Association Rule Classifiers

Tomáš Kliegr
PhD candidate

Multimedia and Vision Research Group
Queen Mary
University of London

Machine Learning and Modelling Seminar
at the Charles University in Prague
April 9, 2015
Classification based on associations

In detail description of the CBA algorithm. The presentation uses excerpts from the original pseudocode published by Liu et al (1998) in [1].

Business Rule CBA

- Simplified version of CBA
- The effect of higher rule expressiveness (disjunctions, negations) on classifier accuracy
- Effect of rule pruning

Monotonicity-exploiting Association Rule Classification

- On going work
- Limitations of CBA (and association rule classifiers in general)
- Proposed solution
- Experimental results
1. Rule Generator: *typically Apriori-like algorithm*

2. Classifier Builder
   1. Prune rules
   2. Sort rules

3. Predict
   1. Apply matching rules: *select either the top matching rule or all matching rules*

CBA - Bing Liu, Wynne Hsu, Yiming. Classification Based on Associations - Integrating Classification and Association Rule Mining. ACM KDD ’98 conference. AAAI

1st CARM algorithm

Follow up:
CMAR - Li, Wenmin, Jiawei Han, and Jian Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. Data Mining, 2001. ICDM’01, IEEE, 2001.


CPAR,...
Bing Liu, Wynne Hsu, Yiming. *Classification Based on Associations - Integrating Classification and Association Rule Mining*. KDD ’98 conference. AAAI

Implementations:
http://www.cs.uic.edu/~liub/
http://cgi.csc.liv.ac.uk/~frans/KDD/Software/CBA/cba.html

1. Rule Generator
   - Mining of Class Association Rules based on Apriori

2. Classifier Builder
   - M1 – many passes over the data
     1. Sort Rules (conf, supp, length)
     2. Data coverage pruning – many passes over data
     3. Default rule pruning

   - M2 – find best rule for each data case
     - Optimized version of data coverage pruning
Classification based on associations (CBA)

1. Rule Generator
   - Mining of Class Association Rules based on Apriori

2. Classifier Builder
   - M1 – many passes over the data
     1. Sort Rules (conf, supp, length)
     2. Data coverage pruning – many passes over data
     3. Default rule pruning

   - M2 – find best rule for each data case
     - Optimized version – slightly more than one pass over data

Both M1 and M2 preserve Condition 1 and Condition 2

CONDITION 1
Each training case is covered by the rule with the highest precedence over other rules covering the case.

CONDITION 2
Every rule in the classifier correctly classifies at least one training case.
CBA – Basic notions

- Item – (attribute, value)
- Item set – set of items
- Large itemset – itemset meeting minSupp threshold
- Input data: A relational table $D$ with $n$ attributes
  - Continuous attributes need to be discretized
- Let $I$ be the set of all items in $D$
- Let $Y$ be the set of class labels
- Let $X \rightarrow Y$, $X \subseteq I$, $Y \subseteq Y$ be a classification association rule (CAR)
  - Right-hand side of the association rule is restricted to the target attribute
  - Rule is associated with confidence and support
Classification based on associations (CBA)

1. Rule Generator
   • Mining of Class Association Rules based on Apriori
2. Classifier Builder
   • **M1 – many passes over the data**
     1. Sort Rules (conf, supp, length)
     2. Data coverage pruning – many passes over data
     3. Default rule pruning
   • M2 – find best rule for each data case
     • Optimized version – slightly more than one pass over data

Both M1 and M2 preserve Condition 1 and Condition 2

**CONDITION 1**
Each training case is covered by the rule with the highest precedence over other rules covering the case.

**CONDITION 2**
Every rule in the classifier correctly classifies at least one training case.
Generating the complete set of CARs

Two parameters: minimum support, minimum confidence

ruleitem: <condset, y>

where condset is a set of items, y

Y is a class label

where (A, 1) is an attribute value pair

Example 2-ruleitem

k-ruleitem

rule item whose condset has k items

frequent (large) rule item

a ruleitem with support above minSup
$F_1 = \{\text{large 1-ruleitems}\};$

$CAR_1 = \text{genRules}(F_1);$

$prCAR_1 = \text{pruneRules}(CAR_1);$  

\begin{verbatim}
for (k = 2; F_{k-1} \neq \emptyset; k++) do
    C_k = \text{candidateGen}(F_{k-1});
    for each data case d \in D do
        C_d = \text{ruleSubset}(C_k, d);
        for each candidate c \in C_d do
            c\text{.condsupCount}++;
            if d\text{.class} = c\text{.class} then c\text{.rulesupCount}++
    end
end
\end{verbatim}

$F_k = \{c \in C_k \mid c\text{.rulesupCount} \geq \text{minsups}\};$

$CAR_k = \text{genRules}(F_k);$

$prCAR_k = \text{pruneRules}(CAR_k);$  

\end{verbatim}

$CAR_s = \bigcup_k CAR_k;$

$prCAR_s = \bigcup_k prCAR_k;$

In the first pass, the algorithm computes the support of individual rule items and discards rule items which are infrequent.
CBA – Rule Generation (CBA-RG)

1. $F_1 = \{\text{large 1-ruleitems}\}$;  
2. $CAR_1 = \text{genRules}(F_1)$;  
3. $prCAR_1 = \text{pruneRules}(CAR_1)$;  
4. \textbf{for} ($k = 2$; $F_{k-1} \neq \emptyset$; $k++$) \textbf{do}  
   5. \hspace{1em} $C_k = \text{candidateGen}(F_{k-1})$;  
   6. \hspace{1em} \textbf{for} each data case $d \in D$ \textbf{do}  
      7. \hspace{2em} $C_d = \text{ruleSubset}(C_k, d)$;  
      8. \hspace{2em} \textbf{for} each candidate $c \in C_d$ \textbf{do}  
         9. \hspace{3em} $c.\text{condsupCount}++$;  
         10. \hspace{3em} \textbf{if} $d.\text{class} = c.\text{class}$ \textbf{then} $c.\text{rulesupCount}++$  
    11. \hspace{1em} \textbf{end}\text{if}$d.\text{class} = c.\text{class}$  
   12. \textbf{end}\text{for} each data case$ d \in D$  
13. $F_k = \{c \in C_k \mid c.\text{rulesupCount} \geq \text{minsup}\}$;  
14. $CAR_k = \text{genRules}(F_k)$;  
15. $prCAR_k = \text{pruneRules}(CAR_k)$;  
16. \textbf{end}\text{for} each data case$ d \in D$  
17. $CARs = \bigcup_k CAR_k$;  
18. $prCARs = \bigcup_k prCAR_k$;  

\textbf{Example 1-ruleitem}  
$\langle (A, 1) \ (B,1)\rangle, \ (\text{class}, 1)\rangle$  
- support = 20\% = 2/10  
- confidence = 66.7\% = 2/3  

$F_k$  
Denotes the set of \textbf{frequent} $k$-rule items  
The elements of this set have the following form:  
$\langle (\text{condset}, \text{condsupCount}), \ (y, \text{rulesupCount})\rangle$  
$\langle (\{ (A, 1) \ (B,1)\}), \ 3\rangle, \ ((\text{class}, 1), \ 2)\rangle$

Source: [1]
Generating the complete set of CARs

1. \( F_1 = \{ \text{large 1-ruleitems} \} \);
2. \( CAR_1 = \text{genRules}(F_1) \);
3. \( prCAR_1 = \text{pruneRules}(CAR_1) \);
4. \( \text{for } (k = 2; F_{k-1} \neq \emptyset; k++) \text{ do} \)
5. \( \quad C_k = \text{candidateGen}(F_{k-1}) \);
6. \( \quad \text{for each data case } d \in D \text{ do} \)
7. \( \quad \quad C_d = \text{ruleSubset}(C_k, d) \);
8. \( \quad \quad \text{for each candidate } c \in C_d \text{ do} \)
9. \( \quad \quad \quad c.\text{condsupCount}++ \);
10. \( \quad \quad \quad \text{if } d.\text{class} = c.\text{class} \text{ then } c.\text{rulesupCount}++ \)
11. \( \quad \quad \end{\text{for}} \)
12. \( \quad \end{\text{for}} \)
13. \( F_k = \{ c \in C_k \mid c.\text{rulesupCount} \geq minsup \} ; \)
14. \( CAR_k = \text{genRules}(F_k) \);
15. \( prCAR_k = \text{pruneRules}(CAR_k) \);
16. \( \end{\text{for}} \)
17. \( CARs = \bigcup_k CAR_k ; \)
18. \( prCARs = \bigcup_k prCAR_k ; \)

For all ruleitems with the same condset, the ruleitem with the highest confidence is chosen as the possible rule (random draw in case of a tie).

R1 <\{(A, 1), (B, 1)\}, (class, 1)>, ruleSupCount =2, condSupCount = 3
R2 <\{(A, 1), (B, 1)\}, (class, 2)>, ruleSupCount =1, condSupCount = 3

Source: [1]

Note: In the genRules step, the description in [1] is not entirely clear to me

We get one possible rule: R1 with confidence 67%
CBA – Rule Generation (CBA-RG)

1 \( F_1 = \{ \text{large 1-rule items} \} \);
2 \( CAR_1 = \text{genRules}(F_1) \);
3 \( prCAR_1 = \text{pruneRules}(CAR_1) \); \hspace{1cm} \text{optional pessimistic rule pruning as in C4.5 [5]}
4 \textbf{for} (k = 2; \( F_{k-1} \neq \emptyset \); k++) \textbf{do}
5 \hspace{1cm} C_k = \text{candidateGen}(F_{k-1}) ;
6 \textbf{for each data case} \( d \in D \) \textbf{do}
7 \hspace{1cm} C_d = \text{ruleSubset}(C_k, d) ;
8 \hspace{1cm} \textbf{for each candidate} \( c \in C_d \) \textbf{do}
9 \hspace{1.5cm} c.\text{condsupCount}++ ;
10 \hspace{1.5cm} \textbf{if} \( d.\text{class} = c.\text{class} \) \textbf{then} \( c.\text{rulesupCount}++ \)
11 \hspace{1.5cm} \textbf{end}
12 \hspace{1cm} \textbf{end}
13 \( F_k = \{ c \in C_k | c.\text{rulesupCount} \geq \text{minsup} \} \);
14 \( CAR_k = \text{genRules}(F_k) \);
15 \( prCAR_k = \text{pruneRules}(CAR_k) \); \hspace{1cm} \text{rule pruning}
16 \textbf{end}
17 \( CARs = \bigcup_k CAR_k \);
18 \( prCARs = \bigcup_k prCAR_k \);

Source: [1]

Pessimist pruning:
1. Try to remove one condition (item) from condset of \( r \)
2. The rule is pruned if the pessimistic error rate of the original rule is higher than that of the pruned rule.

Experimental results in [1] show that pessimistic pruning reduces number of rules in the classifier and has no effect on accuracy.
\begin{align*}
F_1 &= \{\text{large 1-ruleitems}\}; \\
CAR_1 &= \text{genRules}(F_1); \\
prCAR_1 &= \text{pruneRules}(CAR_1); \\
\text{for} \ (k = 2; F_{k-1} \neq \emptyset; k++) \ \text{do} \\
\quad &C_k = \text{candidateGen}(F_{k-1}); \\
\quad \text{for} \ \text{each data case } d \in D \ \text{do} \\
\quad \quad &C_d = \text{ruleSubset}(C_k, d); \\
\quad \quad \text{for} \ \text{each candidate } c \in C_d \ \text{do} \\
\quad \quad \quad &c.\text{condsupCount}++; \\
\quad \quad \quad \text{if} \ d.\text{class} = c.\text{class} \ \text{then} \ c.\text{rulesupCount}++; \\
\quad \end{align*}

end \\
F_k = \{c \in C_k \mid c.\text{rulesupCount} \geq minsup\}; \\
CAR_k = \text{genRules}(F_k); \\
prCAR_k = \text{pruneRules}(CAR_k); \\
\text{end} \\
CARs = \bigcup_k CAR_k; \\
prCARs = \bigcup_k prCAR_k;

\text{Source: [1]}
\[ F_1 = \{ \text{large 1-ruleitems} \}; \]
\[ CAR_1 = \text{genRules}(F_1); \]
\[ prCAR_1 = \text{pruneRules}(CAR_1); \]
\[ \text{for} \ (k = 2; \ F_{k-1} \neq \emptyset; \ k++) \ \text{do} \]
\[ C_k = \text{candidateGen}(F_{k-1}); \]
\[ \text{for} \ \text{each data case} \ d \in D \ \text{do} \]
\[ C_d = \text{ruleSubset}(C_k, d); \]
\[ \text{for} \ \text{each candidate} \ c \in C_d \ \text{do} \]
\[ c.\text{condsupCount}++; \]
\[ \text{if} \ d.\text{class} = c.\text{class} \ \text{then} \ c.\text{rulesupCount}++; \]
\[ \text{end} \]
\[ F_k = \{ c \in C_k \mid c.\text{rulesupCount} \geq \text{minsup} \}; \]
\[ CAR_k = \text{genRules}(F_k); \]
\[ prCAR_k = \text{pruneRules}(CAR_k); \]
\[ \text{end} \]
\[ CARs = \bigcup_k CAR_k; \]
\[ prCARs = \bigcup_k prCAR_k; \]

same principle as aprioriGen [4]

aprioriGen
It takes as argument the set of all frequent (k-1) itemsets. It returns a superset of the set of all large k-itemsets. These are candidate k-itemsets as they are possibly large

Source: [1]
Takes as argument the set of all large (k-1) itemsets and returns a superset of the set of all frequent k-itemsets.

- 1. join step
- 2. prune step
aprioriGen – join step

- **insert** into \( C_k \)
- **select** \( p\text{.item}_1, p\text{.item}_2, \ldots, p\text{.item}_{k-1}, q\text{.item}_{k-1} \)
- **from** \( F_{k-1} \ p, F_{k-1} \ q \)
- **where** \( p\text{.item}_1 = q\text{.item}_1, \ldots, p\text{.item}_{k-2} = q\text{.item}_{k-2}, p\text{.item}_{k-1} < q\text{.item}_{k-1} \)

<table>
<thead>
<tr>
<th>( F_{k-1} )</th>
<th>( C_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2,3}</td>
<td>{1,2,3,4}</td>
</tr>
<tr>
<td>{1,2,4}</td>
<td>{1,2,3,5}</td>
</tr>
<tr>
<td>{1,2,5}</td>
<td>{1,2,4,5}</td>
</tr>
<tr>
<td>{1,3,5}</td>
<td>{2,3,4,5}</td>
</tr>
<tr>
<td>{2,3,4}</td>
<td></td>
</tr>
<tr>
<td>{2,3,5}</td>
<td></td>
</tr>
<tr>
<td>{3,4,5}</td>
<td></td>
</tr>
</tbody>
</table>
**aprioriGen – join step**

```
insert into C_k
select p.item_1, p.item_2,..., p.item_{k-1}, q.item_{k-1}
from F_{k-1} p, F_{k-1} q
where p.item_1 = q.item_1,..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}
```
**aprioriGen – join step**

```sql
insert into C_k
select p.item_1, p.item_2,..., p.item_{k-1}, q.item_{k-1}
from F_{k-1} p, F_{k-1} q
where p.item_1 = q.item_1,..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}
```

<table>
<thead>
<tr>
<th>F_{k-1}</th>
<th>C_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2,3}</td>
<td>{1,2,3,4}</td>
</tr>
<tr>
<td>{1,2,4}</td>
<td>{1,2,3,5}</td>
</tr>
<tr>
<td>{1,2,5}</td>
<td>{1,2,4,5}</td>
</tr>
<tr>
<td>{1,3,5}</td>
<td>{2,3,4,5}</td>
</tr>
<tr>
<td>{2,3,4}</td>
<td></td>
</tr>
<tr>
<td>{2,3,5}</td>
<td></td>
</tr>
<tr>
<td>{3,4,5}</td>
<td></td>
</tr>
</tbody>
</table>
insert into C_k
select p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}
from F_{k-1} p, F_{k-1} q
where p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}
insert into \( C_k \)
select \( p.\text{item}_1, p.\text{item}_2, \ldots, p.\text{item}_{k-1}, q.\text{item}_{k-1} \)
from \( F_{k-1} p, F_{k-1} q \)
where \( p.\text{item}_1 = q.\text{item}_1, \ldots, p.\text{item}_{k-2} = q.\text{item}_{k-2}, p.\text{item}_{k-1} < q.\text{item}_{k-1} \)

<table>
<thead>
<tr>
<th>( F_{k-1} )</th>
<th>( C_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2,3}</td>
<td>{1,2,3,4}</td>
</tr>
<tr>
<td>{1,2,4}</td>
<td>{1,2,3,5}</td>
</tr>
<tr>
<td>{1,2,5}</td>
<td>{1,2,4,5}</td>
</tr>
<tr>
<td>{1,3,5}</td>
<td>{2,3,4,5}</td>
</tr>
<tr>
<td>{2,3,4}</td>
<td></td>
</tr>
<tr>
<td>{2,3,5}</td>
<td></td>
</tr>
<tr>
<td>{3,4,5}</td>
<td></td>
</tr>
</tbody>
</table>
Remove itemsets that can’t possibly have the possible support because there is a subset in it which doesn’t have the level of support i.e. not in the previous pass (k-1).

<table>
<thead>
<tr>
<th>$F_{k-1}$</th>
<th>$C_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2,3}</td>
<td>{1,2,3,4}</td>
</tr>
<tr>
<td>{1,2,4}</td>
<td>{1,2,3,5}</td>
</tr>
<tr>
<td>{1,2,5}</td>
<td>{1,2,4,5}</td>
</tr>
<tr>
<td>{1,3,5}</td>
<td>{2,3,4,5}</td>
</tr>
<tr>
<td>{2,3,4}</td>
<td></td>
</tr>
<tr>
<td>{2,3,5}</td>
<td></td>
</tr>
<tr>
<td>{3,4,5}</td>
<td></td>
</tr>
</tbody>
</table>

Itemset {1,3,4} not in $F_{k-1}$
\[ F_1 = \{ \text{large 1-ruleitems} \}; \]
\[ CAR_1 = \text{genRules}(F_1); \]
\[ prCAR_1 = \text{pruneRules}(CAR_1); \]
\[ \text{for } (k = 2; F_{k-1} \neq \emptyset; k++) \text{ do} \]
\[ C_k = \text{candidateGen}(F_{k-1}); \]
\[ \text{for each data case } d \in D \text{ do} \]
\[ C_d = \text{ruleSubset}(C_k, d); \]
\[ \text{for each candidate } c \in C_d \text{ do} \]
\[ c.\text{condsupCount}++; \]
\[ \text{if } d.\text{class} = c.\text{class} \text{ then } c.\text{rulesupCount}++; \]
\[ \text{end} \]
\[ \text{end} \]
\[ F_k = \{ c \in C_k \mid c.\text{rulesupCount} \geq \text{minsup} \}; \]
\[ CAR_k = \text{genRules}(F_k); \]
\[ prCAR_k = \text{pruneRules}(CAR_k); \]
\[ \text{end} \]
\[ \text{CARs} = \bigcup_k CAR_k; \]
\[ \text{prCARs} = \bigcup_k prCAR_k; \]

Source: [1]
\[ F_1 = \{ \text{large 1-ruleitems} \} ; \]
\[ CAR_1 = \text{genRules} (F_1) ; \]
\[ prCAR_1 = \text{pruneRules} (CAR_1) ; \]
\[ \textbf{for} (k = 2 ; F_{k-1} \neq \emptyset ; k++) \textbf{do} \]
\[ C_k = \text{candidateGen} (F_{k-1}) ; \]
\[ \textbf{for} \text{ each data case } d \in D \textbf{ do} \]
\[ C_d = \text{ruleSubset} (C_k, d) ; \]
\[ \textbf{for} \text{ each candidate } c \in C_d \textbf{ do} \]
\[ c.\text{condsupCount}++ ; \]
\[ \textbf{if} d.\text{class} = c.\text{class} \textbf{ then} c.\text{rulesupCount}++ \]
\[ \textbf{end} \]
\[ \textbf{end} \]
\[ F_k = \{ c \in C_k \mid c.\text{rulesupCount} \geq \text{minsup} \}; \]
\[ CAR_k = \text{genRules} (F_k) ; \]
\[ prCAR_k = \text{pruneRules} (CAR_k) ; \]
\[ \textbf{end} \]
\[ CARs = \bigcup_k CAR_k ; \]
\[ prCARs = \bigcup_k prCAR_k ; \]

This implies many scans of the database: for each data case, all candidate rules with matching condsets are found, and their support statistics are updated.

Candidate rule \( c \) has the following form:
\[ < (\text{condset}, \text{condsupCount}), (y, \text{rulesupCount}) > \]

Source: [1]
\begin{verbatim}
1  \( F_1 = \{ \text{large 1-ruleitems} \}; \)
2  \( CAR_1 = \text{genRules}(F_1); \)
3  \( prCAR_1 = \text{pruneRules}(CAR_1); \)
4  \textbf{for} \( (k = 2; F_{k-1} \neq \emptyset; k++) \) \textbf{do}
5       \( C_k = \text{candidateGen}(F_{k-1}); \)
6  \textbf{for} each data case \( d \in D \) \textbf{do}
7       \( C_d = \text{ruleSubset}(C_k, d); \)
8  \textbf{for} each candidate \( c \in C_d \) \textbf{do}
9       \( c.\text{condsupCount}++; \)
10  \textbf{if} \( d.\text{class} = c.\text{class} \) \textbf{then} \( c.\text{rulesupCount}++; \)
11  \textbf{end}
12 \textbf{end}
13  \( F_k = \{ c \in C_k \mid c.\text{rulesupCount} \geq \text{minsups} \}; \)
14  \( CAR_k = \text{genRules}(F_k); \)
15  \( prCAR_k = \text{pruneRules}(CAR_k); \)
16 \textbf{end}
17  \( CARs = \bigcup_k CAR_k; \)
18  \( prCARs = \bigcup_k prCAR_k; \)
\end{verbatim}

Only frequent rule items are retained.

Source: [1]
\begin{verbatim}
1  \( F_1 = \{ \text{large 1-ruleitems} \}; \)
2  \( CAR_1 = \text{genRules}(F_1); \)
3  \( prCAR_1 = \text{pruneRules}(CAR_1); \)
4  \textbf{for} (k = 2; F_{k-1} \neq \emptyset; k++) \textbf{do} \\
5  \hspace{1em} C_k = \text{candidateGen}(F_{k-1}); \\
6  \hspace{1em} \textbf{for} each data case \( d \in D \textbf{ do} \\
7  \hspace{2em} C_d = \text{ruleSubset}(C_k, d); \\
8  \hspace{2em} \textbf{for} each candidate \( c \in C_d \textbf{ do} \\
9  \hspace{3em} c.\text{condsupCount}++ \\
10 \hspace{3em} \textbf{if} d.\text{class} = c.\text{class} \textbf{then} c.\text{rulesupCount}++ \\
11 \hspace{2em} \textbf{end} \\
12 \hspace{1em} \textbf{end} \\
13  \hspace{1em} F_k = \{ c \in C_k \mid c.\text{rulesupCount} \geq minsup \}; \\
14  \hspace{1em} CAR_k = \text{genRules}(F_k); \\
15  \hspace{1em} prCAR_k = \text{pruneRules}(CAR_k); \\
16 \textbf{end} \\
\end{verbatim}

\( CARs = \bigcup_k CAR_k; \)

\( prCARs = \bigcup_k prCAR_k; \)

Source: [1]

CBA – Rule Generation (CBA-RG)

1. \[ F_1 = \{ \text{large 1-ruleitems} \}; \]
2. \[ CAR_1 = \text{genRules}(F_1); \]
3. \[ prCAR_1 = \text{pruneRules}(CAR_1); \]
4. \[ \text{for } (k = 2; \ F_{k-1} \neq \emptyset; \ k++) \text{ do} \]
5. \[ \quad C_k = \text{candidateGen}(F_{k-1}); \]
6. \[ \quad \text{for each data case } d \in D \text{ do} \]
7. \[ \quad \quad C_d = \text{ruleSubset}(C_k, d); \]
8. \[ \quad \quad \text{for each candidate } c \in C_d \text{ do} \]
9. \[ \quad \quad \quad c.\text{condsupCount}++; \]
10. \[ \quad \quad \quad \text{if } d.\text{class} = c.\text{class} \text{ then } c.\text{rulesupCount}++; \]
11. \[ \quad \quad \text{end}\]
12. \[ \quad \text{end}\]
13. \[ F_k = \{ c \in C_k \mid c.\text{rulesupCount} \geq \text{minsup} \}; \]
14. \[ CAR_k = \text{genRules}(F_k); \]
15. \[ prCAR_k = \text{pruneRules}(CAR_k); \]
16. \[ \text{end}\]
17. \[ CARs = \bigcup_k CAR_k; \quad \text{final set of CARs} \]
18. \[ prCARs = \bigcup_k prCAR_k; \quad \text{final set of CARs after pruning} \]

Source: [1]
In CBA-RG there are separate counters for condset and ruleitem. This allows to compute the confidence of the rule as rulesupCount/condsupCount.
Classification based on associations (CBA)

1. Rule Generator
   - Mining of Class Association Rules based on Apriori

2. Classifier Builder
   - **M1** – many passes over the data
     1. Sort Rules (conf, supp, length)
     2. Data coverage pruning – many passes over data
     3. Default rule pruning
   - **M2** – find best rule for each data case
     - Optimized version – slightly more than one pass over data

CONDITION 1
Each training case is covered by the rule with the highest precedence over other rules covering the case.

CONDITION 2
Every rule in the classifier correctly classifies at least one training case.

Both M1 and M2 preserve Condition 1 and Condition 2
CBA-Classifier Builder (CB M1)

1. \( R = \text{sort}(R); \)
2. \textbf{for} each rule \( r \in R \) in sequence \textbf{do}
3. \hspace{1em} \textit{temp} = \emptyset;
4. \hspace{1em} \textbf{for} each case \( d \in D \) \textbf{do}
5. \hspace{2em} \textbf{if} \( d \) satisfies the conditions of \( r \) \textbf{then}
6. \hspace{3em} store \( d.\text{id} \) in \textit{temp} and mark \( r \) if it correctly classifies \( d \);
7. \hspace{2em} \textbf{if} \( r \) is marked \textbf{then}
8. \hspace{3em} insert \( r \) at the end of \( C \);
9. \hspace{3em} delete all the cases with the ids in \textit{temp} from \( D \);
10. \hspace{3em} selecting a default class for the current \( C \);
11. \hspace{3em} compute the total number of errors of \( C \);
12. \hspace{1em} \textbf{end}
13. \textbf{end}
14. Find the first rule \( p \) in \( C \) with the lowest total number of errors and drop all the rules after \( p \) in \( C \);
15. Add the default class associated with \( p \) to end of \( C \), and return \( C \) (our classifier).

Rule ranking criteria
- Confidence
- Support
- Rule length
  (shorter is better)

Source: [1], naïve CBA-CB algorithm M1
R = sort(R);
for each rule r ∈ R in sequence do
    temp = ∅;
    for each case d ∈ D do
        if d satisfies the conditions of r then
            store d.id in temp and mark r if it correctly classifies d;
        if r is marked then
            insert r at the end of C;
        delete all the cases with the ids in temp from D;
        selecting a default class for the current C;
        compute the total number of errors of C;
    end
end
Find the first rule p in C with the lowest total number of errors and drop all the rules after p in C;
Add the default class associated with p to end of C, and return C (our classifier).

Source: [1], naïve CBA-CB algorithm M1
1. $R = \text{sort}(R)$;
2. for each rule $r \in R$ in sequence do
3.     $\text{temp} = \emptyset$;
4.     for each case $d \in D$ do
5.         if $d$ satisfies the conditions of $r$ then
6.             store $d$.id in $\text{temp}$ and mark $r$ if it correctly
7.                 classifies $d$;
8.         if $r$ is marked then
9.             insert $r$ at the end of $C$;
10.        delete all the cases with the ids in $\text{temp}$ from $D$;
11.        selecting a default class for the current $C$;
12.        compute the total number of errors of $C$;
13.    end
14. end
15. Find the first rule $p$ in $C$ with the lowest total number of errors and drop all the rules after $p$ in $C$;
16. Add the default class associated with $p$ to end of $C$, and return $C$ (our classifier).

Source: [1], naïve CBA-CB algorithm M1
CBA-Classifier Builder (CB M1)

1. \( R = \text{sort}(R); \)
2. \textbf{for} each rule \( r \in R \) in sequence \textbf{do} \nl 3. \hspace{1em} \textit{temp} = \emptyset; \nl 4. \hspace{1em} \textbf{for} each case \( d \in D \) \textbf{do} \nl 5. \hspace{2em} \textbf{if} \( d \) satisfies the conditions of \( r \) \textbf{then} \nl 6. \hspace{3em} \text{store} \ d.\text{id} \ \text{in} \ \textit{temp} \ \text{and mark} \ r \ \text{if it correctly} \nl \hspace{3em} \text{classifies} \ d; \nl 7. \hspace{1em} \textbf{if} \ r \ \text{is marked} \ \textbf{then} \nl 8. \hspace{1em} \text{insert} \ r \ \text{at the end of} \ C; \nl 9. \hspace{1em} \text{delete all the cases with the ids in} \ \textit{temp} \ \text{from} \ D; \nl 10. \hspace{1em} \text{selecting a default class for the current} \ C; \nl 11. \hspace{1em} \textit{compute the total number of errors of} \ C; \nl 12. \end \nl 13. \textbf{end} \nl 14. \text{Find the first rule} \ p \ \text{in} \ C \ \text{with the lowest total number} \nl \text{of errors and drop all the rules after} \ p \ \text{in} \ C; \nl 15. \text{Add the default class associated with} \ p \ \text{to end of} \ C, \nl \text{and return} \ C \ \text{(our classifier).} \nl

Source: [1], naïve CBA-CB algorithm M1

Total number of errors made by the current set of rules in C and the default rule.
1 \( R = \text{sort}(R); \)
2 \textbf{for} each rule \( r \in R \) in sequence \textbf{do}
3 \hspace{1em} \textit{temp} = \emptyset;
4 \hspace{1em} \textbf{for} each case \( d \in D \) \textbf{do}
5 \hspace{2em} \textbf{if} \( d \) satisfies the conditions of \( r \) \textbf{then}
6 \hspace{3em} store \( d.\text{id} \) in \textit{temp} and mark \( r \) if it correctly classifies \( d \);
7 \hspace{1em} \textbf{if} \( r \) is marked \textbf{then}
8 \hspace{2em} insert \( r \) at the end of \( C \);
9 \hspace{2em} delete all the cases with the ids in \textit{temp} from \( D \);
10 \hspace{2em} selecting a default class for the current \( C \);
11 \hspace{2em} compute the total number of errors of \( C \);
12 \hspace{1em} \textbf{end}
13 \textbf{end}
14 \textbf{Find the first rule} \( p \) \textbf{in} \( C \) \textbf{with the lowest total number of errors} \textbf{and drop all the rules after} \( p \) \textbf{in} \( C \);
15 \textbf{Add the default class associated with} \( p \) \textbf{to end of} \( C \), \textbf{and return} \( C \) (our classifier).

“Default rule pruning”

Source: [1], naïve CBA-CB algorithm M1
$R = \text{sort}(R);$

for each rule $r \in R$ in sequence do

\hspace{1em} \text{temp} = \emptyset;

for each case $d \in D$ do

\hspace{2em} \text{if } d \text{ satisfies the conditions of } r \text{ then}

\hspace{3em} \text{store } d.\text{id} \text{ in } \text{temp} \text{ and mark } r \text{ if it correctly classifies } d;

\hspace{2em} \text{if } r \text{ is marked then}

\hspace{3em} \text{insert } r \text{ at the end of } C;

\hspace{3em} \text{delete all the cases with the ids in } \text{temp} \text{ from } D;

\hspace{3em} \text{selecting a default class for the current } C;

\hspace{3em} \text{compute the total number of errors of } C;

end

end

Find the first rule $p$ in $C$ with the lowest total number of errors and drop all the rules after $p$ in $C$;

Add the default class associated with $p$ to end of $C$, and return $C$ (our classifier).

Properties:

**CONDITION 1**

Each training case is covered by the rule with the highest precedence over other rules covering the case.

**CONDITION 2**

Every rule in $C$ correctly classifies at least one (remaining) training case.

Source: [1], naïve CBA-CB algorithm M1
CBA-Classifier Builder (CB M1)

1. \( R = \text{sort}(R); \)
2. for each rule \( r \in R \) in sequence do
3. \( \text{temp} = \emptyset; \)
4. for each case \( d \in D \) do
5. if \( d \) satisfies the conditions of \( r \) then
6. store \( d.id \) in \( \text{temp} \) and mark \( r \) if it correctly classifies \( d \);
7. if \( r \) is marked then
8. insert \( r \) at the end of \( C \);
9. delete all the cases with the ids in \( \text{temp} \) from \( D \);
10. selecting a default class for the current \( C \);
11. compute the total number of errors of \( C \);
12. end
13. end
14. Find the first rule \( p \) in \( C \) with the lowest total number of errors and drop all the rules after \( p \) in \( C \);
15. Add the default class associated with \( p \) to end of \( C \), and return \( C \) (our classifier).

Source: [1], naïve CBA-CB algorithm M1

CBA-CB M1 is simple but inefficient – many passes over the database.
CBA-Classifier Builder (CB M2)

- CBA M1 makes one pass over the remaining data for each rule
- CBA M2 makes “slightly more than one pass” over the data: finds the best rule in R cover each case d in D

  Stage 1 – *Find the highest precedence rule (cRule) that correctly classifies d, and also the highest precedence rule (wRule) that wrongly classifies d*

  Stage 2 – *Process data cases which in stage 1 were found to have wRule with higher precedence than cRule*

  Stage 3 – *Final rule selection and “default rule pruning”*
1. Rule Generator
   - Mining of Class Association Rules based on Apriori
2. Classifier Builder
   - M1 – many passes over the data
     1. Sort Rules (conf, supp, length)
     2. Data coverage pruning – many passes over data
     3. Default rule pruning
   - M2 – find best rule for each data case
     - Optimized version – slightly more than one pass over data

Both M1 and M2 preserve Condition 1 and Condition 2

**CONDITION 1**
Each training case is covered by the rule with the highest precedence over other rules covering the case.

**CONDITION 2**
Every rule in the classifier correctly classifies at least one training case.
\begin{verbatim}
1  Q = ∅; U = ∅; A = ∅;
2  for each case \( d \in D \) do
3      cRule = maxCoverRule(\( C_c \), \( d \));
4      wRule = maxCoverRule(\( C_w \), \( d \));
5      U = U \cup \{cRule\};
6      if cRule > wRule then
7          Q = Q \cup \{cRule\};
8          mark cRule;
9      else
10         A = A \cup \langle d.id, d.class, cRule, wRule \rangle
11  end
\end{verbatim}

Find the highest precedence rule that covers \( d \).
\( C_c \) is the set of rules having the same class as \( d \).

\textit{cRule} ... the highest precedence rule that correctly classifies \( d \)
CBA-CB M2  Stage 1

\[
\begin{align*}
Q &= \emptyset; U = \emptyset; A = \emptyset; \\
\text{for each case } d \in D & \quad \text{do} \\
& \quad cRule = \text{maxCoverRule}(C_c, d); \\
& \quad wRule = \text{maxCoverRule}(C_w, d); \\
& \quad U = U \cup \{cRule\}; \\
& \quad cRule.classCasesCovered[d.class]++; \\
& \quad \text{if } cRule \succ wRule \text{ then} \\
& \quad \quad Q = Q \cup \{cRule\}; \\
& \quad \quad \text{mark } cRule; \\
& \quad \text{else } \quad A = A \cup <d.id, d.class, cRule, wRule> \\
\end{align*}
\]

Finds the highest precedence rule that covers \(d\).

- \(C_w\) is the set of rules having different class than \(d\).

\text{wRule ... the highest precedence rule that incorrectly classifies } d
1 \quad Q = \emptyset; \ U = \emptyset; \ A = \emptyset;
2 \quad \textbf{for each case } d \in D \ 	extbf{do}
3 \quad \quad c\text{Rule} = \text{maxCoverRule}(C_c, d);
4 \quad \quad w\text{Rule} = \text{maxCoverRule}(C_w, d);
5 \quad \quad U = U \cup \{c\text{Rule}\};
6 \quad \quad c\text{Rule}\.\text{classCasesCovered}[d\.\text{class}]++;
7 \quad \quad \textbf{if } c\text{Rule} \succ w\text{Rule} \ 	extbf{then}
8 \quad \quad \quad Q = Q \cup \{c\text{Rule}\};
9 \quad \quad \quad \text{mark } c\text{Rule};
10 \quad \quad \textbf{else } A = A \cup <d\.\text{id}, d\.\text{class}, c\text{Rule}, w\text{Rule}>$
11 \ 	extbf{end}$

Source: [1], CBA-CB algorithm M2

U is the set of all cRules.
1 \[ Q = \emptyset; \ U = \emptyset; \ A = \emptyset; \]
2 \textbf{for each case } d \in D \textbf{ do}
3 \hspace{1em} cRule = \text{maxCoverRule}(C_c, d);
4 \hspace{1em} wRule = \text{maxCoverRule}(C_w, d);
5 \hspace{1em} U = U \cup \{cRule\};
6 \hspace{1em} cRule.classCasesCovered[d.class]++;
7 \hspace{1em} \textbf{if cRule > wRule then}
8 \hspace{2em} Q = Q \cup \{cRule\};
9 \hspace{2em} \text{mark } cRule;
10 \hspace{1em} \textbf{else }
11 \hspace{2em} A = A \cup \langle d.id, d.class, cRule, wRule \rangle

For each cRule, the field classCasesCovered holds the number of cases it covers in each class.

Source: [1], CBA-CB algorithm M2
1 \[ Q = \emptyset; \quad U = \emptyset; \quad A = \emptyset; \]
2 \textbf{for} each case \[ d \in D \textbf{ do} \]
3 \hspace{1em} cRule = maxCoverRule(\[ C_c, d \]);
4 \hspace{1em} wRule = maxCoverRule(\[ C_w, d \]);
5 \hspace{1em} U = U \cup \{ cRule \};
6 \hspace{1em} cRule.classCasesCovered[d.class]++;
7 \hspace{1em} \textbf{if} cRule \succ wRule \textbf{ then}
8 \hspace{2em} Q = Q \cup \{ cRule \};
9 \hspace{2em} \text{mark } cRule;
10 \hspace{1em} \textbf{else} \quad A = A \cup <\text{d.id, d.class, cRule, wRule}> \]
11 \hspace{1em} \textbf{end}

Source: [1], CBA-CB algorithm M2
\begin{verbatim}
Q = \emptyset; U = \emptyset; A = \emptyset;
for each case d \in D do
    cRule = maxCoverRule(C_c, d);
    wRule = maxCoverRule(C_w, d);
    U = U \cup \{cRule\};
    cRule.classCasesCovered[d.class]++;
    if cRule > wRule then
        Q = Q \cup \{cRule\};
    mark cRule;
else
    A = A \cup <d.id, d.class, cRule, wRule>
end
\end{verbatim}

Q holds the set of cRules that have a higher precedence than their corresponding wRules.

Source: [1], CBA-CB algorithm M2
\begin{verbatim}
1 \( Q = \emptyset; \ U = \emptyset; \ A = \emptyset; \)
2 \textbf{for} each case \( d \in D \) \textbf{do}
3 \hspace{1em} cRule = \text{maxCoverRule}(C_c, d);
4 \hspace{1em} wRule = \text{maxCoverRule}(C_w, d);
5 \hspace{1em} U = U \cup \{cRule\};
6 \hspace{1em} cRule.classCasesCovered[d.class]++;
7 \hspace{1em} \textbf{if} cRule > wRule \textbf{then}
8 \hspace{2em} Q = Q \cup \{cRule\};
9 \hspace{2em} \text{mark} \ cRule;
10 \hspace{1em} \textbf{else} \ A = A \cup <d.id, d.class, cRule, wRule>
11 \textbf{end}
\end{verbatim}

The cRule is marked to denote it classifies the case correctly.

Source: [1], CBA-CB algorithm M2
\[
\begin{align*}
Q &= \emptyset; U = \emptyset; A = \emptyset; \\
\text{for each case } d \in D &\text{ do} \\
    cRule &= \text{maxCoverRule}(C_c, d); \\
    wRule &= \text{maxCoverRule}(C_w, d); \\
    U &= U \cup \{cRule\}; \\
    cRule\text{.classCasesCovered}[d\text{.class}] &= \text{++}; \\
\text{if } cRule \succ wRule &\text{ then} \\
    Q &= Q \cup \{cRule\}; \\
    \text{mark } cRule; \\
\text{else } A &= A \cup <d\text{.id}, d\text{.class}, cRule, wRule> \\
\text{end}
\end{align*}
\]

Source: [1], CBA-CB algorithm M2

Unfavourable case
If wRule is better ranked than cRule, a record is added to the “problem bin” A.

A is a data structure:
\(<\text{dID}, y, \text{cRule}, \text{wRule}>,\)
\(\text{dID} \ldots \text{id of the case d}\)
\(y \ldots \text{the class of d}\)
In stage 2, the algorithm processes the data cases stored in A: for these data cases, the highest precedence rule was wRule.

```plaintext
1  for each entry <dID, y, cRule, wRule> ∈ A do
2    if wRule is marked then
3       cRule.classCasesCovered[y]--;  
4       wRule.classCasesCovered[y]++;  
5    else
6       wSet = allCoverRules(U, dID.case, cRule);
7       for each rule w ∈ wSet do
8          w.replace = w.replace ∪ {<cRule, dID, y>};  
9          w.classCasesCovered[y]++;  
10         end  
11     Q = Q ∪ wSet
12    end
```

If wRule is marked, it means it also acts as a highest precedence cRule in at least one other case.

Source: [1], CBA-CB algorithm M2
In stage 2, the algorithm processes the data cases stored in $A$: for these data cases, the highest precedence rule was $w\text{Rule}$.

The algorithm accepts the error. The case $d$ will be classified by $w\text{Rule}$.

Since in stage 1, $d$ was counted under $c\text{Rule}$, the algorithm subtracts $d$ from the number of cases covered by $c\text{Rule}$, and increments the number of cases covered by $w\text{Rule}$.

Source: [1], CBA-CB algorithm M2, Stage 1
In stage 2, the algorithm processes the data cases stored in A: for these data cases, the highest precedence rule was \( w\text{Rule} \).

The algorithm accepts the error. The case \( d \) will be classified by \( w\text{Rule} \).

For case \( d \), both Condition 1 and Condition 2 are satisfied.

**CONDITION 1**
Each training case is covered by the rule with the highest precedence over other rules covering the case.

**CONDITION 2**
Every rule in \( C \) correctly classifies at least one (remaining) training case.
In stage 2, the algorithm processes the data cases stored in A: for these data cases, the highest precedence rule was $w_{\text{Rule}}$.

```plaintext
for each entry $<dID, y, cRule, w_{\text{Rule}}> \in A$ do
  if $w_{\text{Rule}}$ is marked then
    $cRule$.classCasesCovered[y]--;
    $w_{\text{Rule}}$.classCasesCovered[y]++;
  else
    $w_{\text{Set}} = \text{allCoverRules}(U, dID\text{-case}, cRule)$;
    for each rule $w \in w_{\text{Set}}$ do
      $w$.replace = $w$.replace $\cup \{<cRule, dID, y>\}$;
      $w$.classCasesCovered[y]++;
    end
  end
end
$Q = Q \cup w_{\text{Set}}$
```

Since $w_{\text{Rule}}$ is not marked, it does not act as a $c_{\text{Rule}}$ for another rule. However, there may be multiple higher precedence rules (than $c_{\text{Rule}}$) that covered and classify it incorrectly.

`allCoverRules()` returns all rules that wrongly classify $dID$ and have higher precedence than $c_{\text{Rule}}$. It processed only the rules in $U$, which is the set of all $c_{\text{Rules}}$.

Source: [1], CBA-CB algorithm M2
In stage 2, the algorithm processes the data cases stored in A: for these data cases, the highest precedence rule was \( wRule \).

```plaintext
for each entry \( <dID, y, cRule, wRule> \in A \) do
  if \( wRule \) is marked then
    \( cRule.classCasesCovered[y]--; \)
    \( wRule.classCasesCovered[y]++; \)
  else \( wSet = \) allCoverRules(\( U, dID.case, cRule \));
  for each rule \( w \in wSet \) do
    \( w.replace = w.replace \cup \{ <cRule, dID, y> \} \);
    \( w.classCasesCovered[y]++; \)
  end
  \( Q = Q \cup wSet \)
end
```

Since \( wRule \) is not marked, it does not act as a \( cRule \) for any instance. However, there may be multiple other higher precedence rules (than \( cRule \)) that covered and classify it incorrectly.

\( wSet \) is a subset of \( U \), which is the set of all rules that act as \( cRule \) for some instance. Rules in \( wSet \) may replace \( cRule \) when classifying the instance \( dID \). For each of these rules, we note which \( cRule \) and which instance is replaced.

Source: [1], CBA-CB algorithm M2
In stage 2, the algorithm processes the data cases stored in A:
for these data cases, the highest precedence rule was \texttt{wRule}.

```plaintext
1. for each entry \texttt{<dID, y, cRule, wRule> \in A} do
2.      if \texttt{wRule} is marked then
3.          \texttt{cRule.classCasesCovered[y]}--;
4.          \texttt{wRule.classCasesCovered[y]}++;
5.      else
6.          \texttt{wSet = allCoverRules(U, dID.case, cRule)};
7.          for each rule \texttt{w \in wSet} do
8.              \texttt{w.replace = w.replace \cup \{<cRule, dID, y\>} };
9.              \texttt{w.classCasesCovered[y]}++;
10.         end
11.      end
12. end
```

Since \texttt{wRule} is not marked, it does not act as a \texttt{cRule} for any instance. However, there may be multiple other higher precedence rules (than \texttt{cRule}) that covered and classify it incorrectly.

Indicates that the rule might cover the case \texttt{dID}.

Source: [1], CBA-CB algorithm M2
In stage 2, the algorithm processes the data cases stored in $A$: for these data cases, the highest precedence rule was $wRule$.

```
for each entry $<dID, y, cRule, wRule> \in A$ do
  if $wRule$ is marked then
    $cRule.classCasesCovered[y]--$;
    $wRule.classCasesCovered[y]++$;
  else $wSet = allCoverRules(U, dID.case, cRule);$;
  for each rule $w \in wSet$ do
    $w.replace = w.replace \cup \{<cRule, dID, y>\};$
    $w.classCasesCovered[y]++$;
  end
end
$Q = Q \cup wSet$
```

In Stage 1, $Q$ was set to hold $cRules$ that had a higher precedence than their corresponding $wRules$. Now $Q$ is extended with rules in $wSet$.

Source: [1], CBA-CB algorithm M2
In stage 3, the algorithm chooses the final set of rules.

```
1  classDistr = compClassDistri(D);
2  ruleErrors = 0;
3  Q = sort(Q);
4  for each rule r in Q in sequence do
5      if r.classCasesCovered[r.class] ≠ 0 then
6          for each entry <rul, did, y> in r.replace do
7              if the did case has been covered by a
8                previous r then
9                  r.classCasesCovered[y]--;
10             else  rul.classCasesCovered[y]--;
11             ruleErrors = ruleErrors + errorsOfRule(r);
12            classDistr = update(r, classDistr);
13            defaultClass = selectDefault(classDistr);
14            defaultErrors = defErr(defaultClass, classDistr);
15            totalErrors = ruleErrors + defaultErrors;
16        Insert <r, default-class, totalErrors> at end of C
17      end
18  end
19  Find the first rule p in C with the lowest totalErrors,
20      and then discard all the rules after p from C;
21  Add the default class associated with p to end of C;
22  Return C without totalErrors and default-class;
```
In stage 3, the algorithm chooses the final set of rules.

```plaintext
1  classDistr = compClassDistri(D);
2  ruleErrors = 0;
3  Q = sort(Q);
4  for each rule r in Q in sequence do
5      if r.classCasesCovered[r.class] ≠ 0 then
6         for each entry <rul, did, y> in r.replace do
7            if the did case has been covered by a
8               previous r then
9               r.classCasesCovered[y]--;
10          else  rul.classCasesCovered[y]--;
11         ruleErrors = ruleErrors + errorsOfRule(r);
12  classDistr = update(r, classDistr);
13  defaultClass = selectDefault(classDistr);
14  defaultErrors = defErr(defaultClass, classDistr);
15  totalErrors = ruleErrors + defaultErrors;
16  Insert <r, default-class, totalErrors> at end of C
17  end
18  end
19  Find the first rule p in C with the lowest totalErrors,
     and then discard all the rules after p from C;
20  Add the default class associated with p to end of C;
21  Return C without totalErrors and default-class;
```

Counts the number of training cases in each class in the initial training data.

Source: [1], CBA-CB algorithm M2
\begin{align*}
1. \text{classDistr} &= \text{compClassDistri}(D); \\
2. \text{ruleErrors} &= 0; \\
3. \text{Q} &= \text{sort}(\text{Q}); \\
4. \text{for each rule } r \text{ in } \text{Q} \text{ in sequence do} \\
5. & \quad \text{if } r.\text{classCasesCovered}[r.\text{class}] \neq 0 \text{ then} \\
6. & \quad \quad \text{for each entry } <\text{rul}, dID, y> \text{ in } r.\text{replace do} \\
7. & \quad \quad \quad \text{if the } dID \text{ case has been covered by a} \\
8. & \quad \quad \quad \quad \text{previous } r \text{ then} \\
9. & \quad \quad \quad \quad r.\text{classCasesCovered}[y]--; \\
10. & \quad \quad \else \quad rul.\text{classCasesCovered}[y]--; \\
11. & \quad \quad \text{ruleErrors} = \text{ruleErrors} + \text{errorsOfRule}(r); \\
12. & \quad \text{classDistr} = \text{update}(r, \text{classDistr}); \\
13. & \quad \text{defaultClass} = \text{selectDefault}(\text{classDistr}); \\
14. & \quad \text{defaultErrors} = \text{defErr}(\text{defaultClass}, \text{classDistr}); \\
15. & \quad \text{totalErrors} = \text{ruleErrors} + \text{defaultErrors}; \\
16. & \quad \text{Insert } <r, \text{default-class}, \text{totalErrors}> \text{ at end of } C \\
17. \text{end} \\
18. \text{end} \\
19. \text{Find the first rule } p \text{ in } C \text{ with the lowest } \text{totalErrors}, \\
20. \text{and then discard all the rules after } p \text{ from } C; \\
21. \text{Add the default class associated with } p \text{ to end of } C; \\
22. \text{Return } C \text{ without } \text{totalErrors} \text{ and } \text{default-class};
\end{align*}

Records the number of errors made so far on the training data

Source: [1], CBA-CB algorithm M2
1. $classDistr = \text{compClassDistri}(D)$;
2. $ruleErrors = 0$;
3. $Q = \text{sort}(Q)$;
4. for each rule $r$ in $Q$ in sequence do
5.   if $r.classCasesCovered[r.class] \neq 0$ then
6.     for each entry $<rul, dID, y>$ in $r.replace$ do
7.       if the $dID$ case has been covered by a previous $r$ then
8.         $r.classCasesCovered[y]--$;
9.       else $rul.classCasesCovered[y]--$;
10.      $ruleErrors = ruleErrors + \text{errorsOfRule}(r)$;
11.     $classDistr = \text{update}(r, \text{classDistr})$;
12.     $defaultClass = \text{selectDefault}(\text{classDistr})$;
13.     $defaultErrors = \text{defErr}(\text{defaultClass, classDistr})$;
14.     $totalErrors = ruleErrors + defaultErrors$;
15.     Insert $<r, \text{default-class, totalErrors}>$ at end of $C$
16.   end
17. end
18. Find the first rule $p$ in $C$ with the lowest $totalErrors$, and then discard all the rules after $p$ from $C$;
19. Add the default class associated with $p$ to end of $C$;
20. Return $C$ without $totalErrors$ and default-class;

Rule ranking criteria
- Confidence
- Support
- Rule length
  (shorter is better)

**CONDITION 1**
Each training case is covered by the rule with the highest precedence over other rules covering the case

Source: [1], CBA-CB algorithm M2
\begin{verbatim}
1  classDistr = compClassDistri(D);
2  ruleErrors = 0;
3  Q = sort(Q);
4  for each rule r in Q in sequence do
5      if r.classCasesCovered[r.class] ≠ 0 then
6          for each entry <rul, dID, y> in r.replace do
7              if the dID case has been covered by a
8                  previous r then
9                  r.classCasesCovered[y]--;
10             else  rul.classCasesCovered[y]--;
11          ruleErrors = ruleErrors + errorsOfRule(r);
12          classDistr = update(r, classDistr);
13          defaultClass = selectDefault(classDistr);
14          defaultErrors = defErr(defaultClass, classDistr);
15          totalErrors = ruleErrors + defaultErrors;
16          Insert <r, default-class, totalErrors> at end of C
17      end
18  end
19  Find the first rule p in C with the lowest totalErrors,
20     and then discard all the rules after p from C;
21  Add the default class associated with p to end of C;
22  Return C without totalErrors and default-class;
\end{verbatim}

If rule r no longer correctly classifies any class, it is not saved to the final rule list.

Source: [1], CBA-CB algorithm M2
CBA-CB M2  Stage 3

1. `classDistr = compClassDistri(D);`
2. `ruleErrors = 0;`
3. `Q = sort(Q);`
4. `for each rule r in Q in sequence do`
5. `if r.classCasesCovered[r.class] ≠ 0 then`
6. `for each entry <rul, did, y> in r.replace do`
7. `if the did case has been covered by a previous r then`
8. `r.classCasesCovered[y]--;`
9. `else rul.classCasesCovered[y]--;`
10. `ruleErrors = ruleErrors + errorsOfRule(r);`
11. `classDistr = update(r, classDistr);`
12. `defaultClass = selectDefault(classDistr);`
13. `defaultErrors = defErr(defaultClass, classDistr);`
14. `totalErrors = ruleErrors + defaultErrors;`
15. `Insert <r, default-class, totalErrors> at end of C`
16. `end`
17. `end`
18. Find the first rule p in C with the lowest totalErrors, and then discard all the rules after p from C;
19. Add the default class associated with p to end of C;
20. Return C without totalErrors and default-class;

Source: [1], CBA-CB algorithm M2

r.replace holds the list of cRules (rul), which this rule replaces (as wRule)

r tries to replace each rule rul in r.replace
This won’t succeed if there is a higher precedence rule r, which covers d.
1. \( \text{classDistr} = \text{compClassDistri}(D) \);
2. \( \text{ruleErrors} = 0 \);
3. \( Q = \text{sort}(Q) \);
4. for each rule \( r \) in \( Q \) in sequence do
   5. if \( r.\text{classCasesCovered}[r.\text{class}] \neq 0 \) then
      6. for each entry \( <\text{rul}, dID, y> \) in \( r.\text{replace} \) do
         7. if the \( dID \) case has been covered by a previous \( r \) then
             8. \( r.\text{classCasesCovered}[y]-- \);
         9. else \( rul.\text{classCasesCovered}[y]-- \);
    10. \( \text{ruleErrors} = \text{ruleErrors} + \text{errorsOfRule}(r) \);
    11. \( \text{classDistr} = \text{update}(r, \text{classDistr}) \);
    12. \( \text{defaultClass} = \text{selectDefault}(\text{classDistr}) \);
    13. \( \text{defaultErrors} = \text{defErr}(\text{defaultClass}, \text{classDistr}) \);
    14. \( \text{totalErrors} = \text{ruleErrors} + \text{defaultErrors} \);
    15. Insert \( <r, \text{default-class}, \text{totalErrors}> \) at end of \( C \)
5. end
16. end
17. Find the first rule \( p \) in \( C \) with the lowest \( \text{totalErrors} \), and then discard all the rules after \( p \) from \( C \);
18. Add the default class associated with \( p \) to end of \( C \);
19. Return \( C \) without \( \text{totalErrors} \) and \( \text{default-class} \);

Errors caused by the current rule and previously processed higher precedence rules.
classDistr = compClassDistri(D);
ruleErrors = 0;
Q = sort(Q);
for each rule \( r \) in \( Q \) in sequence do
    if \( r.classCasesCovered[r.class] \neq 0 \) then
        for each entry \( <rul, dID, y> \) in \( r.replace \) do
            if the \( dID \) case has been covered by a previous \( r \) then
                \( r.classCasesCovered[y]--; \)
            else \( rul.classCasesCovered[y]--; \)
            ruleErrors = ruleErrors + errorsOfRule(\( r \));
    classDistr = update(\( r \), classDistr);
    defaultClass = selectDefault(classDistr);
    defaultErrors = defErr(defaultClass, classDistr);
    totalErrors = ruleErrors + defaultErrors;
    Insert \( <r, default-class, totalErrors> \) at end of \( C \)
end

Find the first rule \( p \) in \( C \) with the lowest \( totalErrors \), and then discard all the rules after \( p \) from \( C \);
Add the default class associated with \( p \) to end of \( C \);
Return \( C \) without \( totalErrors \) and \( default-class \).
1. \(classDistr = \text{compClassDistri}(D)\);
2. \(ruleErrors = 0;\)
3. \(Q = \text{sort}(Q);\)
4. \(\text{for each rule } r \text{ in } Q \text{ in sequence do}\)
5. \(\text{if } r.\text{classCasesCovered}[r.\text{class}] \neq 0 \text{ then}\)
6. \(\text{for each entry } <rul, dID, y> \text{ in } r.\text{replace do}\)
7. \(\text{if the } dID \text{ case has been covered by a previous } r \text{ then}\)
8. \(r.\text{classCasesCovered}[y]--;\)
9. \(\text{else } rul.\text{classCasesCovered}[y]--;\)
10. \(\text{ruleErrors} = \text{ruleErrors} + \text{errorsOfRule}(r);\)
11. \(classDistr = \text{update}(r, \text{classDistr});\)
12. \(\text{defaultClass} = \text{selectDefault}(\text{classDistr});\)
13. \(\text{defaultErrors} = \text{defErr}(\text{defaultClass}, \text{classDistr});\)
14. \(\text{totalErrors} = \text{ruleErrors} + \text{defaultErrors};\)
15. \(\text{Insert } <r, \text{default-class, totalErrors}> \text{ at end of } C\)
16. \(\text{end}\)
17. \(\text{end}\)
18. \(\text{Find the first rule } p \text{ in } C \text{ with the lowest } totalErrors,\)
19. \(\text{and then discard all the rules after } p \text{ from } C;\)
20. \(\text{Add the default class associated with } p \text{ to end of } C;\)
21. \(\text{Return } C \text{ without } totalErrors \text{ and } \text{default-class};\)

Counts the number of training cases in each class in the initial training data.

Majority class in the remaining training data.
classDistr = compClassDistri(D);
ruleErrors = 0;
Q = sort(Q);
for each rule r in Q in sequence do
  if r.classCasesCovered[r.class] ≠ 0 then
    for each entry <rul, dID, y> in r.replace do
      if the dID case has been covered by a previous r then
        r.classCasesCovered[y]--;  
      else  rul.classCasesCovered[y]--; 
    ruleErrors = ruleErrors + errorsOfRule(r);
  classDistr = update(r, classDistr);
  defaultClass = selectDefault(classDistr);
  defaultErrors = defErr(defaultClass, classDistr);
  totalErrors = ruleErrors + defaultErrors;
  Insert <r, default-class, totalErrors> at end of C
end
end
Find the first rule p in C with the lowest totalErrors, 
and then discard all the rules after p from C;
Add the default class associated with p to end of C;
Return C without totalErrors and default-class;

Counts the number of training cases in each class in the initial training data.
The number of errors the default class will make in the remaining data.

Source: [1], CBA-CB algorithm M2
1 \textit{classDistr} = \text{compClassDistri}(D); \\
2 \textit{ruleErrors} = 0; \\
3 Q = \text{sort}(Q); \\
4 \textbf{for each rule } r \text{ in } Q \text{ in sequence do} \\
5 \quad \textbf{if } r.\text{classCasesCovered}[r.\text{class}] \neq 0 \textbf{ then} \\
6 \quad \quad \textbf{for each entry } <\textit{rul}, dID, y> \text{ in } r.\text{replace do} \\
7 \quad \quad \quad \textbf{if the } dID \text{ case has been covered by a} \\
8 \quad \quad \quad \quad \textbf{previous } r \textbf{ then} \\
9 \quad \quad \quad \quad r.\text{classCasesCovered}[y]--; \\
10 \quad \quad \quad \textbf{else } rul.\text{classCasesCovered}[y]--; \\
11 \quad \text{ruleErrors} = \text{ruleErrors} + \text{errorsOfRule}(r); \\
12 \quad \textit{classDistr} = \text{update}(r, \textit{classDistr}); \\
13 \quad \text{defaultClass} = \text{selectDefault}(\textit{classDistr}); \\
14 \quad \text{defaultErrors} = \text{defErr}(\text{defaultClass}, \textit{classDistr}); \\
15 \quad \text{totalErrors} = \text{ruleErrors} + \text{defaultErrors}; \\
16 \quad \textbf{end } r \textbf{ is added to the final rule list} \\
17 \textbf{end } \textbf{for each rule } r \text{ in } Q \text{ in sequence do} \\
18 \text{Find the first rule } p \text{ in } C \text{ with the lowest } totalErrors, \\
19 \quad \text{and then discard all the rules after } p \text{ from } C; \\
20 \text{Add the default class associated with } p \text{ to end of } C; \\
21 \text{Return } C \text{ without } totalErrors \text{ and } default-class;
Final rule pruning (3\textsuperscript{rd})

```c
classDistr = compClassDistri(D);
ruleErrors = 0;
Q = sort(Q);
for each rule r in Q in sequence do
    if r.classCasesCovered[r.class] ≠ 0 then
        for each entry <rul, dID, y> in r.replace do
            if the dID case has been covered by a previous r then
                r.classCasesCovered[y]--;
            else rul.classCasesCovered[y]--;
            ruleErrors = ruleErrors + errorsOfRule(r);
        classDistr = update(r, classDistr);
        defaultClass = selectDefault(classDistr);
        defaultErrors = defErr(defaultClass, classDistr);
        totalErrors = ruleErrors + defaultErrors;
        Insert <r, default-class, totalErrors> at end of C
    end
end
Find the first rule p in C with the lowest totalErrors, and then discard all the rules after p from C;
Add the default class associated with p to end of C;
Return C without totalErrors and default-class;
```

“default rule pruning”

Source: [1], CBA-CB algorithm M2
## CBA-CB M2 Stage 3

<table>
<thead>
<tr>
<th>Datasets</th>
<th>c4.5rules w/o discr.</th>
<th>c4.5rules discr.</th>
<th>CBA (CARs + infreq w/o pru.)</th>
<th>CBA (CARs w/o pru.)</th>
<th>No. of CARs w/o pru.</th>
<th>Run time (sec) (CBA-RG) w/o pru.</th>
<th>Run time (sec) (CBA-RG) pru.</th>
<th>Run time (sec) (CBA-CB) M1</th>
<th>Run time (sec) (CBA-CB) M2</th>
<th>No. of Rules in C</th>
</tr>
</thead>
<tbody>
<tr>
<td>anneal*</td>
<td>5.2</td>
<td>6.5</td>
<td>1.9</td>
<td>1.9</td>
<td>3.2</td>
<td>3.6</td>
<td>65081</td>
<td>611</td>
<td>14.33 14.36</td>
<td>0.08 0.06</td>
</tr>
<tr>
<td>australian*</td>
<td>15.3</td>
<td>13.5</td>
<td>13.5</td>
<td>13.4</td>
<td>13.2</td>
<td>13.4</td>
<td>46564</td>
<td>4064</td>
<td>5.00 5.05</td>
<td>0.20 0.22</td>
</tr>
<tr>
<td>auto*</td>
<td>19.9</td>
<td>29.2</td>
<td>21.0</td>
<td>23.1</td>
<td>24.0</td>
<td>27.2</td>
<td>50236</td>
<td>3969</td>
<td>3.30 3.55</td>
<td>0.12 0.06</td>
</tr>
<tr>
<td>breast-w</td>
<td>5.0</td>
<td>3.9</td>
<td>3.9</td>
<td>3.9</td>
<td>4.2</td>
<td>4.2</td>
<td>2831</td>
<td>399</td>
<td>0.30 0.33</td>
<td>0.02 0.03</td>
</tr>
<tr>
<td>cleve*</td>
<td>21.8</td>
<td>18.2</td>
<td>18.1</td>
<td>19.1</td>
<td>16.7</td>
<td>16.7</td>
<td>48854</td>
<td>1634</td>
<td>4.00 4.30</td>
<td>0.04 0.06</td>
</tr>
<tr>
<td>crx*</td>
<td>15.1</td>
<td>15.9</td>
<td>14.3</td>
<td>14.3</td>
<td>14.1</td>
<td>14.1</td>
<td>42877</td>
<td>4717</td>
<td>4.90 5.06</td>
<td>0.43 0.30</td>
</tr>
<tr>
<td>diabetes</td>
<td>25.8</td>
<td>27.6</td>
<td>24.8</td>
<td>25.5</td>
<td>24.7</td>
<td>25.3</td>
<td>3315</td>
<td>162</td>
<td>0.25 0.28</td>
<td>0.03 0.01</td>
</tr>
<tr>
<td>german*</td>
<td>27.7</td>
<td>29.5</td>
<td>27.2</td>
<td>26.5</td>
<td>25.2</td>
<td>26.5</td>
<td>69277</td>
<td>4561</td>
<td>5.60 6.00</td>
<td>1.04 0.28</td>
</tr>
<tr>
<td>glass</td>
<td>31.3</td>
<td>27.5</td>
<td>27.4</td>
<td>27.4</td>
<td>27.4</td>
<td>27.4</td>
<td>4234</td>
<td>291</td>
<td>0.20 0.22</td>
<td>0.02 0.00</td>
</tr>
<tr>
<td>heart</td>
<td>19.2</td>
<td>18.9</td>
<td>19.6</td>
<td>19.6</td>
<td>18.5</td>
<td>18.5</td>
<td>52309</td>
<td>624</td>
<td>4.70 4.60</td>
<td>0.04 0.03</td>
</tr>
<tr>
<td>hepatitis*</td>
<td>19.4</td>
<td>22.6</td>
<td>15.1</td>
<td>15.1</td>
<td>15.1</td>
<td>15.1</td>
<td>63134</td>
<td>2275</td>
<td>2.80 2.79</td>
<td>0.09 0.05</td>
</tr>
<tr>
<td>horse*</td>
<td>17.4</td>
<td>16.3</td>
<td>18.2</td>
<td>17.9</td>
<td>18.7</td>
<td>18.7</td>
<td>62745</td>
<td>7846</td>
<td>3.2 3.33</td>
<td>0.35 0.19</td>
</tr>
<tr>
<td>hypo*</td>
<td>0.8</td>
<td>1.2</td>
<td>1.6</td>
<td>1.6</td>
<td>1.9</td>
<td>1.7</td>
<td>37631</td>
<td>493</td>
<td>45.60 45.30</td>
<td>1.02 0.40</td>
</tr>
<tr>
<td>ionosphere*</td>
<td>10.0</td>
<td>8.0</td>
<td>7.9</td>
<td>7.9</td>
<td>8.2</td>
<td>8.2</td>
<td>55701</td>
<td>10055</td>
<td>3.75 4.00</td>
<td>0.56 0.41</td>
</tr>
<tr>
<td>iris</td>
<td>4.7</td>
<td>5.3</td>
<td>7.1</td>
<td>7.1</td>
<td>7.1</td>
<td>7.1</td>
<td>72</td>
<td>23</td>
<td>0.00 0.00</td>
<td>0.00 0.00</td>
</tr>
<tr>
<td>labor</td>
<td>20.7</td>
<td>21.0</td>
<td>17.0</td>
<td>17.0</td>
<td>17.0</td>
<td>17.0</td>
<td>5565</td>
<td>313</td>
<td>0.17 0.20</td>
<td>0.00 0.00</td>
</tr>
<tr>
<td>led7</td>
<td>26.5</td>
<td>26.5</td>
<td>27.8</td>
<td>27.8</td>
<td>27.8</td>
<td>27.8</td>
<td>464</td>
<td>336</td>
<td>0.40 0.45</td>
<td>0.11 0.10</td>
</tr>
<tr>
<td>lymph*</td>
<td>26.5</td>
<td>21.0</td>
<td>20.3</td>
<td>18.9</td>
<td>20.3</td>
<td>19.6</td>
<td>40401</td>
<td>2965</td>
<td>2.70 2.70</td>
<td>0.07 0.05</td>
</tr>
<tr>
<td>pima</td>
<td>24.5</td>
<td>27.5</td>
<td>26.9</td>
<td>27.0</td>
<td>27.4</td>
<td>27.6</td>
<td>2977</td>
<td>125</td>
<td>0.23 0.25</td>
<td>0.04 0.02</td>
</tr>
<tr>
<td>sick*</td>
<td>1.5</td>
<td>2.1</td>
<td>2.8</td>
<td>2.8</td>
<td>2.7</td>
<td>2.7</td>
<td>71828</td>
<td>627</td>
<td>32.60 33.40</td>
<td>0.62 0.40</td>
</tr>
<tr>
<td>sonar*</td>
<td>29.8</td>
<td>27.8</td>
<td>24.3</td>
<td>21.7</td>
<td>24.3</td>
<td>21.7</td>
<td>57061</td>
<td>1693</td>
<td>5.34 5.22</td>
<td>0.30 0.12</td>
</tr>
<tr>
<td>tic-tac-toe</td>
<td>0.6</td>
<td>0.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>7063</td>
<td>1378</td>
<td>0.62 0.71</td>
<td>0.12 0.08</td>
</tr>
<tr>
<td>vehicle*</td>
<td>27.4</td>
<td>33.6</td>
<td>31.3</td>
<td>31.2</td>
<td>31.5</td>
<td>31.3</td>
<td>23446</td>
<td>5704</td>
<td>6.33 6.33</td>
<td>1.40 0.40</td>
</tr>
<tr>
<td>waveform*</td>
<td>21.9</td>
<td>24.6</td>
<td>20.2</td>
<td>20.2</td>
<td>20.4</td>
<td>20.6</td>
<td>9699</td>
<td>3396</td>
<td>13.65 13.55</td>
<td>2.72 1.12</td>
</tr>
<tr>
<td>wine</td>
<td>7.3</td>
<td>7.9</td>
<td>8.4</td>
<td>8.4</td>
<td>8.4</td>
<td>8.4</td>
<td>38070</td>
<td>1494</td>
<td>2.34 2.65</td>
<td>0.11 0.04</td>
</tr>
<tr>
<td>zoo*</td>
<td>7.8</td>
<td>7.8</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>52198</td>
<td>2049</td>
<td>2.73 2.70</td>
<td>0.61 0.32</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>16.7</strong></td>
<td><strong>17.1</strong></td>
<td><strong>15.6</strong></td>
<td><strong>15.6</strong></td>
<td><strong>15.7</strong></td>
<td><strong>15.8</strong></td>
<td><strong>35140</strong></td>
<td><strong>2377</strong></td>
<td><strong>6.35 6.44</strong></td>
<td><strong>0.39 0.18</strong></td>
</tr>
</tbody>
</table>

Source: [1]
Outline

Classification based on associations (CBA)


Business Rule CBA (brCBA)

- Simplified version of CBA
- The effect of higher rule expressiveness (disjunctions, negations) on classifier accuracy
- Effect of rule pruning

Monotonicity Exploiting Association Rule Classification (MARC)

- On going work
- Limitations of CBA (and association rule classifiers in general)
- Proposed solution
- Experimental results
Learning Business Rules
with Association Rule Classifiers

Presented at RuleML2014
(*abridged updated version*)

Tomáš Klígr¹,⁴, Jaroslav Kuchař¹,², Davide Sottara³, Stanislav Vojíř¹

¹ Dep. of Inf. And Knowl. Eng., University of Economics, Prague
² Web Engineering Group, Czech Technical University
³ Biomedical Informatics Department, Arizona State University
⁴ Multimedia and Vision Research Group, Queen Mary, University of London
Business rules

With Business Rule Management System (BRMS) applications can invoke decision logic which is input in the form of rules, instead of procedural code.

+ This reduces reliance on the IT experts
- Requires extensive subject matter expertise
- (A lot of) Expert time

RESTRICTION: We focus on “classification business rules”.

Knowledge base

Domain experts

Rule Engine

Iris versicolor

Iris ???

petalWidth>1.75 → iris-virginica
petalLength=[2.45;4.75] and → iris-versicolor
sepalWidth = [3.05;3.4]
...
Business rule learning

Ideally, the rule learning algorithm executed on the database of iris varieties would substitute the human expert.

As we will see, rule learning algorithms often yield rule sets that are
- Conflicting
- Contain redundant rules
- Excessive number of rules
- Syntactically simple
- Probabilistic

Knowledge base

Rule Learning

Rule Engine

Database

Iris versicolor

petalWidth>1.75 → iris-virginica
petalLength=[2.45;4.75] and → iris-versicolor
sepalWidth = [3.05;3.4]
...
Problem statement

- Conflicting
- Contain redundant rules
- Excessive number of rules
- Syntactically simple
- Probabilistic

R1: petalWidth > 1.75

R2: petalWidth > 1.75 and
    sepalWidth = [3.05;3.4]

\[ \Rightarrow \text{iris-virginica,} \]
\[ \text{supp= 0.296, conf=1} \]

\[ \Rightarrow \text{iris-virginica} \]
\[ \text{supp= 0.100, conf=1} \]

... 50 more rules

R9: sepalLength= (5.55;3.40] and
    sepalWidth < 3.05

\[ \Rightarrow \text{iris-versicolor} \]
\[ \text{supp=0.230, conf=0.05} \]

While this is not an issue for a completely automated “black box”
classifier, in a business setting the policy can be that the rule set
a) is expert-reviewed before deployment,
b) each decision made by the system can be explained,
c) the rules must be convertible to a form that can be processes by BRMS
BR Learning Requirements

Business rule learning needs a rule-learning approach, which has
- BRMS supported rule expressiveness
- Syntactically rich
- Small number of output rules
- Exhaustive set of rules
- Ability to control rule quality

BRMS can then take care of
- Refine the rule base (by Subject Matter Expert)
- Execute rules
  - Classify objects at run time
  - Evaluate complex criteria
  - Handle uncertainty
- Manage rule conflicts
  - Defeasible logic, higher order rules, ...
brCBA

- brCBA is a simplification of CBA, so that the algorithm can be quickly built on top of standard association rule learning implementation (e.g. Christian Borgelt’s arules package in R or LISp-Miner)

**Rule learning (brCBA)**

1. Learn association rules (constrained to contain the class attribute in consequent) with GUHA Method
2. Perform data coverage pruning

**Classification (same as in CBA algorithm)**

A standard BRMS rule engine can be used to apply the model (rule set) on data

- The data coverage pruning makes it simple to understand for business analyst why a specific rule output in the association rule learning was removed. No other pruning is performed.
- The absence of default rule pruning ensures that all rules matching the specified quality measures (minSupp and minConf) are on the output.
- GUHA method learns rich association rules with disjunctions and negations

Business rule learning needs a rule-learning approach, which has
- BRMS supported rule expressiveness
- Small number of output rules
- Exhaustive set of rules
- Expressive rule language
- Rule conflict resolution
- Ability to control rule quality
Rule Pruning

- Data coverage pruning is the most commonly used pruning technique in CBA-derived algorithms

**Algorithm 1 Data Coverage**

*Require*: rules – sorted list of rules, \( T \) – set of objects in the training dataset  
*Ensure*: rules – pruned list of rules

\[
\text{rules} := \text{sort rules according to criteria} \\
\text{for all } \text{rule} \in \text{rules} \text{ do} \\
\quad \text{matches} := \text{set of objects from } T \text{ that match both rule ant. and consequent.} \\
\quad \text{if } \text{matches} = \emptyset \text{ then} \\
\qquad \text{remove rule from rules} \\
\quad \text{else} \\
\qquad \text{remove matches from } T \\
\text{end if} \\
\text{end for} \\
\text{return rules}
\]

Rule ranking criteria
- Confidence
- Support
- Rule length (shorter is better)

This definition does not adhere exactly to CBA data coverage pruning, which removes all data cases matched by the rule antecedent (if it covers at least one positive instance). In brCBA we removed only the correctly classified instances.
Experiment objectives

• Evaluate impact of pruning
  – No pruning (use apriori output directly for classification)
  – brCBA (apriori, then data coverage pruning)
  – Original CBA (data coverage, pessimistic and default rule pruning)

• Evaluate the impact and sensitivity to:
  – minSupport threshold
  – minConfidence threshold

• Evaluate the impact of added rule language expressivity
  – negations
  – disjunctions in rule body
Experimental setup

Datasets
- UCI: Iris, Glass

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Rows</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>150</td>
<td>4</td>
</tr>
<tr>
<td>Glass</td>
<td>214</td>
<td>9</td>
</tr>
</tbody>
</table>

Experiment objectives
1) Compare results with other classifiers
2) Determine impact of:
   - minSupport thr.
   - minConfidence thr.
   - pruning

Preprocessing
- Numerical attributes were discretized with equidistant binning with custom merging of bins with small support

Rule learning
- LISp-Miner implementation, apriori-like setup

Pruning
- Data coverage pruning on/off
**Experimental results**

**Pruning**: decreased the rule count by 90%, lowering accuracy only by 1%

Pruning makes the rule count and accuracy insensitive to minConf threshold (within considerable range)

Effect of pruning. Iris dataset, minimum support threshold 1

*iris dataset*
## Experimental results

<table>
<thead>
<tr>
<th>Dataset, task</th>
<th>support</th>
<th>not pruned</th>
<th>pruned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Rules</td>
<td>Accuracy</td>
</tr>
<tr>
<td>iris</td>
<td>10</td>
<td>87</td>
<td>0.940</td>
</tr>
<tr>
<td>&quot;</td>
<td>2</td>
<td>168</td>
<td>0.947</td>
</tr>
<tr>
<td>&quot;</td>
<td>1</td>
<td>291</td>
<td><strong>0.967</strong></td>
</tr>
<tr>
<td>iris, sequence 1-2</td>
<td>10</td>
<td>904</td>
<td>0.940</td>
</tr>
<tr>
<td>&quot;</td>
<td>2</td>
<td>1661</td>
<td>0.953</td>
</tr>
<tr>
<td>&quot;</td>
<td>1</td>
<td>2653</td>
<td><strong>0.960</strong></td>
</tr>
<tr>
<td>glass</td>
<td>10</td>
<td>32</td>
<td>0.464</td>
</tr>
<tr>
<td>&quot;</td>
<td>2</td>
<td>2374</td>
<td><strong>0.622</strong></td>
</tr>
<tr>
<td>balance scale</td>
<td>10</td>
<td>124</td>
<td><strong>0.891</strong></td>
</tr>
<tr>
<td>&quot;</td>
<td>2</td>
<td>558</td>
<td>0.841</td>
</tr>
<tr>
<td>balance scale, subset 1-2</td>
<td>10</td>
<td>11947</td>
<td>0.758</td>
</tr>
</tbody>
</table>

**Impact of minimum support threshold, minConf=0.6**

**Support:** The lower, the better (and slower).
Experimental results

<table>
<thead>
<tr>
<th>confidence</th>
<th>not pruned</th>
<th>pruned</th>
<th>not pruned</th>
<th>pruned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rules Accuracy</td>
<td></td>
<td>Rules Accuracy</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>58.3</td>
<td>0.529</td>
<td>25.8</td>
<td>0.534</td>
</tr>
<tr>
<td>0.6</td>
<td>31.8</td>
<td>0.464</td>
<td>21.1</td>
<td>0.464</td>
</tr>
<tr>
<td>0.7</td>
<td>10.3</td>
<td>0.290</td>
<td>8.4</td>
<td>0.286</td>
</tr>
<tr>
<td>0.8</td>
<td>2.4</td>
<td>0.117</td>
<td>1.8</td>
<td>0.117</td>
</tr>
<tr>
<td>0.9</td>
<td>0.4</td>
<td>0.010</td>
<td>0.2</td>
<td>0.010</td>
</tr>
</tbody>
</table>

Glass, minSupp=10 objects (5.18%)

<table>
<thead>
<tr>
<th></th>
<th>not pruned</th>
<th>pruned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rules Accuracy</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>96</td>
<td>0.940</td>
</tr>
<tr>
<td>0.6</td>
<td>87</td>
<td>0.940</td>
</tr>
<tr>
<td>0.7</td>
<td>83</td>
<td>0.940</td>
</tr>
<tr>
<td>0.8</td>
<td>76</td>
<td>0.940</td>
</tr>
<tr>
<td>0.9</td>
<td>68</td>
<td>0.900</td>
</tr>
</tbody>
</table>

Iris, minSupp=10 objects (1.78%)

<table>
<thead>
<tr>
<th></th>
<th>not pruned</th>
<th>pruned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rules Accuracy</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>124</td>
<td>0.891</td>
</tr>
<tr>
<td>0.7</td>
<td>86</td>
<td>0.875</td>
</tr>
<tr>
<td>0.8</td>
<td>50</td>
<td>0.790</td>
</tr>
<tr>
<td>0.9</td>
<td>24</td>
<td>0.547</td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
<td>0.047</td>
</tr>
</tbody>
</table>

Balancescale, minSupp 10 objects (1.78%)

Confidence: The lower, the better.
Additional experiments

Datasets
- UCI: Iris, Balance scale, Glass

Preprocessing
- Numerical attributes were discretized with equidistant binning with custom merging of bins with small support

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Rows</th>
<th>Attributes</th>
<th>Bins after preprocessing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>150</td>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td>BalanceScale</td>
<td>625</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>Glass</td>
<td>214</td>
<td>9</td>
<td>19</td>
</tr>
</tbody>
</table>

Rule learning
- Default run (as in apriori)
- **Negations**
  - for each item, a dual “negated” item is created
- **Dynamic binning – nominal attributes (“subset” length = 2)**
- **Dynamic binning – cardinal attributes (“interval” length = 2)**

Pruning
- Data coverage pruning on/off
Higher expressivity rules with GUHA

- The standard apriori algorithm outputs **conjunctive** rules
- BRMS systems routinely work with rules that contain **disjunctions** between attribute values (dynamic binning) or **negated literals**.
- In our experiments, we have employed in the LISp-Miner system which unlike apriori implementations is able to learn higher expressiveness rules.

<table>
<thead>
<tr>
<th>Original intervals</th>
<th>Sequence 1-2</th>
<th>Subset 1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>[30-35)</td>
<td>[30-35),(35-40)</td>
<td>[20-25),(35-40)</td>
</tr>
<tr>
<td>[35-40)</td>
<td>[35-40),(40-45)</td>
<td>[20-25),(40-45)</td>
</tr>
<tr>
<td>[40-45)</td>
<td>[45-50)</td>
<td>[25-30),(30-35)</td>
</tr>
</tbody>
</table>

**Sequence:** binning of following categories (for ordinal attributes)
**Subset:** binning of categories regardless of the order
**Length 1-2:** generated bins contain at least 1 and maximum 2 original bins
Experimental results

**Dynamic binning**

Effect of dynamic binning on cardinal attributes.
*Iris dataset*

\[
\text{sepalWidth} = [3.2;3.44) \ \text{or} \ \text{sepalWidth} = [3.44;3.68) \Rightarrow XClass(Iris-setosa)\
\]

---

Dynamic binning (cardinal attributes) –
better accuracy (3.4% improvement) and lower rule count (18 vs 23).
However – **much** longer learning time (LISP-Miner).

---

Dynamic binning off

Dynamic binning on
Experimental results: dynamic binning

Effect of dynamic binning on nominal attributes. 
*Balancescale dataset*

\[(\text{LeftDistance}=\text{S} \text{ or LeftDistance}=\text{M}) \text{ and } (\text{LeftWeight}=\text{L} \text{ or LeftWeight}=\text{H}) \Rightarrow XClass=\text{L}\]

Dynamic binning (nominal attributes)—worse accuracy, higher rule count and *drastically* longer learning time.
Experimental results negative literals

Effect of including negative literals. *Iris dataset*

\begin{itemize}
\item petalLength = [1;1.59]
\item and petalWidth = [0.1;0.34]
\item and not(sepalLength = [4.3;4.66])
\item and not(sepalWidth = [2;2.34])
\item => XClass(Iris-setosa)
\end{itemize}

Negative literals – worse accuracy, higher rule count and higher learning time.
## Experimental results

<table>
<thead>
<tr>
<th>Dataset/Task</th>
<th>Attributes</th>
<th>Verifications</th>
<th>Rules</th>
<th>Mining duration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>without binning</td>
<td>315</td>
<td>80</td>
<td>less than 1 s</td>
</tr>
<tr>
<td></td>
<td>with negations</td>
<td>13 542</td>
<td>2 472</td>
<td>12 s</td>
</tr>
<tr>
<td></td>
<td>disjunctions (nominal)</td>
<td>19 413</td>
<td>4 715</td>
<td>27 s</td>
</tr>
<tr>
<td>BalanceScale (min Conf 0.5)</td>
<td>without binning</td>
<td>510</td>
<td>146</td>
<td>less than 1 s</td>
</tr>
<tr>
<td></td>
<td>with negations</td>
<td>33 045</td>
<td>9 040</td>
<td>43 s</td>
</tr>
<tr>
<td></td>
<td>disjunctions (nominal)</td>
<td>73 230</td>
<td>17 004</td>
<td>99 s</td>
</tr>
<tr>
<td></td>
<td>disjunctions (cardinal)</td>
<td>9 582</td>
<td>2 122</td>
<td>10 s</td>
</tr>
<tr>
<td></td>
<td>disjunctions (cardinal – 3 values)</td>
<td>45 915</td>
<td>11 846</td>
<td>75 s</td>
</tr>
<tr>
<td>Glass (min Conf 0.9)</td>
<td>without binning</td>
<td>3 920</td>
<td>24</td>
<td>less than 1 s</td>
</tr>
<tr>
<td></td>
<td>with negations</td>
<td>669 075</td>
<td>8 146</td>
<td>64 s</td>
</tr>
<tr>
<td></td>
<td>dynamic binning</td>
<td>not suitable (attributes have only 2 values)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Experimental results overview

<table>
<thead>
<tr>
<th>dataset</th>
<th>previously reported results</th>
<th>brCBA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C4.5 ripper</td>
<td>cba</td>
</tr>
<tr>
<td>iris</td>
<td>0.953</td>
<td>0.940</td>
</tr>
<tr>
<td>glass</td>
<td>0.687</td>
<td>0.691</td>
</tr>
</tbody>
</table>
Monotonicity Exploiting Association Rule Classification
(tentative title)

Working draft
Tomáš Kliegr

Multimedia and Vision Research Group
Queen Mary
University of London

Supervisors:
Prof. Ebroul Izquierdo
Multimedia & Vision Group
Queen Mary University of London

Dr. Christopher Tyson
Department of Economics, Queen Mary
Queen Mary University of London
Association rules identify only the high density regions in the data, which have a strong presence of one target class.

The definition of “high density” is controlled by the *minimum support* parameter, and the definition of strong presence by the *minimum confidence* parameter.

Rules output with minConf = 0.6 and minSupp = 1

Ceteris paribus: Humidity = (40;60)
Challenges

- Ignores regions in the data with small density (otherwise combinatorial explosion).
- Limited to hypercube regions: The problem is further aggravated by the fact that learning is performed on transformed feature space (cardinal features are discretized to bins).
- Does not incorporate the monotonicity assumption and the probability-distribution nature of rule prediction.

Rules output with minConf = 0.75 and minSupp = 3
The MARC algorithm was proposed to address these challenges.

Three fundamental steps:

- Learn association rules
- Postprocess the rules to incorporate the monotonicity assumption
- Annotate the rules with probability density functions

MARC consists of several consecutive procedures:

- Association rule learning and pruning (standard algorithms)
- Rule Extension – the core procedure implementing the mon. assump.
- Rule Fuzzification - further extending rule coverage
- Rule Annotation with probability density functions
- Rule mixture classification
Step 1: learn association rules
Step 2: extend association rules

1) The rule boundaries are adjusted to supporting points in the original feature space.
Step 2: extend association rules

2) The rules are monotonically extended outside the grid in each literal.

The extension is performed to the last point whose inclusion improves rule confidence.
Step 1: learn association rules

2) The rules are monotonically extended outside the grid in each literal.

The extension is performed to the last point whose inclusion improves rule confidence.

Conditional accept
Further extension is not possible
Extension retracts
Step 2: extend association rules

Extension result

Humidity
Step 2: extend association rules

The original rule set contained two rules
Humidity=(40;60] & Temperature=[25;30) => Utility=4
Humidity=(80;100] => Utility=2

Through rule extension, these rules were enlarged and refined to
Humidity=(38;58] & Temperature=[25;34) => Utility=4
Humidity=(85;95] => Utility=2

To further extend the coverage of the instance space,
Rules are extended by appending fuzzy borders
Step 3: fuzzify rules

The coverage of each literal created over a cardinal attribute in the body of a rule is extended by appending a value adjacent to the lowest and highest values.
Step 4: annotate rules with probability distributions

Probability density functions are stored for each distinct feature value across all points in the training data that are covered by the rule.
Step 2: aggregate distributions for matching rule

1) Locate the nearest supporting annotated line segments
2) Aggregate the probability density functions
Step 2: apply model - aggregate rules

Instance can be covered by multiple rules

1) The distribution is first aggregated for the individual rules.

2) The aggregation is performed across the rules

Naturally provides ranking of related content
- Draft of the algorithm
- Several standard datasets from the UCI repository and a large dataset from the CLEF recommender system challenge.

<table>
<thead>
<tr>
<th>dataset</th>
<th>autos*</th>
<th>breast*</th>
<th>glass</th>
<th>iris</th>
<th>pima*</th>
<th>clef</th>
</tr>
</thead>
<tbody>
<tr>
<td>decision tree</td>
<td>0.805</td>
<td>0.940</td>
<td>0.663</td>
<td>0.940</td>
<td>0.682</td>
<td>0.02</td>
</tr>
<tr>
<td>random tree</td>
<td>0.408</td>
<td>0.936</td>
<td>0.411</td>
<td>0.907</td>
<td>0.655</td>
<td>0.02</td>
</tr>
<tr>
<td>decision stump</td>
<td>0.352</td>
<td>0.924</td>
<td>0.435</td>
<td>0.667</td>
<td>0.720</td>
<td>0.02</td>
</tr>
<tr>
<td>ripper</td>
<td>0.793</td>
<td>0.916</td>
<td>0.641</td>
<td>0.927</td>
<td>0.721</td>
<td>NA-T</td>
</tr>
<tr>
<td>logistic regression</td>
<td>0.711</td>
<td>0.962</td>
<td>0.555</td>
<td>0.933</td>
<td>0.768</td>
<td>NA-M</td>
</tr>
<tr>
<td>svm-rbf kernel</td>
<td>0.340</td>
<td>0.971</td>
<td>0.559</td>
<td>0.727</td>
<td>0.725</td>
<td>0.15</td>
</tr>
<tr>
<td>svm-linear kernel</td>
<td>0.440</td>
<td>0.968</td>
<td>0.471</td>
<td>0.667</td>
<td>0.753</td>
<td>NA-M</td>
</tr>
<tr>
<td>neural network</td>
<td>0.774</td>
<td>0.971</td>
<td>0.692</td>
<td>0.967</td>
<td>0.744</td>
<td>NA-T</td>
</tr>
<tr>
<td>MARC</td>
<td>0.843</td>
<td>0.936</td>
<td>0.682</td>
<td>0.940</td>
<td>0.717</td>
<td>0.11</td>
</tr>
</tbody>
</table>


**Relevant publications**

Jaroslav Kuchar, Tomáš Kliegr: InBeat: Recommender System as a Service. CLEF (Working Notes) 2014: 837-844

Tomáš Kliegr, Jaroslav Kuchař: Orwellian Eye: Video Recommendation with Microsoft Kinect. ECAI 2014: 1227-1228

Jaroslav Kuchař, Tomáš Kliegr: GAIN: web service for user tracking and preference learning - a smart TV use case. RecSys 2013: 467-468