinR ← distinct values of attribute A in corrCovByR
9:    V' ← [min(distValsLinR), max(distValsLinR)]
10:   r ← replace literal in r with new literal = (A, V')
11: end for
12: return r
```

3.4 Extension

The extension process is depicted in Algorithm 5. The ranges of literals in the body of each rule are attempted to be enlarged. The range of each literal is increased one literal and one boundary at a time. The extension is generally accepted only if it improves rule confidence. To overcome local minima, the extension process can provisionally accept drop in confidence compared to the seed rule.

First, for given seed rule r the algorithm retrieves all possible extensions. These are generated by Algorithm 6. The algorithm refers to the notion of

1. rule A is ranked higher if confidence of rule A is greater than that of rule B,
2. rule A is ranked higher if confidence of rule A is the same as confidence of rule B, but support of rule A is greater than that of rule B,
3. rule A is ranked higher if rule A has shorter antecedent (fewer conditions) than rule B.

Figure 1: Rule ranking criteria.

Crisp accept: rule is accepted if i) the support of the candidate does not drop below the original rule and ii) confidence improves at least by the predefined threshold.

1. IF $\Delta_{conf} \geq minImprovement$ **and** $\Delta_{supp} \geq 0$ then **true**
2. ELSE **false**

Conditional accept: rule is conditionally accepted if confidence improves at least by the predefined threshold.

1. IF $\Delta_{conf} \geq minCondImprovement$ then **true**
2. ELSE **false**

Figure 2: crispAccept($\Delta_{conf}, \Delta_{supp}, minImprovement$) and *conditionalAccept(*$\Delta_{conf}, minCondImprovement$).

direct extension presented in Definition 1. Candidate extensions are sorted according to criteria applied by CBA, which are depicted in Figure 1.

Definition 1 ***(direct extension of literal defined over cardinal attribute)*** *Let $l = (A, V)$ be a cardinal literal, and $V = \langle x_i, \ldots, x_j, \ldots, x_k \rangle$ a value range. A direct extension of l is a set E_l of up to two literals derived from l: higher direct extension and lower direct extension. A higher direct extension of l is a literal $l_H = (A_i, V')$, where $V' = \langle x_i, \ldots, x_j, \ldots, x_k, x_{k+1} \rangle$. A lower direct extension of l is a literal $l_L = (A_i, V')$, where $V' = \langle x_{i-1}, x_i, \ldots, x_j, \ldots, x_k \rangle$. If both higher and lower extension exist, E_l has two elements, if only one exists, E_l has one element, if none of these extensions exists, E_l is empty.*

On line 7, an extension is accepted if it meets criteria for crisp accept (Fig. 2 top), which is based on improvement in confidence over the last confirmed ex-

tension. If the extension does not meet one of these conditions, it can still be conditionally accepted on line 18 (Fig. 2 bottom). The conditional accept sets a direction, on which the algorithm "locks" the beam extension, and on lines 12-26 verifies, whether this direction will yield an unconditional accept, or not. The *getBeamExtension* procedure is depicted in Algorithm 7. Note that the extension can be accepted also if the extension does not cover any additional training object, i.e. the confidence remains unchanged as well as the support.

By default, extend is accepted if it does not deteriorate rule confidence, which corresponds to $minImprovement = 0$. This value can be increased if the user desires to reduce the number of extensions. As for the conditional accept process, by default all extensions in given direction are tried until all values are exhausted, which corresponds to $minCondImprovement = -1$. The user may wish to decrease this value to obtain faster failure of the conditional extension process, improving performance on datasets with many distinct values (cf. Subs. 4.4.3).

3.5 Postpruning

The previous steps affected individual rules, changing their coverage. The number of rules can now be reduced using adaptation of CBA's data coverage and default rule pruning (Algorithm 8). This will also add a default rule to the end of the rule list. We refer to this second iteration[3] of CBA as *postpruning*. Each rule is matched against the training data. If a rule does not correctly classify any object, it is discarded. Otherwise, the rule is kept. In any case, objects matching the rule are discarded. The data coverage pruning is combined with default rule pruning, which determines the rule with the lowest number of errors on training data if rules below it are replaced by a default rule, and performs this replace.

3.6 Default Rule Overlap Pruning

Default rule overlap pruning (*drop* for short) iterates through all rules classifying into the same class as the default rule. These rules all overlap with the default rule both in terms of coverage and class assigned and are thus candidates for pruning. They can be removed only if their removal will not change classification of instances *in training data/in the entire instance space* they correctly classify by rules that are between them and the default rule. We consider two versions of drop: *transaction-based* and *range-based*.

The transaction-based version, depicted in Algorithm 9, removes rule if there is no transaction in the training data, which would be misclassified as a result of the removal.

The *range-based* version analyzes overlaps in the range of literals between the pruning candidate and all the potentially clashing rules (rules classifying

[3]The result of the first iteration of data coverage pruning is the rule list on the input to QCBA. We obtained better results if default rule pruning is not performed during the first iteration (within CBA), since in this way QCBA is left with more rules to optimize.

Algorithm 5 Rule Extension *extendRule()*

Require: $train = \{x_l | l = 1, \ldots, n\}$ – n train objects, defined over m attributes: $\{A_i | i = 1, \ldots, m\}$, r – input rule, $minImprovement \in (-1, 1)$ with 0 as default, $minCondImprovement \in (-1, 0)$ with -1 as default
Ensure: extended rule r

```
1: curBest ← r
2: directExtensions ← getExtensions(r,train)
3: repeat
4:   extensionSuccessful ← false
5:   for cand ∈ directExtensions {Iteration in order according to criteria in Figure 1} do
6:     Δ_conf ← conf(cand) - conf(curBest), Δ_supp ← sup(cand) - sup(curBest)
7:     if crispAccept(Δ_conf, Δ_supp, minImprovement) then
8:       curBest ← cand, extensionSuccessful ← true
9:       break
10:    else if conditionalAccept(Δ_conf, minCondImprovement) then
11:      enlgmnt ← cand
12:      loop
13:        enlgmnt ← getBeamExtension(enlgmnt)
14:        if enlgmnt=∅ then
15:          break
16:        end if
17:        Δ_conf ← conf(enlgmnt) - conf(curBest), Δ_supp ← sup(enlgmnt) - sup(curBest)
18:        if crispAccept(Δ_conf, Δ_supp, minImprovement) then
19:          curBest ← enlgmnt, extensionSuccessful ← true
20:          break
21:        else if conditionalAccept(Δ_conf, minCondImprovement) then
22:          continue {Extension in conditional accept band}
23:        else
24:          break
25:        end if
26:      end loop
27:      if extensionSuccessful = true then
28:        break
29:      end if
30:    else
31:      continue {Improvement below conditional threshold, going to next candidate}
32:    end if
33:  end for
34: until extensionSuccessful = false
35: return curBest
```

to different class) below it. The pruning is confirmed only if the potentially clashing rules cover different "geometric" regions (Algorithm 10). Range-based pruning thus guarantees a solution that generalizes beyond the training data. Its potential disadvantage is that it removes less rules, since it is stronger than the transaction-based pruning.

Algorithm 6 Get Extensions *getExtensions()*

Require: $train = \{x_l | l = 1, \ldots, n\}$ – n train objects, defined over m attributes: $\{A_i | i = 1, \ldots, m\}$, rule r
Ensure: up to two extensions of rule r

```
extendedRules ← ∅
for literal ∈ body(r) do
  if type(literal)=nominal then {Nominal attributes are skipped}
    continue
  end if
  neighbourhood ← direct extension of literal in train {See Def. 1}
  for extendedLiteral ∈ neighbourhood do
    extRule ← replace literal in r with extendedLiteral
    extendedRules ← extendedRules ∪ extRule
  end for
end for
return extendedRules
```

Algorithm 7 Beam Rule Extension *getBeamExtension()*

[H!]
Require: *rule* $\{r\}$
Ensure: extended rule r or null

```
literal ← let r' be a rule from which r was created by direct extension by replacing l ∈ R
  by literal
if type(literal) = nominal then
  return ∅ {Extension is not applicable on nominal attributes}
else
  extendedLiteral ← direct extension of literal
  if extendedLiteral = ∅ then
    return ∅ {Direct extension not found}
  end if
  extRule ← replace literal in r with extendedLiteral
end if
return extRule
```

4 Experiments

In this section, we present evaluation of the QCBA framework on a number of standard datasets. The evaluation focuses on comparison with CBA in terms of accuracy, classifier size and runtime. All results were obtained using open source CBA and QCBA implementations available at `https://cran.r-project.org/web/packages/arc/` and `https://github.com/kliegr/qcba`. All experiments can be directly replicated using the evaluation framework published at `https://github.com/kliegr/arcBench`. In order to verify the correctness of our CBA implementation, the last subsection is devoted to comparison of results we obtained and those reported by the CBA authors.

Algorithm 8 Postpruning *postPruning()*

Require: rules – output of *extendRuleList()*, set of training instances T
Ensure: pruned *rules* (some elements of input rule list removed, default rule added)

```
 1:
 2: cutoffRule ← ∅
 3: cutoffClass ← most frequent class in T
 4: lowestTotalError ← |T| − |t ∈ T : class(t) = cutoffClass|
 5: totalErrorsWithoutDefault ← 0
 6: rules ← sort rules according to criteria in Fig. 1
 7: defClass ← most frequent class in T
 8: {Data coverage pruning}
 9: for all r ∈ rules do
10:    covered ← instances in T matched by antecedent(r)
11:    corrCovered ← instances in T matched by antecedent(r) and consequent(r)
12:    T := T \ covered {Remove instances covered by r from training data}
13:    if corrCovered=∅ then
14:       rules ← rules \ r {remove r from rules}
15:    else
16:       misclassified ← covered - corrCovered
17:       totalErrorsWithoutDefault ← totalErrorsWithoutDefault + misclassified
18:       defClass ← most frequent class in T
19:       defaultRuleError ← |T| − |t ∈ T : class(t) = defClass|
20:       totalErrorWithDefault ← defaultRuleError + totalErrorsWithoutDefault
21:       if totalErrorWithDefault<lowestTotalError then
22:          cutoffRule, lowestTotalError, cutoffClass ← r, totalErrorWithDefault, defClass
23:       end if
24:    end if
25: end for
26: {Default rule pruning}
27: rules ← remove all rules below cutoffRule from rules
28: rules ← append new default rule "{} → cutoffClass' at the end of rules
29: return rules
```

4.1 Datasets

University of California provides at `https://archive.ics.uci.edu` a set of publicly available datasets, which are commonly used for benchmarking machine learning algorithms. We chose 22 datasets to perform the evaluation. The selection criteria were a) at least one numerical predictor attribute, b) the dataset being previously used in evaluation of symbolic learning algorithms in one of the following seminal papers: [3, 15, 20, 24].

Details of the selected datasets are given in Table 3. Several datasets come from visual information processing or signal processing domains (ionosphere, letter, segment, sonar). The second strongly represented domain are medical datasets (colic, breast-w, diabetes, heart-statlog, lymph). Eleven datasets are binary classification problems, nine datasets are multinominal (more than two classes) and two datasets have ordinal class attribute (autos and labour).

All datasets contain cardinal attributes, which needed to be pre-discretized for CBA and QCBA. All numeric explanatory attributes with three or more distinct values were subject to discretization using the MDLP algorithm wrapped in our *arc* package. Other algorithms involved in the benchmark did not re-

Algorithm 9 Default Rule Overlap Pruning (Transaction-based) *drop-tr()*

Require: *rules*, set of training instances T
Ensure: pruned *rules* (some elements of *rules* removed)

```
1: defRule ← default (last) rule in rules
2: for all prunCand ∈ rules do
3:   if consequent(prunCand) ≠ class(defRule) or prunCand = defRule  then
4:     continue
5:   end if
6:   corrCovered ← instances in T correctly classified by prunCand
7:   nonEmptyIntersection ← FALSE
8:   for all candClash ∈ rules below prunCand in rules do
9:     if consequent(candClash) = class(defRule)  then
10:      continue
11:    end if
12:    candClashCovered ← instances in T matching antecedent(candClash)
13:    if candClashCovered ∩ corrCovered ≠ ∅ then
14:      nonEmptyIntersection ← TRUE
15:      break
16:    end if
17:  end for
18:  if nonEmptyIntersection = FALSE then
19:    rules ← rules \ prunCand
20:  end if
21: end for
22: return rules
```

quire prediscretization. The evaluation was performed using a 10-fold stratified cross-validation. All evaluations used exactly the same folds.

4.2 Experiment Setup

The CBA algorithm has three hyperparameters – minimum confidence, minimum support thresholds and the total number of candidate rules. In [20] it is recommended to use 50% as minimum confidence, 1% as minimum support. For our experiments, we used these thresholds. In [20] the total number of rules used was 80.000, however it was noted that the performance starts to stabilize already around 60.000 rules. According to our experiments, there is virtually no difference between the 80.000 and 50.000 threshold apart from the higher computation time for the former, therefore we used 50.000.[4] We also limited the maximum number of items per itemset to 5. The experiments were performed using our CBA implementation in the arc R package, which is available in CRAN and on GitHub.[5]

The QCBA algorithm does not have any mandatory thresholds. The extension process (Algorithm 5) contains two numeric parameters, which were left set to their default values *minImprovement=0* and *minCondImprovement=-1.*

[4] In our best setup, we observed less than 0.1% improvement in average accuracy and 5% increase in average rule count when the maximum number of rules was increased from 50.000 to 80.000.

[5] `https://github.com/kliegr/arc` and `https://cran.r-project.org/web/packages/arc/`

Algorithm 10 Default Rule Overlap Pruning (Range-based) *drop-ra()*

Require: *rules*, set of training instances T
Ensure: pruned *rules* (some elements of *rules* removed)

defRule ← default (last) rule in *rules*
for all *prunCand* ∈ *rules* **do**
  if *consequent(prunCand)* ≠ *class(defRule)* **or** *prunCand* = *defRule* **then**
    continue
  end if
  literals ← literals in *antecedent*(*prunCand*)
  attributes ← attributes appearing in *antecedent*(*prunCand*)
  clashingRuleFound ← FALSE
  for all *candClash* ∈ rules below *prunCand* in *rules* **do**
    if *consequent(candClash) = class(defRule)* **then**
      continue
    end if
    sharedAttributes ← *attributes*∩ attributes in *antecedent*(*candClash*)
    if *sharedAttributes* = ∅ **then**
      clashingRuleFound ← TRUE {No shared attribute with potentially disjunct ranges results in the two rules overlapping on a subset of the data space}
      break
    end if
    literalsInClashOnSharedAtt ← literals in *antecedent*(*candClash*) defined over attributes in *sharedAttributes*
    {if there is NO intersection on at least one of the shared attributes we have no CLASH}
    attLeastOneAttDisjunct ← FALSE
    for all *literalCC* = (A, V) ∈ *literalsInClashOnSharedAtt* **do**
      literal ← get literal in *antecedent*(*prunCand*), which is defined over attribute A
      if V has empty intersection with value range of *literal* **then**
        attLeastOneAttDisjunct ← TRUE
        break
      end if
    end for
    if *attLeastOneAttDisjunct* = FALSE **then**
      clashingRuleFound ← TRUE
    end if
  end for
  if *clashingRuleFound* = FALSE **then**
    rules ← *rules* \ *prunCand*
  end if
end for
return *rules*

These default values have natural explanations (cf. Subs. 3.4) and the tuning of these thresholds can be generally recommended only for improving runtime on larger datasets.

Preferably, QCBA should obtain on its input model built with CBA, but without default rule pruning not performed. Such variation of CBA is not reported to be separately described in [20] or in other prior research. Our CBA implementation was adapted to allow deactivation of default rule pruning.

Table 3: Overview of datasets involved in the benchmark. att. denotes number of attributes, inst. number of instances (objects), miss. whether or not the dataset contains missing observations.

dataset	att.	inst.	miss.	class	description
anneal	39	898	Y	nominal (6)	NA
australian	15	690	N	binary	credit card applications
autos	26	205	Y	ordinal (7)	riskiness of second hand cars
breast-w	10	699	Y	binary	breast cancer
colic	23	368	Y	binary	horse colic (surgical or not)
credit-a	16	690	Y	binary	credit approval
credit-g	21	1000	N	binary	credit risk
diabetes	9	768	N	binary	diabetes
glass	10	214	N	nominal (6)	types of glass
heart-statlog	14	270	N	binary	diagnosis of heart disease
hepatitis	20	155	Y	binary	hepatitis prognosis (die/live)
hypothyroid	30	3772	Y	nominal (3)	NA
ionosphere	35	351	N	binary	radar data
iris	5	150	N	nominal (3)	types of irises (flowers)
labor	17	57	Y	ordinal (3)	employer's contribution to health plan
letter	17	20000	N	nominal (26)	letter recognition
lymph	19	148	N	nominal (4)	lymphography domain
segment	20	2310	N	nominal (7)	image segment classification
sonar	61	208	N	binary	determine object based on sonar signal
spambase	58	4601	N	binary	spam detection
vehicle	19	846	N	nominal (4)	object type based on silhouette
vowel	13	990	N	nominal (11)	NA

4.3 Evaluation Methodology

We evaluated several variations of the QCBA setup. As a baseline, we use a CBA run with default parameters. The purpose of this evaluation is to show the effect on classification performance and the size of the model.

Classification performance is measured by accuracy, which is computed as $correct/N$, where *correct* is the number of correct predictions and N the total number of objects. All results are reported using ten fold cross validation with macro averaging. The average accuracy for all 22 datasets is reported as an indicative comparison measure. For a more reliable comparison, we included the won-tie-loss matrix, which compares two classifiers by reporting the number of datasets where the reference classifier wins, loses or the two classifiers perform equally well. We include p-value for the Wilcoxon signed test, which is recommended for comparison of classifiers over multiple datasets in the authoritative work of [6].

We use three metrics to measure the size of the model: average antecedent length (number of conditions in the rule), number of rules per model and average number of conditions per model computed as number of rules times average antecedent length.

Despite our implementation not being optimized for speed, we decided to

include benchmark indicating how much processing power the postprocessing by QCBA requires. The build times reported in the table were computed as an average of classifier learning time for 220 models (10 folds for each of the 22 datasets). In addition to the absolute run time, which can be very volatile across software and hardware platforms, we include the relative execution time with the CBA baseline being assigned the score of 1.0. The reported time includes discovery of candidate association rules, data coverage and default rule pruning.

4.4 Results

Summary of results is presented in Table 4, which includes baseline results for CBA and then seven different configurations for QCBA. This allows us to demonstrate the effect of all individual postprocessing steps comprising QCBA. Configuration #1 corresponds to refit tuning step being performed on top of CBA, configuration #2 to refit tuning step and literal pruning, etc. Configuration #6 and #7 correspond to the full QCBA with the difference whether transaction-based (#6) or range-based (#7) default rule overlap pruning (drop) was performed.

4.4.1 Accuracy

The QCBA setup which produces the highest accuracy while achieving maximum reduction in the size of the classifier is configuration #7. This includes all tuning steps, very closely followed by #5, which excludes the default overlap pruning. These configurations have the same average accuracy as CBA and surpass CBA what concerns the won-tie-loss metric: they win on 14 datasets while CBA wins on 7 datasets and there is a draw[6] on 1 dataset. The p-value for the Wilcoxon signed rank test indicates that the change in the won-tie-loss matrix is not significantly different compared to CBA for neither of the QCBA configurations. It should be noted that the p-value of 0.12 is close to the 10% level significance level. QCBA configurations #5 and #7 thus marginally improve on CBA results. The configuration number #6 performs exactly equally well as CBA winning on 11 datasets and loosing also on 11 datasets.

4.4.2 Classifier size

The models produced by the best-performing QCBA configuration #7 are smaller than CBA models: there is a reduction of 21% in the number of rules and 18% in the average number of conditions. These reductions combined amount to 35% reduction in the model size in terms of total number of conditions. Further reduction in model size can be achieved by the transaction variant of default overlap pruning (#6), which reduces the size of the model with the average of 133 conditions by 53% compared to the original CBA model (285 conditions) while incurring 1% drop in the average accuracy and tie (11-0-11) in terms of the won-tie-loss record.

[6]Draw occurs when the accuracies on given dataset rounded to 0.1 percent match.

Table 4: QCBA evaluation – aggregate results for 22 UCI datasets

configuration	cba	#1	#2	#3	#4	#5	#6	#7
refit		Y	Y	Y	Y	Y	Y	Y
literal pruning		-	Y	Y	Y	Y	Y	Y
trimming		-	-	Y	Y	Y	Y	Y
extension		-	-	-	Y	Y	Y	Y
postpruning		-	-	-	-	Y	Y	Y
def. rule overlap - tran.		-	-	-	-	-	Y	-
def. rule overlap - range		-	-	-	-	-	-	Y
wins/ties/losses vs CBA		14-1-7	15-0-7	12-0-10	11-0-11	14-1-7	11-0-11	14-1-7
P-value (Wilcoxon)		.34	.57	.73	.61	.12	.32	.12
accuracy (macro average)	.81	.81	.81	.81	.81	.81	.80	.81
avg conditions / rule	3.4	3.4	2.8	2.8	2.8	2.8	2.8	2.8
avg number of rules	84	92	92	92	92	66	48	65
avg conditions / model	285	311	260	260	260	184	133	184
build time [s] (median)	12	24	20	20	43	43	43	43
build time normalized	1.0	1.9	2.0	2.0	17.4	17.3	17.3	17.4

As follows from comparison of #5 and #7, the range-based pruning was ineffective on this collection of datasets.

4.4.3 Runtime

The results for runtime are reported in the last two rows of Table 4. It can be seen that the refit, literal pruning and trimming tuning take together roughly as much time on average as learning a CBA model. The most computationally intensive operation is extension. If we look at the median build times, we see that the QCBA prolongs CBA execution by factor of 3.5.

The discrepancy between median and average build times for QCBA can be explained by several datasets for which QCBA extension step takes excessive time to complete, which increases the average runtime and leaves median run time unaffected. Extension is particularly slow when there is a large number of distinct values in the dataset. The slowest datasets were segment, letter and spambase. The segment and letter datasets contain various image metrics and spambase word frequency attributes. Such datasets are not typical representatives of use cases, where interpretable machine learning models are required. Nevertheless, the evaluation of the runtime indicates that the computational optimization of the extension algorithm is one of the most important areas for further work.

4.5 Verification of Results

The official implementation by authors of CBA [20] is not publicly available. We used our implementation to obtain the baseline results for CBA to support the central assertion that MARC (QCBA) reduces model size of CBA classifiers while keeping accuracy unaffected. In order to verify that our implementation

Table 5: Comparison of our results (included as *baseline* in the table) with [20] (*Liu*). *acc* denotes accuracy, *rules* number of rules in the classifier, *con* number of conditions in rule antecedent

	CBA (baseline)			CBA (Liu)		QCBA (#5)			QCBA (#7)		
	acc	rules	con	acc	rules	acc	rules	con	acc	rules	con
anneal	.96	27	3.0	.98	34	.99	25	2.3	.99	25	2.3
australian	.85	109	4.0	.87	148	.82	42	3.8	.87	74	3.8
autos	.79	57	3.0	.79	54	.79	44	2.5	.78	50	2.5
breast-w	.95	51	2.8	.96	49	.95	20	2.7	.95	31	2.7
diabetes	.75	51	3.9	.75	57	.76	30	2.9	.77	40	3.0
glass	.71	28	3.9	.73	27	.69	22	2.8	.69	24	2.8
hepatitis	.79	32	3.9	.85	23	.82	22	3.0	.82	28	3.0
hypothyroid	.98	29	3.1	.98	35	.98	15	2.4	.99	16	2.5
ionosphere	.92	53	2.5	.92	45	.86	22	1.9	.88	40	1.9
iris	.92	6	2.0	.93	5	.93	4	1.1	.93	5	1.2
labor	.84	11	3.6	.83	12	.86	8	1.8	.88	11	1.6
lymph	.81	38	3.7	.80	36	.79	37	2.9	.79	37	2.9
sonar	.74	44	2.9	.76	37	.72	19	2.7	.77	35	2.8
vehicle	.69	147	3.9	.69	125	.69	79	3.6	.71	106	3.7
average	.84	49	3.3	.84	49	.83	28	2.6	.84	37	2.6

is correct, we compared accuracy and number of rules[7] reported in [20] for 14 datasets with results that we obtained. As noted earlier, we used nearly identical CBA setting as [20] reports.

Detailed results are present in Table 5. While there are small variations for individual datasets, the overall average accuracy for our CBA implementation and the official one is equal at 84%. Regarding number of rules, original CBA has on average 49 rules, which is also exactly the same number as for our baseline implementation.

The precise match of the results came as a surprise, because our implementation of CBA does not include the optional pessimistic pruning step used in [20], which is used in the first CBA phase when candidate association rules are generated.[8] According to results reported in [20], the absence of pessimistic pruning has no effect on classifier accuracy, which is congruent with our results. While our results also indicate that this pruning has no effect on the number of rules in the classifier, the account of its effects in [20] suggests that it could be an effective technique for reducing the time required to build the model.

In summary, comparison of our results with those reported by [20] confirms the conclusion that QCBA reduces the number of rules in CBA-built model while not negatively affecting the accuracy of the classifier.

[7] The average length of the rules was unfortunately not reported in [20].

[8] Note that [20] explicitly marks pessimistic pruning step in CBA as optional. We could thus use standard association rule learner to obtain candidate rules.

5 Conclusion

Conceptually, the association rule classification approach to building rule-based classifiers can be more efficient on large and sparse datasets than traditionally used separate-and-conquer approaches, because of the possibility to use highly-optimized frequent pattern mining algorithms to generate the initial rule list. The presented framework ameliorates one of the major drawbacks of association rules, the adherence of the rules comprising the classifier to the multidimensional grid created by discretization of numerical attributes.

While CBA [20] is the first association rule classification approach, it is still one of the best rule-based classification algorithms what concerns balance between comprehensibility of the model, predictive power and scalability. Numerous enhancements to CBA have been proposed since the seminal paper of [20]. According to our review, the modifications in all the succeeding association rule classification approaches negatively affect comprehensibility of the resulting rule-based model, yielding none or very small improvement in accuracy. In contrast, QCBA reduces the number of rules for all 22 datasets in our evaluation, while retaining the favourable properties of CBA, making CBA models smaller and thus more comprehensible. The accuracy of the resulting models is unimpaired or even improves on most datasets.

The mechanism by which QCBA works is largely based on recovering the information lost in prediscretizing quantitative attributes. All previous adaptations of association rule classification for numerical data known to the author were fuzzy approaches. For example, the state-of-the-art FARC-HD association rule classifier outputs rules with fuzzy regions, which makes the rules less comprehensible than the crisp rules output by CBA. QCBA is, to the author's knowledge, the first non-fuzzy association rule classification algorithm supporting quantitative attributes. QCBA reuses some of the central concepts in CBA, such as data coverage pruning, but introduces several new enhancements and pruning steps (especially trimming, extension, default rule overlap pruning). While similar algorithms may have been used in other symbolic learning algorithms, their use in the context of association rule learning and classification is – as to the author's knowledge – novel. QCBA design avoids introduction of new data-specific thresholds for the user to set or optimize, which constitutes a certain advancement over previous quantitative association rule learning approaches such as QuantMiner or NAR-Discovery. QCBA does, however, contain several parameters that can be changed to speed up model building.

The most imminent future work is improvement of the proposed algorithms in terms of scalability. This can include incorporation of the pessimistic pruning, using the M2 version of data coverage pruning proposed in [20] instead of the M1 version and optimization of the extension algorithm, which is according to the results of the runtime benchmark the biggest bottleneck on some datasets.

The implementation of the QCBA algorithm for the R environment, code for replication of results in this paper, interactive tutorial, and additional resources including is available under an open license at `http://github.com/kliegr/qcba`.

References

[1] Adhikary, D., Roy, S.: Trends in quantitative association rule mining techniques. In: Recent Trends in Information Systems (ReTIS), 2015 IEEE 2nd International Conference on, pp. 126–131. IEEE (2015)

[2] Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD, pp. 207–216. ACM Press (1993)

[3] Alcala-Fdez, J., Alcala, R., Herrera, F.: A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning. IEEE Transactions on Fuzzy Systems **19**(5), 857–872 (2011)

[4] Chen, Z., Chen, G.: Building an associative classifier based on fuzzy association rules. International Journal of Computational Intelligence Systems **1**(3), 262–273 (2008)

[5] Cohen, W.W.: Fast effective rule induction. In: Proceedings of the Twelfth International Conference on Machine Learning, pp. 115–123. Morgan Kaufmann (1995)

[6] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)

[7] Elkano, M., Galar, M., Sanz, J.A., Fernndez, A., Barrenechea, E., Herrera, F., Bustince, H.: Enhancing multiclass classification in farc-hd fuzzy classifier: On the synergy between n-dimensional overlap functions and decomposition strategies. IEEE Transactions on Fuzzy Systems **23**(5), 1562–1580 (2015). DOI 10.1109/TFUZZ.2014.2370677

[8] Fayyad, U.M., Irani, K.B.: Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In: 13th International Joint Conference on Uncertainly in Artificial Intelligence (IJCAI93), pp. 1022–1029 (1993)

[9] Friedman, J., Hastie, T., Tibshirani, R.: The elements of statistical learning: Data mining, inference, and prediction. Springer Series in Statistics ((2009)

[10] Fürnkranz, J.: Separate-and-conquer rule learning. Artificial Intelligence Review **13**(1), 3–54 (1999)

[11] Goethals, B., Zaki, M.J.: FIMI03: Workshop on frequent itemset mining implementations. In: 3rd IEEE ICDM Workshop on Frequent Itemset Mining Implementations, pp. 1–13 (2003)

[12] González, A., Pérez, R.: Selection of relevant features in a fuzzy genetic learning algorithm. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **31**(3), 417–425 (2001)

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

[14] Hu, Y.C., Chen, R.S., Tzeng, G.H.: Finding fuzzy classification rules using data mining techniques. Pattern Recognition Letters **24**(1), 509–519 (2003)

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

[16] Ishibuchi, H., Yamamoto, T., Nakashima, T.: Hybridization of fuzzy GBML approaches for pattern classification problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **35**(2), 359–365 (2005)

[17] Kliegr, T., Kuchař, J., Sottara, D., Vojíř, S.: Learning business rules with association rule classifiers. In: A. Bikakis, P. Fodor, D. Roman (eds.) Rules on the Web. From Theory to Applications: 8th International Symposium, RuleML 2014, Co-located with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18-20, 2014. Proceedings, pp. 236–250. Springer International Publishing, Cham (2014)

[18] Li, W., Han, J., Pei, J.: CMAR: Accurate and efficient classification based on multiple class-association rules. In: Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM '01, pp. 369–376. IEEE, Washington, DC, USA (2001)

[19] Li, W., Han, J., Pei, J.: CMAR: Accurate and efficient classification based on multiple class-association rules. In: Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, pp. 369–376. IEEE (2001)

[20] Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD'98, pp. 80–86. AAAI Press (1998)

[21] Liu, B., Ma, Y., Wong, C.K.: Classification using association rules: weaknesses and enhancements. Data mining for scientific applications **591** (2001)

[22] Mansoori, E.G., Zolghadri, M.J., Katebi, S.D.: Sgerd: A steady-state genetic algorithm for extracting fuzzy classification rules from data. IEEE Transactions on Fuzzy Systems **16**(4), 1061–1071 (2008)

[23] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)

[24] Quinlan, J.R.: Improved use of continuous attributes in c4. 5. Journal of artificial intelligence research **4**, 77–90 (1996)

[25] Salleb-Aouissi, A., Vrain, C., Nortet, C.: Quantminer: A genetic algorithm for mining quantitative association rules. In: IJCAI, vol. 7 (2007)

[26] Song, C., Ge, T.: Discovering and managing quantitative association rules. In: Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, pp. 2429–2434. ACM (2013)

[27] Thabtah, F.: Pruning techniques in associative classification: Survey and comparison. Journal of Digital Information Management **4**(3) (2006)

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

[29] Yin, X., Han, J.: CPAR: Classification based on predictive association rules. In: Proceedings of the SIAM International Conference on Data Mining, pp. 369–376. SIAM Press, San Franciso (2003)