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

Outline

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

Classification Association Rule Mining

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.
MMAC - Thabtah, Fadi A., Peter Cowling, and Yonghong Peng. MMAC: A new multi-class, multi-label associative classification approach. Data Mining, 2004. ICDM'04. IEEE, 2004.
CPAR,…

Classification based on associations (CBA)

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 $X \rightarrow y, X \subseteq I, y \in$ Y be a classification association rule (CAR)
  - Right hand side of the association rule is restricted to the target attribute
  - Rule $X \rightarrow y$ 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 $\in$ Y is a class label

where (A,1) is an attribute value pair

<{(A, 1), (B, 1)}, (class, 1)>
Example 2-ruleitem

Rule item =~ rule

k-ruleitem

rule item whose condset has *k* items

frequent (large) rule item

a ruleitem with support above minSup

CBA – Rule Generation (CBA-RG)

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
prCAR_1 = pruneRules(CAR_1);
for (k = 2; F_{k-1} ≠ ∅; k++) do
    C_k = candidateGen(F_{k-1});
    for each data case d ∈ D do
        C_d = ruleSubset(C_k, d);
        for each candidate c ∈ C_d do
            c.condsupCount++;
            if d.class = c.class then c.rulesupCount++
        end
    end
    F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
    CAR_k = genRules(F_k);
    prCAR_k = pruneRules(CAR_k);
end
CARs = ∪_k CAR_k;
prCARs = ∪_k prCAR_k;
```

In the first pass, the algorithm computes the support of individual rule items and discards rule items which are infrequent.

Source: [1]

CBA – Rule Generation (CBA-RG)

$F_1 = \{\text{large 1-ruleitems}\}$;
$CAR_1 = \text{genRules}(F_1)$;
$prCAR_1 = \text{pruneRules}(CAR_1)$;
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.condsupCount++;
            if d.class = c.class **then** c.rulesupCount++
        end
    end
    $F_k = \{c \in C_k \mid c.\text{rulesupCount} \geq minsup\}$;
    $CAR_k = \text{genRules}(F_k)$;
    $prCAR_k = \text{pruneRules}(CAR_k)$;
end
$CARs = \bigcup_k CAR_k$;
$prCARs = \bigcup_k prCAR_k$;

determine frequent/large 1- rule items (count class and item occurrences)

Example 1-ruleitem
<{(A, 1) (B,1)}, (class, 1)>
- support = 20% = 2/10
- confidence = 66.7% = 2/3

F_k
Denotes the set of **frequent** k-rule items
The elements of this set have the following form:
`<(condset,condsupCount), (y, rulesupCount)>`
`<({(A, 1) (B,1)}, 3), ((class, 1), 2)>`

Source: [1]

Generating the complete set of CARs

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

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

We get one possible rule: R1 with confidence 67%

Source: [1]

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

CBA – Rule Generation (CBA-RG)

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
prCAR_1 = pruneRules(CAR_1);
for (k = 2; F_{k-1} ≠ ∅; k++) do
    C_k = candidateGen(F_{k-1});
    for each data case d ∈ D do
        C_d = ruleSubset(C_k, d);
        for each candidate c ∈ C_d do
            c.condsupCount++;
            if d.class = c.class then c.rulesupCount++
        end
    end
    F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
    CAR_k = genRules(F_k);
    prCAR_k = pruneRules(CAR_k);
end
CARs = ∪_k CAR_k;
prCARs = ∪_k prCAR_k;
```

optional pessimistic rule pruning as in C4.5 [5]

rule pruning

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

CBA – Rule Generation (CBA-RG)

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
prCAR_1 = pruneRules(CAR_1);
for (k = 2; F_{k-1} ≠ ∅; k++) do
    C_k = candidateGen(F_{k-1});
    for each data case d ∈ D do
        C_d = ruleSubset(C_k, d);
        for each candidate c ∈ C_d do
            c.condsupCount++;
            if d.class = c.class then c.rulesupCount++
        end
    end
    F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
    CAR_k = genRules(F_k);
    prCAR_k = pruneRules(CAR_k);
end
CARs = ∪_k CAR_k;
prCARs = ∪_k prCAR_k;
```

subsequent passes of the CBA-RG

Source: [1]

CBA – Rule Generation (CBA-RG)

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
prCAR_1 = pruneRules(CAR_1);
for (k = 2; F_{k-1} ≠ ∅; k++) do
    C_k = candidateGen(F_{k-1});
    for each data case d ∈ D do
        C_d = ruleSubset(C_k, d);
        for each candidate c ∈ C_d do
            c.condsupCount++;
            if d.class = c.class then c.rulesupCount++
        end
    end
    F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
    CAR_k = genRules(F_k);
    prCAR_k = pruneRules(CAR_k);
end
CARs = ∪_k CAR_k;
prCARs = ∪_k prCAR_k;
```

same principle as aprioriGen [4]

Source: [1]

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

aprioriGen

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 $p.item_1$, $p.item_2$,…, $p.item_{k-1}$, $q.item_{k-1}$

from F_{k-1} p, F_{k-1} q

where $p.item_1 = q.item_1$,…, $p.item_{k-2} = q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$

F_{k-1}	C_k
{1,2,3}	{1,2,3,4}
{1,2,4}	{1,2,3,5}
{1,2,5}	{1,2,4,5}
{1,3,5}	{2,3,4,5}
{2,3,4}	
{2,3,5}	
{3,4,5}	

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 $p.item_1 = q.item_1$,…, $p.item_{k-2} = q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$

F_{k-1}	C_k
{1,2,3}	{1,2,3,4}
{1,2,4}	{1,2,3,5}
{1,2,5}	{1,2,4,5}
{1,3,5}	{2,3,4,5}
{2,3,4}	
{2,3,5}	
{3,4,5}	

aprioriGen – join step

insert into C_k

select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$

from F_{k-1} p, F_{k-1} q

where $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$

F_{k-1}	C_k
{1,2,3}	{1,2,3,4}
{1,2,4}	{1,2,3,5}
{1,2,5}	{1,2,4,5}
{1,3,5}	{2,3,4,5}
{2,3,4}	
{2,3,5}	
{3,4,5}	

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 $p.item_1 = q.item_1$,…, $p.item_{k-2} = q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$

F_{k-1}	C_k
{1,2,3}	{1,2,3,4}
{1,2,4}	{1,2,3,5}
{1,2,5}	{1,2,4,5}
{1,3,5}	{2,3,4,5}
{2,3,4}	
{2,3,5}	
{3,4,5}	

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 $p.item_1 = q.item_1$,…, $p.item_{k-2} = q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$

F_{k-1}	C_k
{1,2,3}	{1,2,3,4}
{1,2,4}	{1,2,3,5}
{1,2,5}	{1,2,4,5}
{1,3,5}	{2,3,4,5}
{2,3,4}	
{2,3,5}	
{3,4,5}	

aprioriGen – prune step

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).

F_{k-1}	C_k
{1,2,3}	~~{1,2,3,4}~~
{1,2,4}	{1,2,3,5}
{1,2,5}	{1,2,4,5}
{1,3,5}	{2,3,4,5}
{2,3,4}	
{2,3,5}	
{3,4,5}	

Itemset {1,3,4} not in F_{k-1}

CBA – Rule Generation (CBA-RG)

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
prCAR_1 = pruneRules(CAR_1);
for (k = 2; F_k-1 ≠ ∅; k++) do
    C_k = candidateGen(F_k-1);
    for each data case d ∈ D do
        C_d = ruleSubset(C_k, d);
        for each candidate c ∈ C_d do
            c.condsupCount++;
            if d.class = c.class then c.rulesupCount++
        end
    end
    F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
    CAR_k = genRules(F_k);
    prCAR_k = pruneRules(CAR_k);
end
CARs = ∪_k CAR_k;
prCARs = ∪_k prCAR_k;
```

ruleSubset() returns all the ruleitems in C_k whose condsets are supported by d.

Source: [1]

CBA – Rule Generation (CBA-RG)

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

Source: [1]

CBA – Rule Generation (CBA-RG)

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
prCAR_1 = pruneRules(CAR_1);
for (k = 2; F_{k-1} ≠ ∅; k++) do
    C_k = candidateGen(F_{k-1});
    for each data case d ∈ D do
        C_d = ruleSubset(C_k, d);
        for each candidate c ∈ C_d do
            c.condsupCount++;
            if d.class = c.class then c.rulesupCount++
        end
    end
    F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
    CAR_k = genRules(F_k);
    prCAR_k = pruneRules(CAR_k);
end
CARs = ∪_k CAR_k;
prCARs = ∪_k prCAR_k;
```

Only frequent rule items are retained.

Source: [1]

CBA – Rule Generation (CBA-RG)

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
prCAR_1 = pruneRules(CAR_1);
for (k = 2; F_k-1 ≠ ∅; k++) do
    C_k = candidateGen(F_k-1);
    for each data case d ∈ D do
        C_d = ruleSubset(C_k, d);
        for each candidate c ∈ C_d do
            c.condsupCount++;
            if d.class = c.class then c.rulesupCount++
        end
    end
    F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
    CAR_k = genRules(F_k);
    prCAR_k = pruneRules(CAR_k);
end
CARs = ∪_k CAR_k;
prCARs = ∪_k prCAR_k;
```

Source: [1]

CBA – Rule Generation (CBA-RG)

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

Source: [1]

CBA-RG side by side with apriori

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
prCAR_1 = pruneRules(CAR_1);
for (k = 2; F_{k-1} ≠ ∅; k++) do
    C_k = candidateGen(F_{k-1});
    for each data case d ∈ D do
        C_d = ruleSubset(C_k, d);
        for each candidate c ∈ C_d do
            c.condsupCount++;
            if d.class = c.class then c.rulesupCount++
        end
    end
    F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
    CAR_k = genRules(F_k);
    prCAR_k = pruneRules(CAR_k);
end
CARs = ∪_k CAR_k;
prCARs = ∪_k prCAR_k;
```

Source: [1]

```
L_1 = {large 1-itemsets};
for ( k = 2; L_{k-1} ≠ ∅; k++ ) do begin
    C_k = apriori-gen(L_{k-1}); // New candida
    forall transactions t ∈ D do begin
        C_t = subset(C_k, t); // Candidates con
        forall candidates c ∈ C_t do
            c.count++;
    end
    L_k = {c ∈ C_k | c.count ≥ minsup}
end
Answer = ∪_k L_k;
```

Source: [4]

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

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-Classifier Builder (CB 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

Rule ranking criteria

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

CBA-Classifier Builder (CB 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).
```

Data coverage pruning

Add the rule to the classifier if it classifies at least one instance correctly.

Remove all data cases covered by the rule.

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

CBA-Classifier Builder (CB 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).
```

Majority class in the remaining data. This will be used if r is the last rule in the final classifier.

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

CBA-Classifier Builder (CB 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).
```

Total number of errors made by the current set of rules in C and the default rule.

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

CBA-Classifier Builder (CB 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).
```

"Default rule pruning"

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

CBA-Classifier Builder (CB 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

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.

CBA-Classifier Builder (CB 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).
```

CBA-CB M1 is simple but inefficient – many passes over the database.

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

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"*

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-CB M2 Stage 1

```
Q = ∅; U = ∅; A = ∅;
for each case d ∈ D do
    cRule = maxCoverRule(C_c, d);
    wRule = maxCoverRule(C_w, d);
    U = U ∪ {cRule};
    cRule.classCasesCovered[d.class]++;
    if cRule ≻ wRule then
        Q = Q ∪ {cRule};
        mark cRule;
    else  A = A ∪ <d.id, d.class, cRule, wRule>
end
```

Source: [1], CBA-CB algorithm M2

Finds the highest precedence rule that covers d.
C_c is the set of rules having the same class as d.

cRule … the highest precedence rule that correctly classifies d

CBA-CB M2 Stage 1

```
Q = ∅; U = ∅; A = ∅;
for each case d ∈ D do
    cRule = maxCoverRule(C_c, d);
    wRule = maxCoverRule(C_w, d);
    U = U ∪ {cRule};
    cRule.classCasesCovered[d.class]++;
    if cRule ≻ wRule then
        Q = Q ∪ {cRule};
        mark cRule;
    else  A = A ∪ <d.id, d.class, cRule, wRule>
end
```

Source: [1], CBA-CB algorithm M2

Finds the highest precedence rule that covers d.

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

wRule … the highest precedence rule that incorrectly classifies d

CBA-CB M2 Stage 1

```
Q = ∅; U = ∅; A = ∅;
for each case d ∈ D do
    cRule = maxCoverRule(C_c, d);
    wRule = maxCoverRule(C_w, d);
    U = U ∪ {cRule};
    cRule.classCasesCovered[d.class]++;
    if cRule ≻ wRule then
        Q = Q ∪ {cRule};
        mark cRule;
    else  A = A ∪ <d.id, d.class, cRule, wRule>
end
```

Source: [1], CBA-CB algorithm M2

U is the set of all cRules.

CBA-CB M2 Stage 1

```
Q = ∅; U = ∅; A = ∅;
for each case d ∈ D do
    cRule = maxCoverRule(C_c, d);
    wRule = maxCoverRule(C_w, d);
    U = U ∪ {cRule};
    cRule.classCasesCovered[d.class]++;
    if cRule ≻ wRule then
        Q = Q ∪ {cRule};
        mark cRule;
    else A = A ∪ <d.id, d.class, cRule, wRule>
end
```

Source: [1], CBA-CB algorithm M2

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

CBA-CB M2 Stage 1

```
Q = ∅; U = ∅; A = ∅;
for each case d ∈ D do
    cRule = maxCoverRule(C_c, d);
    wRule = maxCoverRule(C_w, d);
    U = U ∪ {cRule};
    cRule.classCasesCovered[d.class]++;
    if cRule ≻ wRule then
        Q = Q ∪ {cRule};
        mark cRule;
    else  A = A ∪ <d.id, d.class, cRule, wRule>
end
```

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 1

$Q = \varnothing$; $U = \varnothing$; $A = \varnothing$;
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 $cRule \succ wRule$ **then**
        $Q = Q \cup \{cRule\}$;
        mark $cRule$;
    else $A = A \cup$ <d.id, d.class, $cRule$, $wRule$>
end

Source: [1], CBA-CB algorithm M2

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

CBA-CB M2 Stage 1

```
Q = ∅; U = ∅; A = ∅;
for each case d ∈ D do
    cRule = maxCoverRule(C_c, d);
    wRule = maxCoverRule(C_w, d);
    U = U ∪ {cRule};
    cRule.classCasesCovered[d.class]++;
    if cRule ≻ wRule then
        Q = Q ∪ {cRule};
        mark cRule;
    else A = A ∪ <d.id, d.class, cRule, wRule>
end
```

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

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 1

```
Q = ∅; U = ∅; A = ∅;
for each case d ∈ D do
    cRule = maxCoverRule(C_c, d);
    wRule = maxCoverRule(C_w, d);
    U = U ∪ {cRule};
    cRule.classCasesCovered[d.class]++;
    if cRule ≻ wRule then
        Q = Q ∪ {cRule};
        mark cRule;
    else  A = A ∪ <d.id, d.class, cRule, wRule>
end
```

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:

<dID, y, cRule, wRule>,

dID … id of the case d

y … the class of d

CBA-CB M2 Stage 2

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

Source: [1], CBA-CB algorithm M2

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

CBA-CB M2 Stage 2

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

Source: [1], CBA-CB algorithm M2

The algorithm accepts the error. The case d will be classified by *wRule*.

Since in stage 1, d was counted under cRule, the algorithm subtracts *d* from the number of cases covered by *cRule*, and increments the number of cases covered by wRule.

```
Q = ∅; U = ∅; A = ∅;
for each case d ∈ D do
      cRule = maxCoverRule(C_c, d);
      wRule = maxCoverRule(C_w, d);
      U = U ∪ {cRule};
      cRule.classCasesCovered[d.class]++;
```

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

CBA-CB M2 Stage 2

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

Source: [1], CBA-CB algorithm M2

The algorithm accepts the error. The case d will be classified by *wRule*.

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.

CBA-CB M2 Stage 2

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

Source: [1], CBA-CB algorithm M2

Since *wRule* is not marked, it does not act as a *cRule* for another rule. However, there may be multiple higher precedence rules (than cRule) that cover d and classify it incorrectly.

allCoverRules() returns all rules that wrongly classify dID and have higher precedence than cRule.
It processed only the rules in U, which is the set of all cRules.

CBA-CB M2 Stage 2

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

Source: [1], CBA-CB algorithm M2

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 cover d 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.

CBA-CB M2 Stage 2

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

Source: [1], CBA-CB algorithm M2

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 cover d and classify it incorrectly.

Indicates that the rule might cover the case dID.

CBA-CB M2 Stage 2

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

Source: [1], CBA-CB algorithm M2

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.

CBA-CB M2 Stage 3

In stage 3, the algorithm chooses the final set of rules.

```
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;
```

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 3

In stage 3, the algorithm chooses the final set of rules.

Counts the number of training cases in each class in the initial 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;
```

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 3

```
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;
```

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

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 3

```
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;
```

Source: [1], CBA-CB algorithm M2

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

CBA-CB M2 Stage 3

```
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;
```

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

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

```
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;
```

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.

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 3

```
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;
```

Errors caused by the current rule and previously processed higher precedence rules.

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 3

```
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;
```

Source: [1], CBA-CB algorithm M2

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

Update class distributions (presumably by removing class counts associated with the rule (in r.classCasesCovered[class]).

CBA-CB M2 Stage 3

```
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.

Majority class in the remaining training data.

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 3

```
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;
```

Source: [1], CBA-CB algorithm M2

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.

CBA-CB M2 Stage 3

```
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;
```

r is added to the final rule list

Source: [1], CBA-CB algorithm M2

CBA-CB M2 Stage 3

Final rule pruning (3rd)

```
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;
```

Source: [1], CBA-CB algorithm M2

"default rule pruning"

CBA-CB M2 Stage 3

Output of CBA-RG

w pruning set to on

rules in the classifier built with CBA-CB, w pruning set to on

Datasets	c4.5rules w/o discr.	c4.5rules discr.	CBA (CARs + infreq) w/o pru.	CBA (CARs + infreq) pru.	CBA (CARs) w/o pru	CBA (CARs) pru.	No. of CARs w/o pru.	No. of CARs pru.	Run time (sec) (CBA-RG) w/o pru	Run time (sec) (CBA-RG) pru.	Run time (sec) (CBA-CB) M1	Run time (sec) (CBA-CB) M2	No. of Rules in C
anneal*	5.2	6.5	1.9	1.9	3.2	3.6	65081	611	14.33	14.36	0.08	0.06	34
australian*	15.3	13.5	13.5	13.4	13.2	13.4	46564	4064	5.00	5.05	0.20	0.22	148
auto*	19.9	29.2	21.0	23.1	24.0	27.2	50236	3969	3.30	3.55	0.12	0.06	54
breast-w	5.0	3.9	3.9	3.9	4.2	4.2	2831	399	0.30	0.33	0.02	0.03	49
cleve*	21.8	18.2	18.1	19.1	16.7	16.7	48854	1634	4.00	4.30	0.04	0.06	78
crx*	15.1	15.9	14.3	14.3	14.1	14.1	42877	4717	4.90	5.06	0.43	0.30	142
diabetes	25.8	27.6	24.8	25.5	24.7	25.3	3315	162	0.25	0.28	0.03	0.01	57
german*	27.7	29.5	27.2	26.5	25.2	26.5	69277	4561	5.60	6.00	1.04	0.28	172
glass	31.3	27.5	27.4	27.4	27.4	27.4	4234	291	0.20	0.22	0.02	0.00	27
heart	19.2	18.9	19.6	19.6	18.5	18.5	52309	624	4.70	4.60	0.04	0.03	52
hepatitis*	19.4	22.6	15.1	15.1	15.1	15.1	63134	2275	2.80	2.79	0.09	0.05	23
horse*	17.4	16.3	18.2	17.9	18.7	18.7	62745	7846	3.2	3.33	0.35	0.19	97
hypo*	0.8	1.2	1.6	1.6	1.9	1.7	37631	493	45.60	45.30	1.02	0.40	35
ionosphere*	10.0	8.0	7.9	7.9	8.2	8.2	55701	10055	3.75	4.00	0.56	0.41	45
iris	4.7	5.3	7.1	7.1	7.1	7.1	72	23	0.00	0.00	0.00	0.00	5
labor	20.7	21.0	17.0	17.0	17.0	17.0	5565	313	0.17	0.20	0.00	0.00	12
led7	26.5	26.5	27.8	27.8	27.8	27.8	464	336	0.40	0.45	0.11	0.10	71
lymph*	26.5	21.0	20.3	18.9	20.3	19.6	40401	2965	2.70	2.70	0.07	0.05	36
pima	24.5	27.5	26.9	27.0	27.4	27.6	2977	125	0.23	0.25	0.04	0.02	45
sick*	1.5	2.1	2.8	2.8	2.7	2.7	71828	627	32.60	33.40	0.62	0.40	46
sonar*	29.8	27.8	24.3	21.7	24.3	21.7	57061	1693	5.34	5.22	0.30	0.12	37
tic-tac-toe	0.6	0.6	0.0	0.0	0.0	0.0	7063	1378	0.62	0.71	0.12	0.08	8
vehicle*	27.4	33.6	31.3	31.2	31.5	31.3	23446	5704	6.33	6.33	1.40	0.40	125
waveform*	21.9	24.6	20.2	20.2	20.4	20.6	9699	3396	13.65	13.55	2.72	1.12	386
wine	7.3	7.9	8.4	8.4	8.4	8.4	38070	1494	2.34	2.65	0.11	0.04	10
zoo*	7.8	7.8	5.4	5.4	5.4	5.4	52198	2049	2.73	2.70	0.61	0.32	7
Average	16.7	17.1	15.6	15.6	15.7	15.8	35140	2377	6.35	6.44	0.39	0.18	69

Source: [1]

Outline

Classification based on associations (CBA)

In detail description of the CBA algorithm, based on the paper of Liu et al (1998).

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

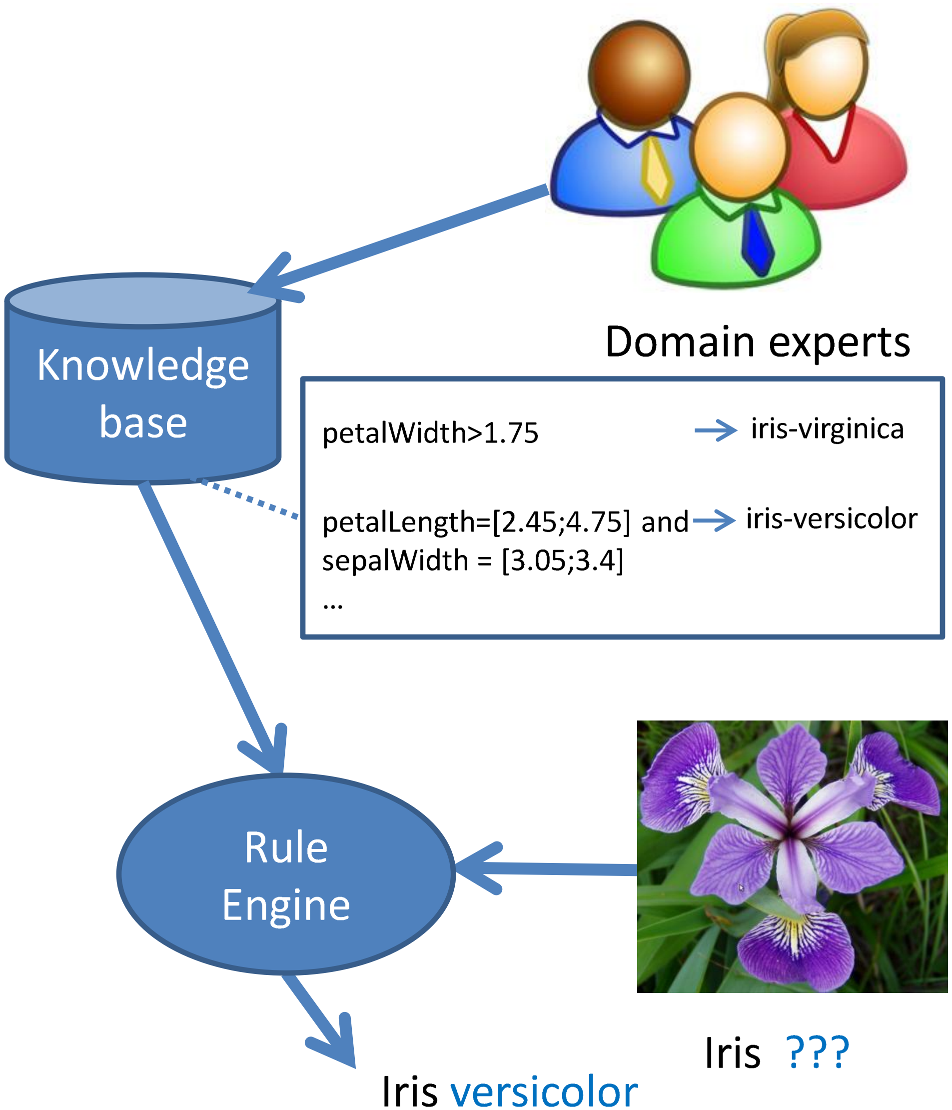
with Association Rule Classifiers

Presented at RuleML2014
(abridged updated version)

Tomáš Kliegr[1,4], Jaroslav Kuchař[1,2], Davide Sottara[3], Stanislav Vojíř[1]

[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



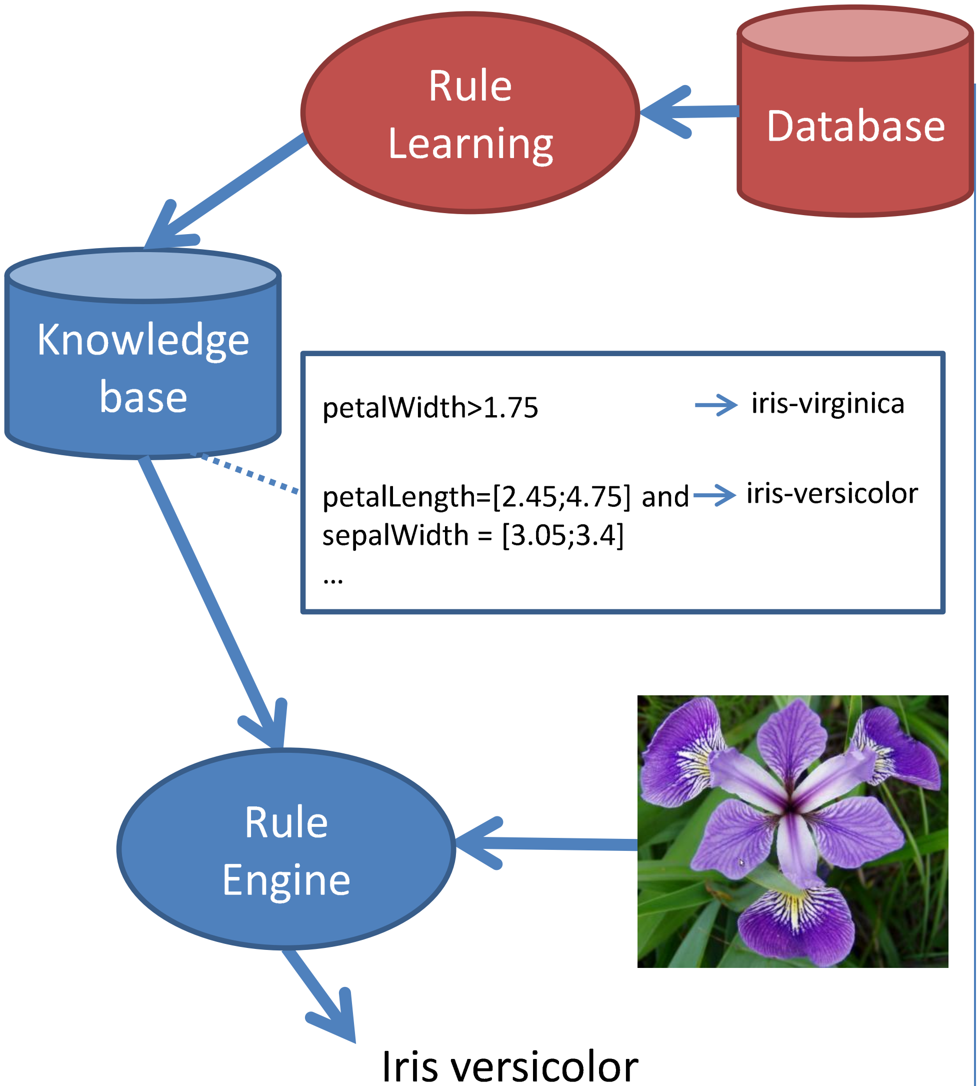
With Business Rule Management System (BRMS) applications can invoke decision logic which is input in the form of rules, instead of procedural code

+ This reduces reliance on the IT experts

- Requires extensive subject matter expertise
- (A lot of) Expert time

RESTRICTION: We focus on “classification business rules”.

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 yields rule sets that are

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

Problem statement

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

R1: petalWidth>1.75 → iris-virginica, supp= 0.296, conf=1

R2: petalWidth>1.75 and sepalWidth = [3.05;3.4] → iris-virginica supp= 0.100, conf=1

…

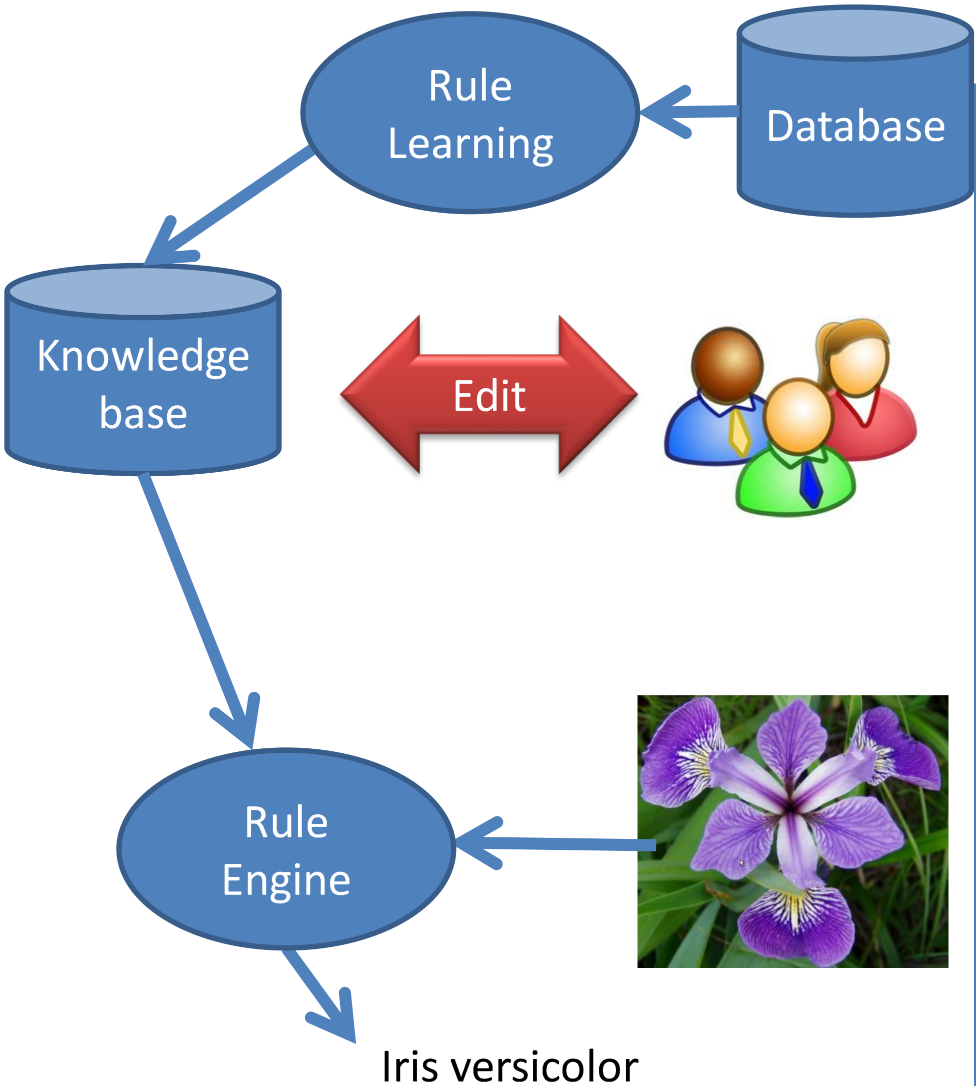
R9: sepalLength= (5.55;3.40] and sepalWidth<3.05 → iris-versicolor supp=0.230, conf=0.05

… 50 more rules

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

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

- 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

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

```
rules := sort rules according to criteria
for all rule ∈ rules do
    matches:= set of objects from T that match both rule ant. and conseq.
    if matches==∅ then
        remove rule from rules
    else
        remove matches from T
    end if
end for
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

Dataset	Rows	Attributes
Iris	150	4
Glass	214	9

Preprocessing

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

Rule learning

- LISp-Miner implementation, apriori-like setup

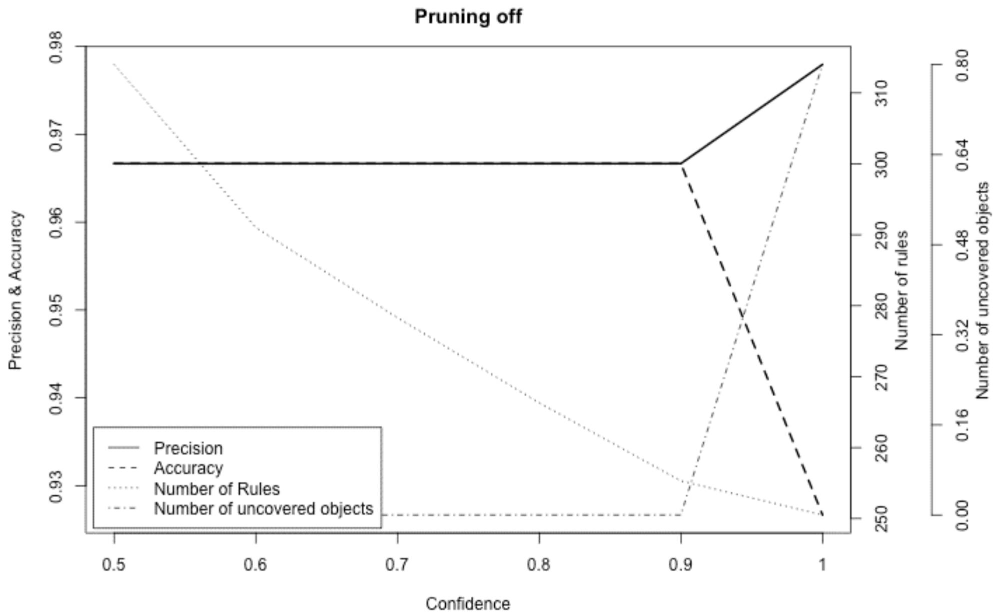
Pruning

- Data coverage pruning on/off

Experiment objectives

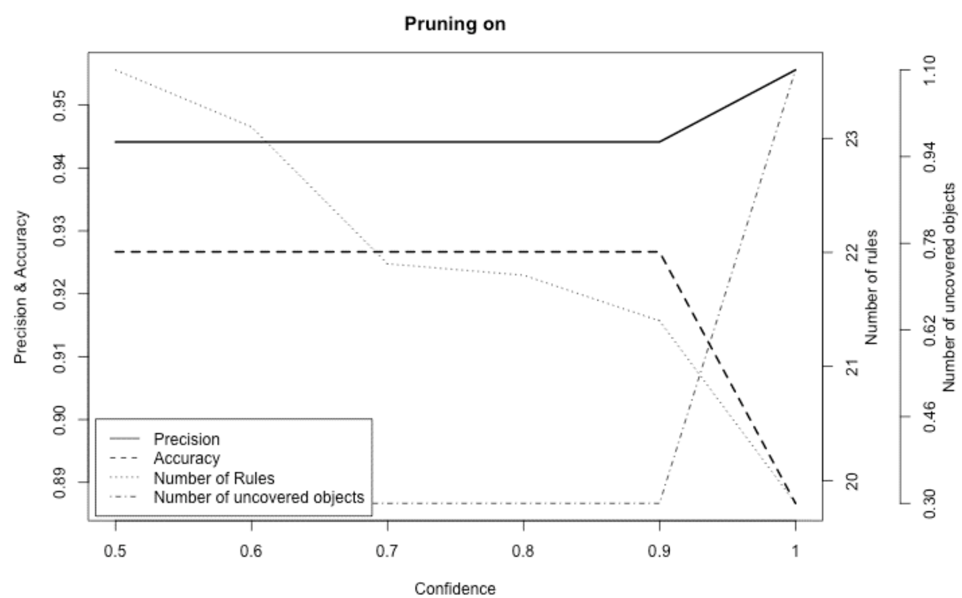
1) Compare results with other classifiers
2) Determine impact of:
- minSupport thr.
- minConfidence thr.
- pruning

Experimental results pruning



Effect of pruning. Iris dataset, minimum support threshold 1

iris dataset



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)

Experimental results minSupp

Dataset, task	support	not pruned Rules	not pruned Accuracy	pruned Rules	pruned Accuracy
iris	10	87	0.940	19	0.920
"	2	168	0.947	21	0.913
"	1	291	**0.967**	23	0.927
iris, sequence 1-2	10	904	0.940	17	0.953
"	2	1661	0.953	19	**0.960**
"	1	2653	**0.960**	19	**0.960**
glass	10	32	0.464	21	0.464
"	2	2374	**0.622**	68	0.608
balance scale	10	124	**0.891**	78	0.870
"	2	558	0.841	216	0.714
balance scale, subset 1-2	10	11947	0.758	153	**0.779**

Impact of minimum support threshold, minConf=0.6

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

Experimental results minConf

confidence	not pruned Rules	Accuracy	pruned Rules	Accuracy
0.5	58.3	0.529	25.8	**0.534**
0.6	31.8	0.464	21.1	0.464
0.7	10.3	0.290	8.4	0.286
0.8	2.4	0.117	1.8	0.117
0.9	0.4	0.010	0.2	0.010

Glass, minSupp=10 objects (5.18%)

confidence	not pruned Rules	Accuracy	pruned Rules	Accuracy
0.5	96	0.940	20	0.920
0.6	87	0.940	19	0.920
0.7	83	0.940	17	0.920
0.8	76	0.940	17	0.920
0.9	68	0.900	15	0.880

Iris, minSupp=10 objects (1.78%)

confidence	not pruned Rules	Accuracy	pruned Rules	Accuracy
0.6	124	0.891	78	0.870
0.7	86	0.875	70	0.864
0.8	50	0.790	50	0.782
0.9	24	0.547	24	0.547
1.0	1	0.047	1	0.047

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

Dataset	Rows	Attributes	Bins after preprocessing
Iris	150	4	18
BalanceScale	625	4	20
Glass	214	9	19

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.

Original intervals
[20-25)
[25-30)
[30-35)
[35-40)
[40-45)
[45-50)

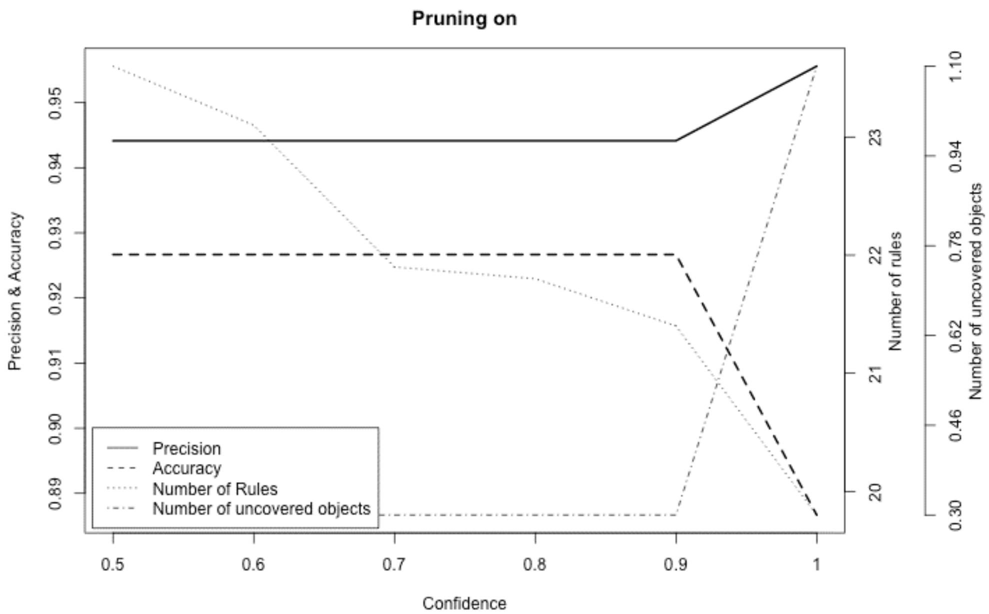
Sequence 1-2
Original intervals...
[20-25),[25-30)
[25-30),[30-35)
[30-35),[35-40)
[35-40),[40-45)
...

Subset 1-2
Original intervals...
[20-25),[25-30)
[20-25),[30-35)
[20-25),[35-40)
[20-25),[40-45)
[25-30),[30-35)
...

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

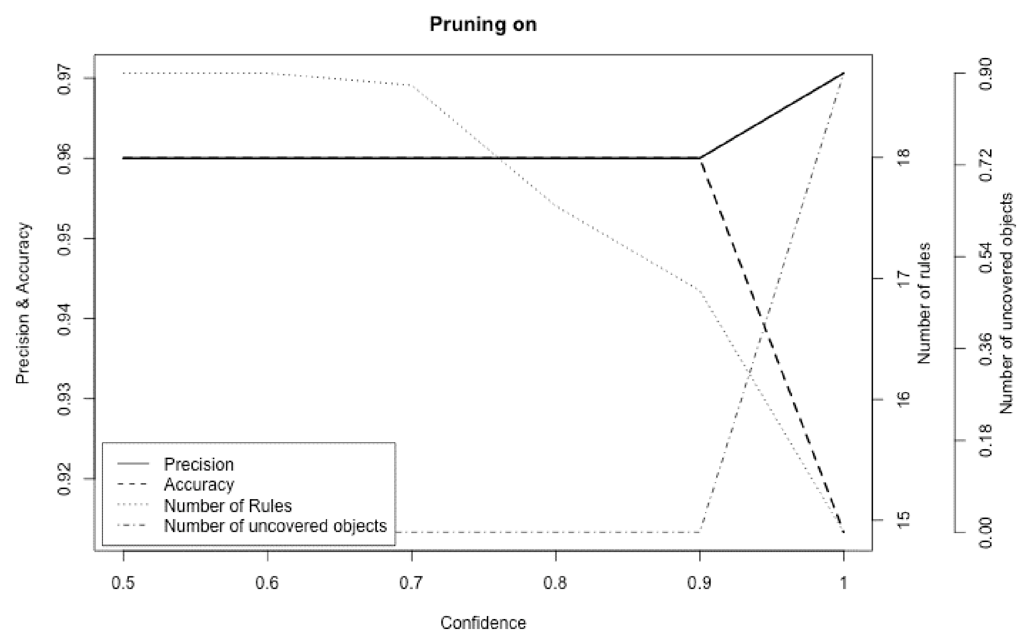
Effect of dynamic binning on cardinal attributes.
Iris dataset



Dynamic binning off

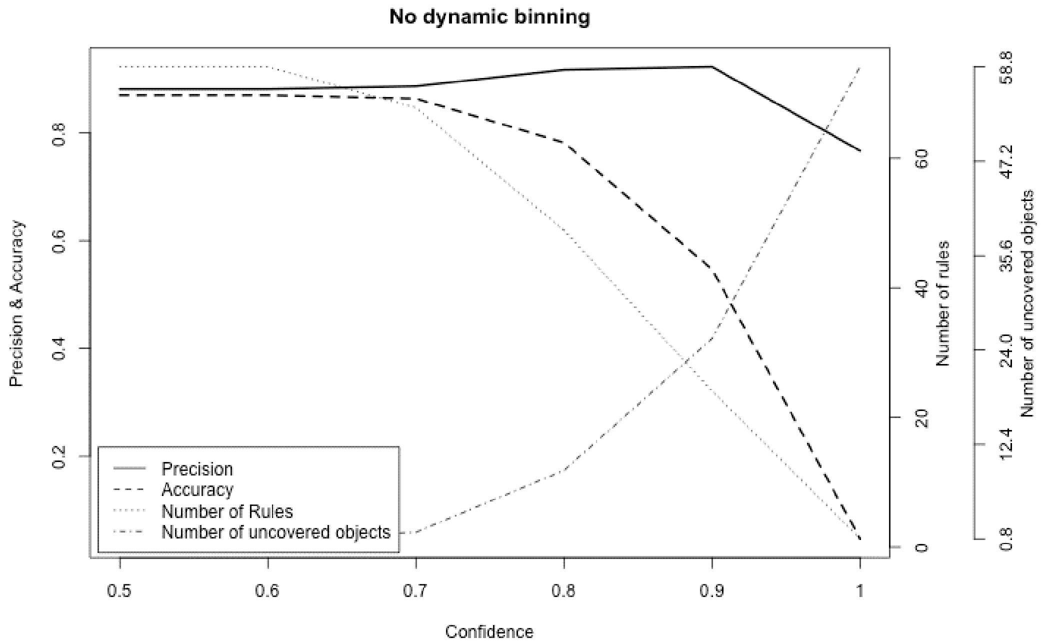
sepalWidth = [3.2;3.44) or
sepalWidth = [3.44;3.68) =>
XClass(Iris-setosa)

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



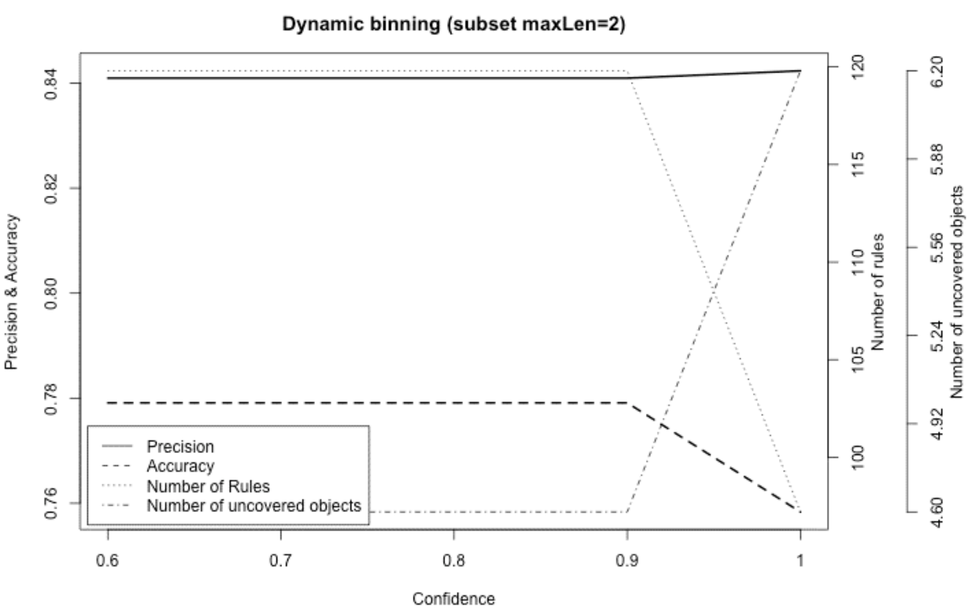
Dynamic binning on

Experimental results dynamic binning



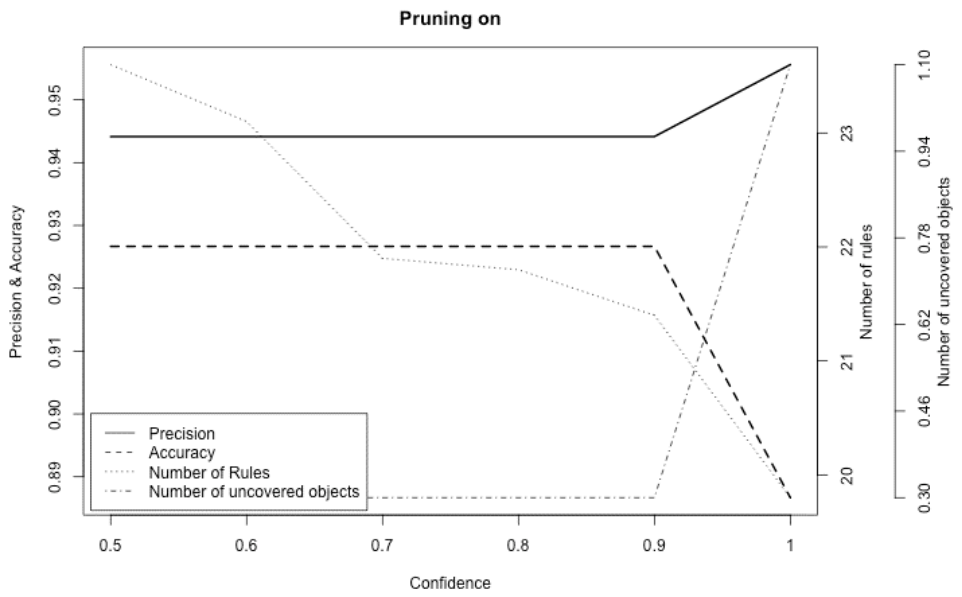
Effect of dynamic binning on nominal attributes.
Balancescale dataset

(LeftDistance=S or LeftDistance=M) and (LeftWeight=L or LeftWeight=H) => XClass=L



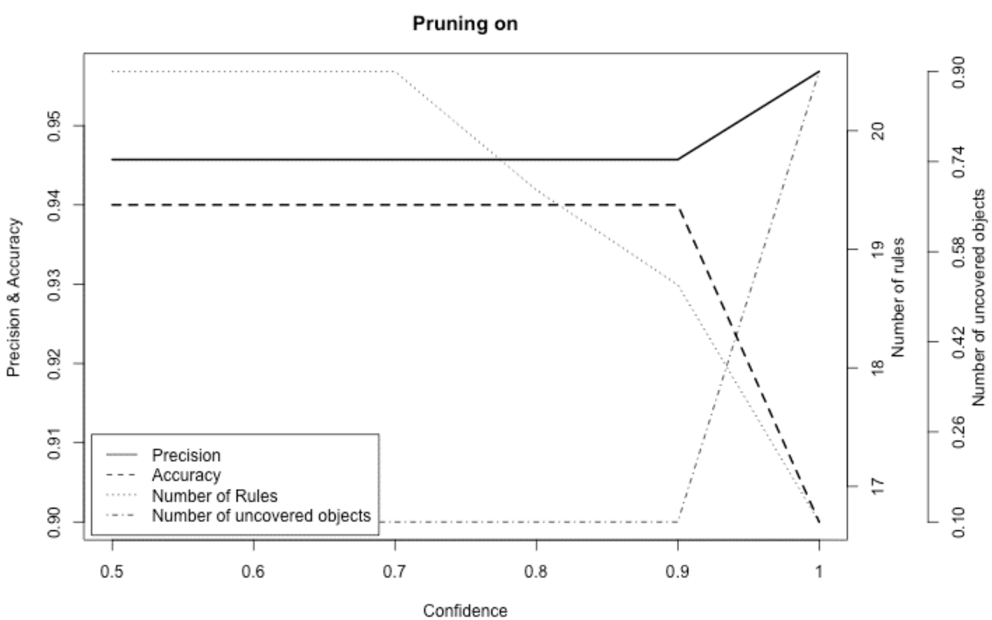
Dynamic binning (nominal attributes)– worse accuracy, higher rule count and **drastically** longer learning time.

Experimental results negative literals



Effect of including negative literals.
Iris dataset

petalLength =[1;1.59)
and petalWidth=[0.1;0.34)
and not(sepalLength=[4.3;4.66)
and not(sepalWidth=[2;2.34)
=> XClass(Iris-setosa)



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

Experimental results time complexity

Dataset/Task	Attributes	Verifications	Rules	Mining duration
	without binning	315	80	less than 1 s
	with negations	13 542	2 472	12 s
	disjunctions (nominal)	19 413	4 715	27 s
BalanceScale (min Conf 0,5)	without binning	510	146	less than 1 s
	with negations	33 045	9 040	43 s
	disjunctions (nominal)	73 230	17 004	99 s
	disjunctions (cardinal)	9 582	2 122	10 s
	disjunctions (cardinal – 3 values)	45 915	11 846	75 s
Glass (min Conf 0,9)	without binning	3 920	24	less than 1 s
	with negations	669 075	8 146	64 s
	dynamic binning	*not suitable (attributes have only 2 values)*		

Experimental results overview

	previously reported results					brCBA	
dataset	C4.5	ripper	cba	cmar	cpar	not pruned	pruned
iris	0.953	0.940	0.947	0.940	0.947	**0.967**	0.960
glass	0.687	0.691	**0.739**	0.701	0.744	0.622	0.612

Monotonicity Exploiting Association Rule Classification

(tentative title)

Working draft

Tomáš Kliegr

Multimedia and Vision Research Group
Queen Mary
University of London

Supervisors:

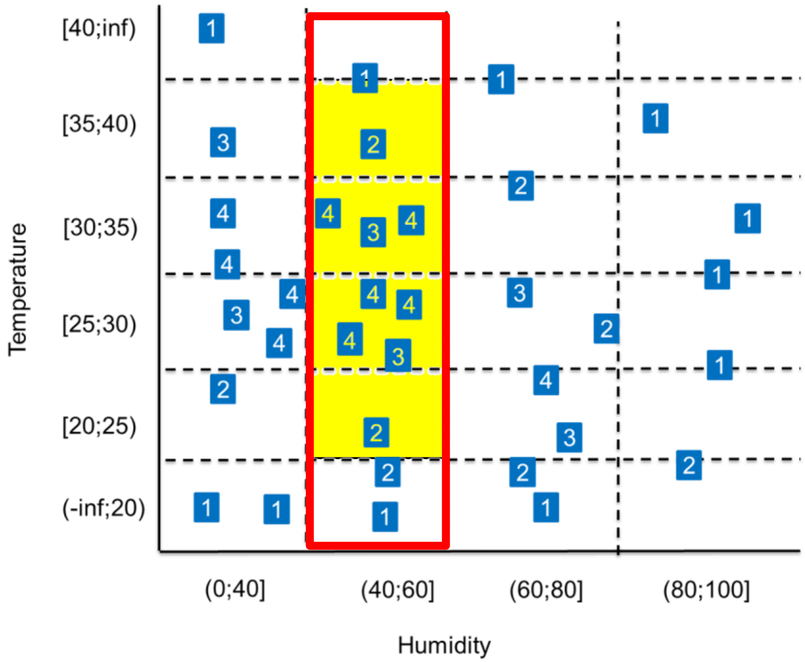
Prof. Ebroul Izquierdo
Multimedia & Vision Group
Queen Mary University of London

Dr. Christopher Tyson
Department of Economics, Queen Mary
Queen Mary University of London

Limitations of CBA

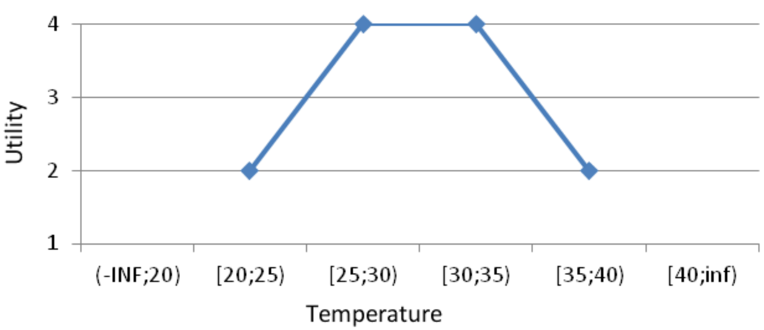
Association rules identify only the high density regions in the data, which have a strong presence of one target class.

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



Humidity=(40;60] & Temperature=[20;25) => Utility=2
Humidity=(40;60] & Temperature=[25;30) => Utility=4
Humidity=(40;60] & Temperature=[30;35) => Utility=4
Humidity=(40;60] & Temperature=[35;40) => Utility=2

Corresponding
Conditional utility model

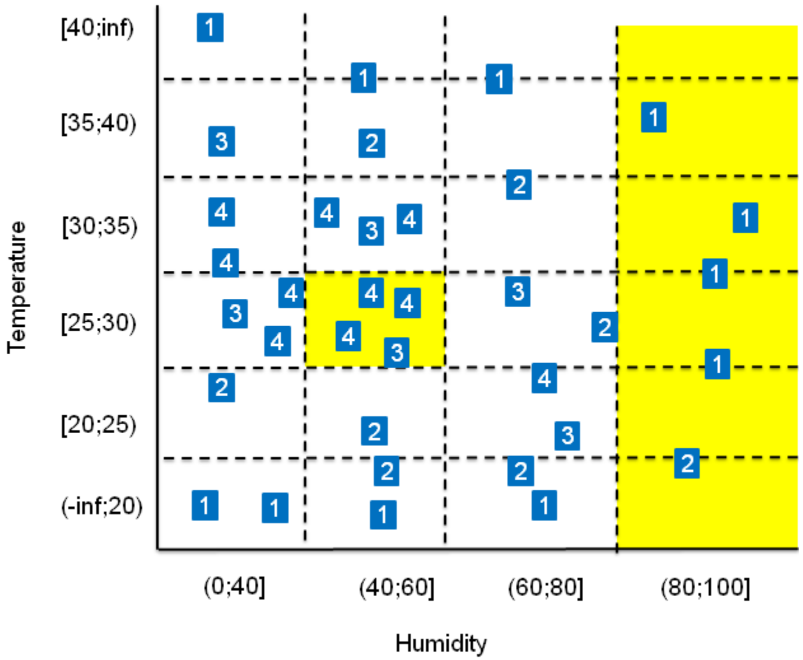


Ceteris paribus: Humidity = (40;60]

Rules output with minConf = 0.6 and minSupp =1

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**

Monotonicity Exploiting Association Rule Classification

The MARC algorithm was proposed to address these challenges.

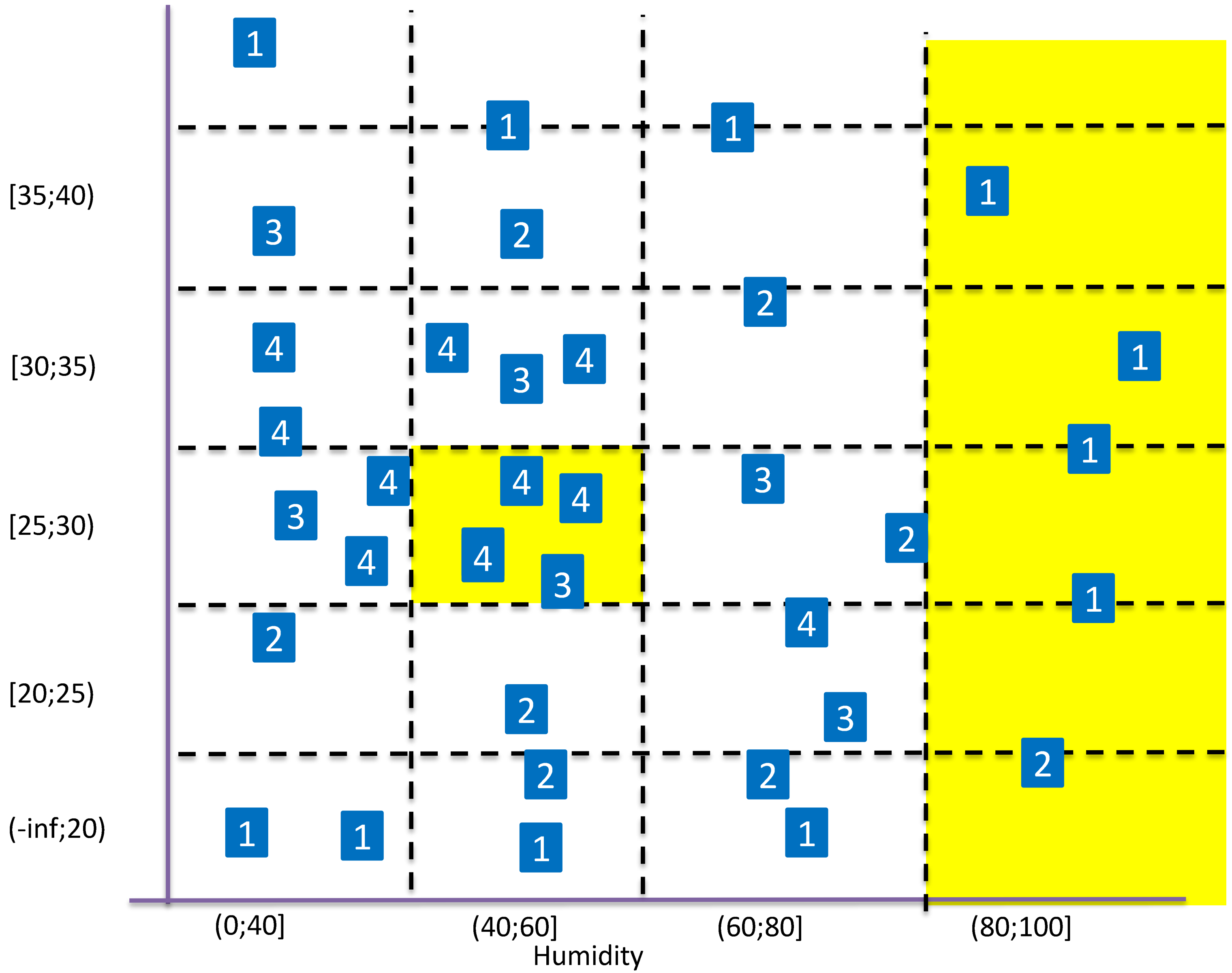
Three fundamental steps:

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

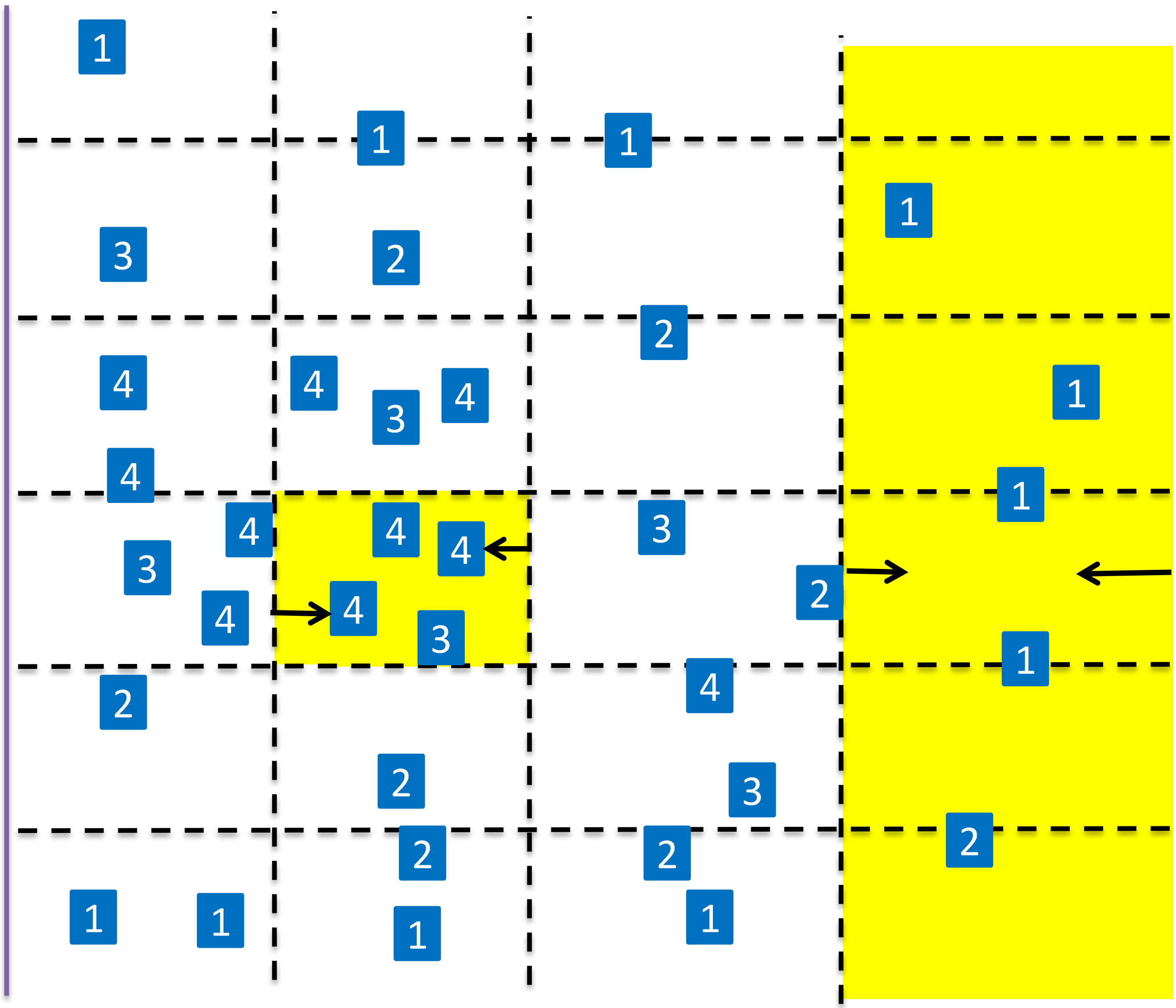
MARC consists of several consecutive procedures:

- 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

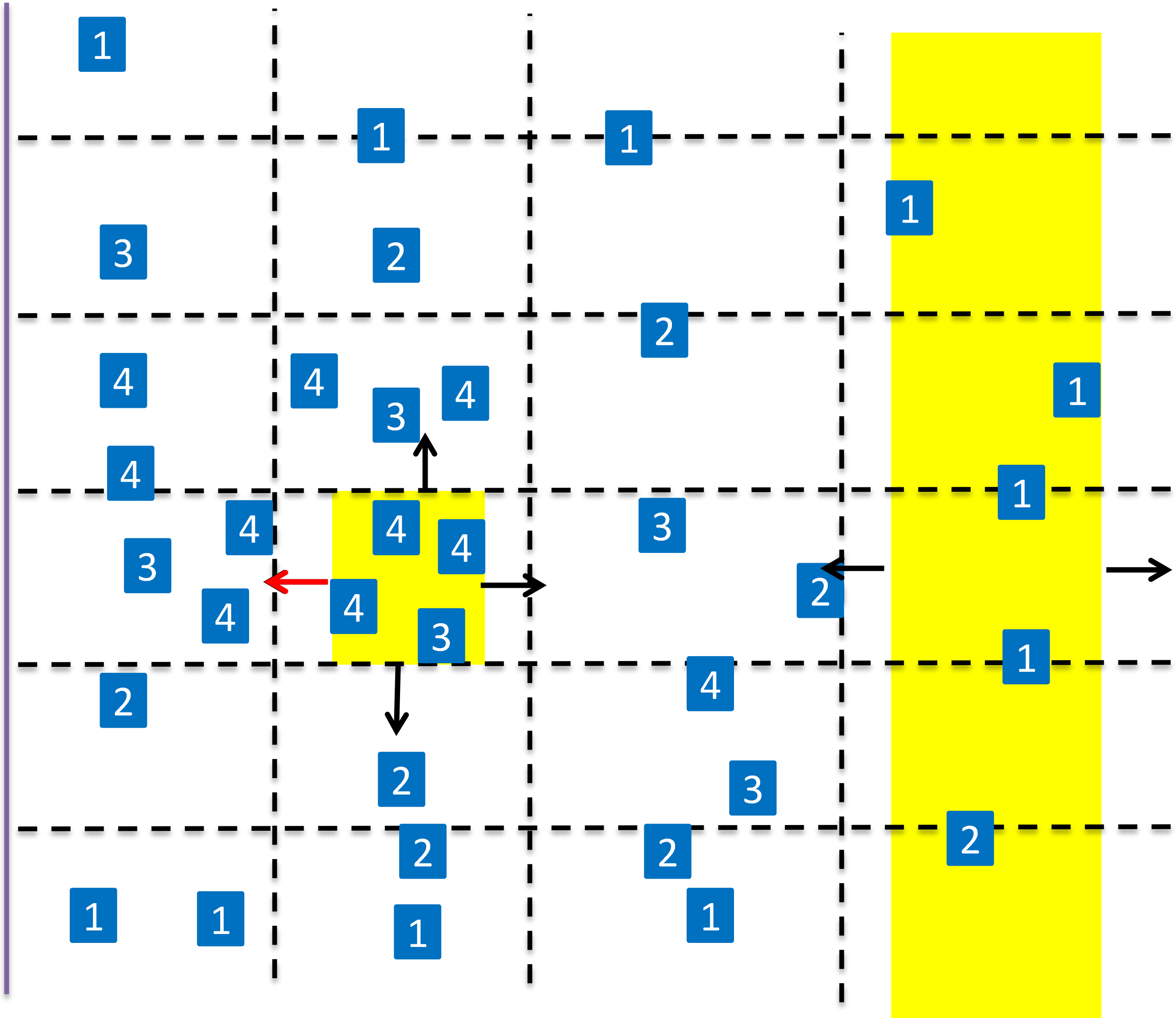


Step 2: extend association rules



1) The rule boundaries are adjusted to supporting points in he original feature space

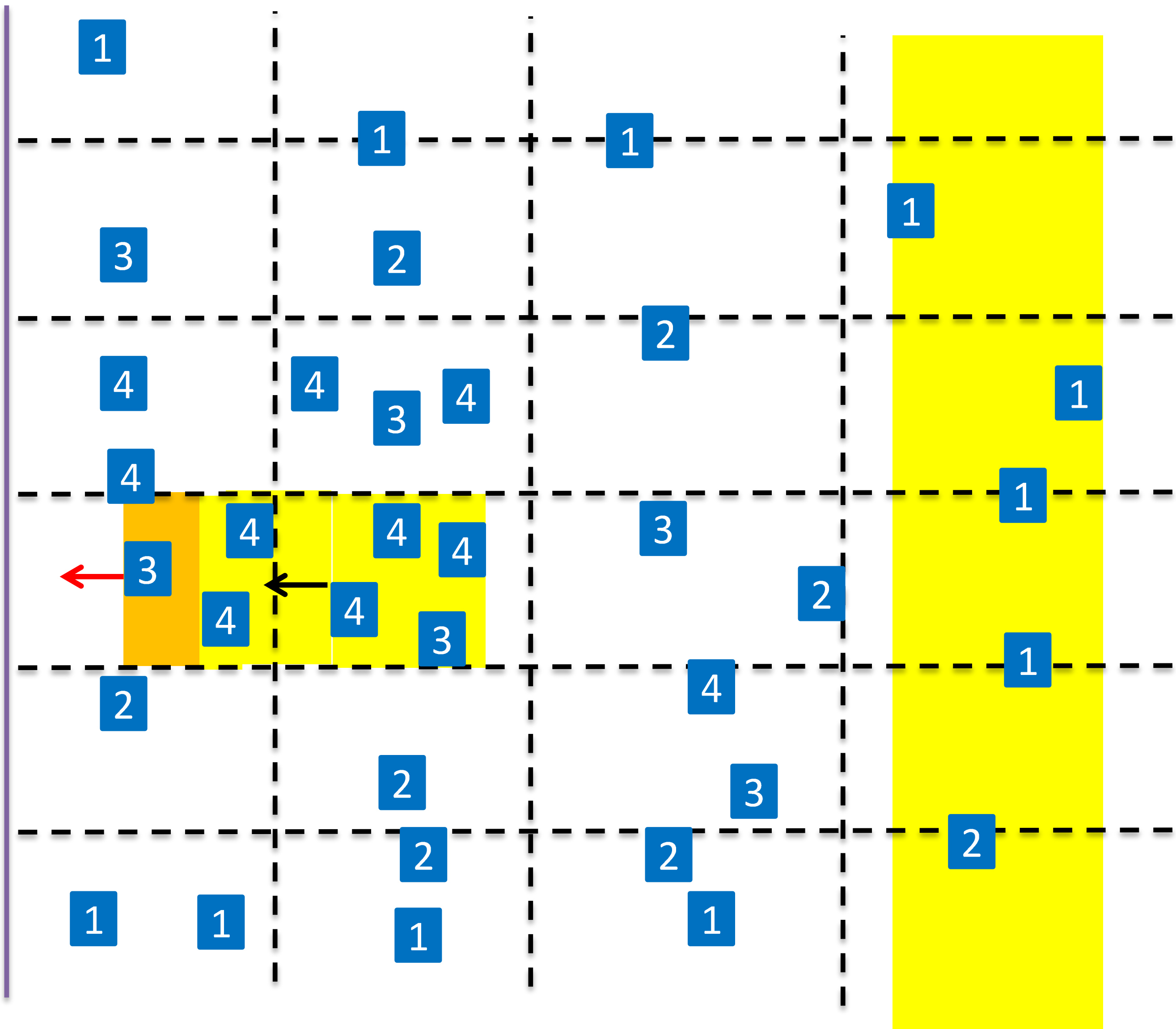
Step 2: extend association rules



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.

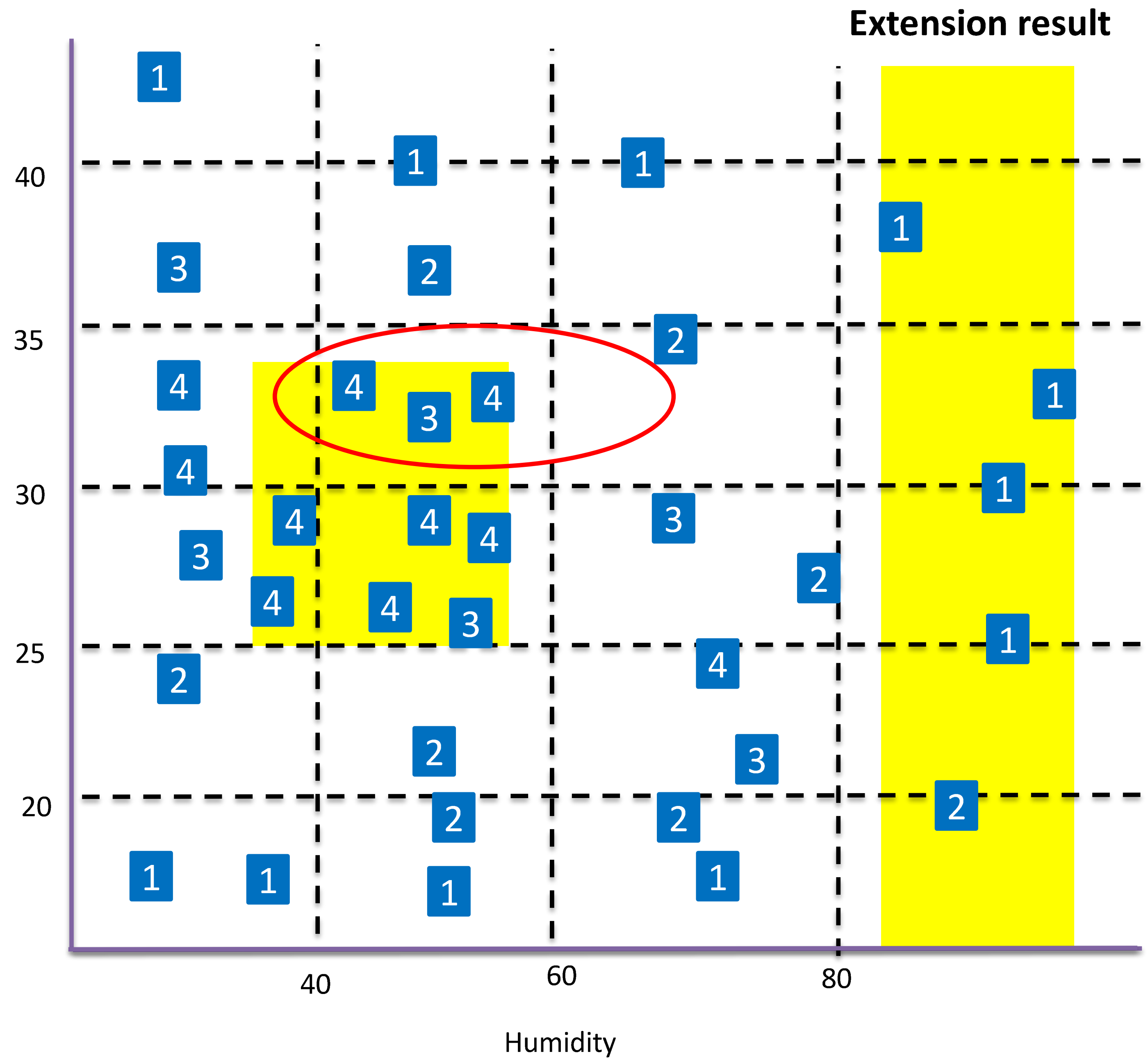
The extension is performed to the last point whose inclusion improves rule confidence

Conditional accept

Further extension is not possible

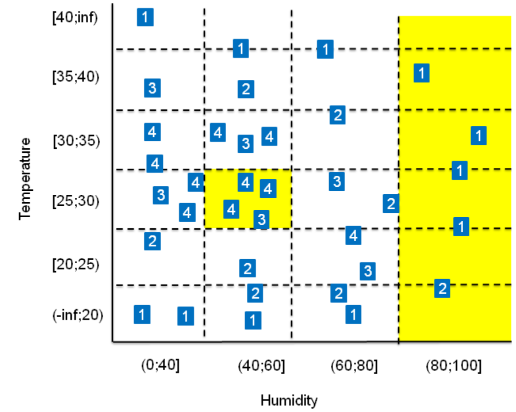
Extension retracts

Step 2: extend association rules

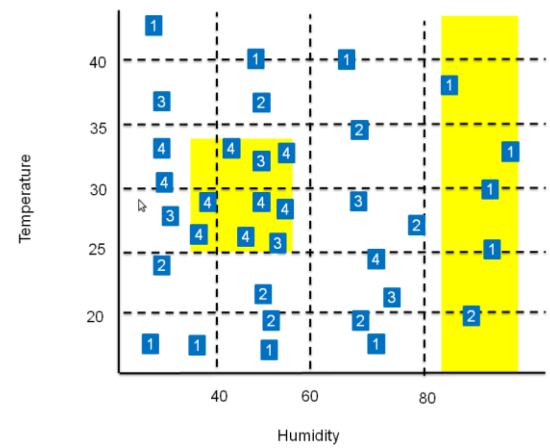


Step 2: extend association rules

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

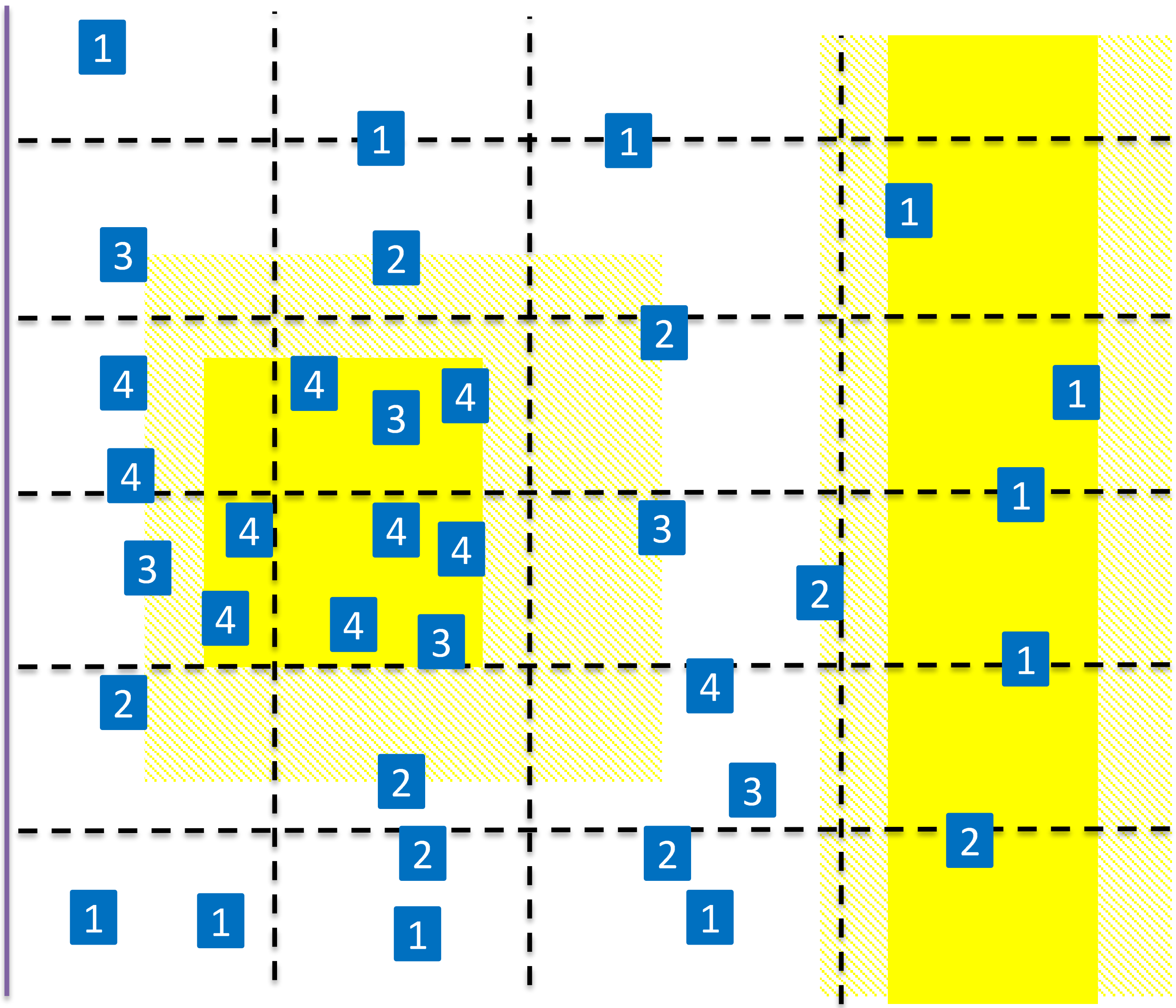


Through rule extension, these rules were enlarged and refined to
Humidity=(38;58] & Temperature=[25;34) => Utility=4
Humidity=(85;95] => Utility=2



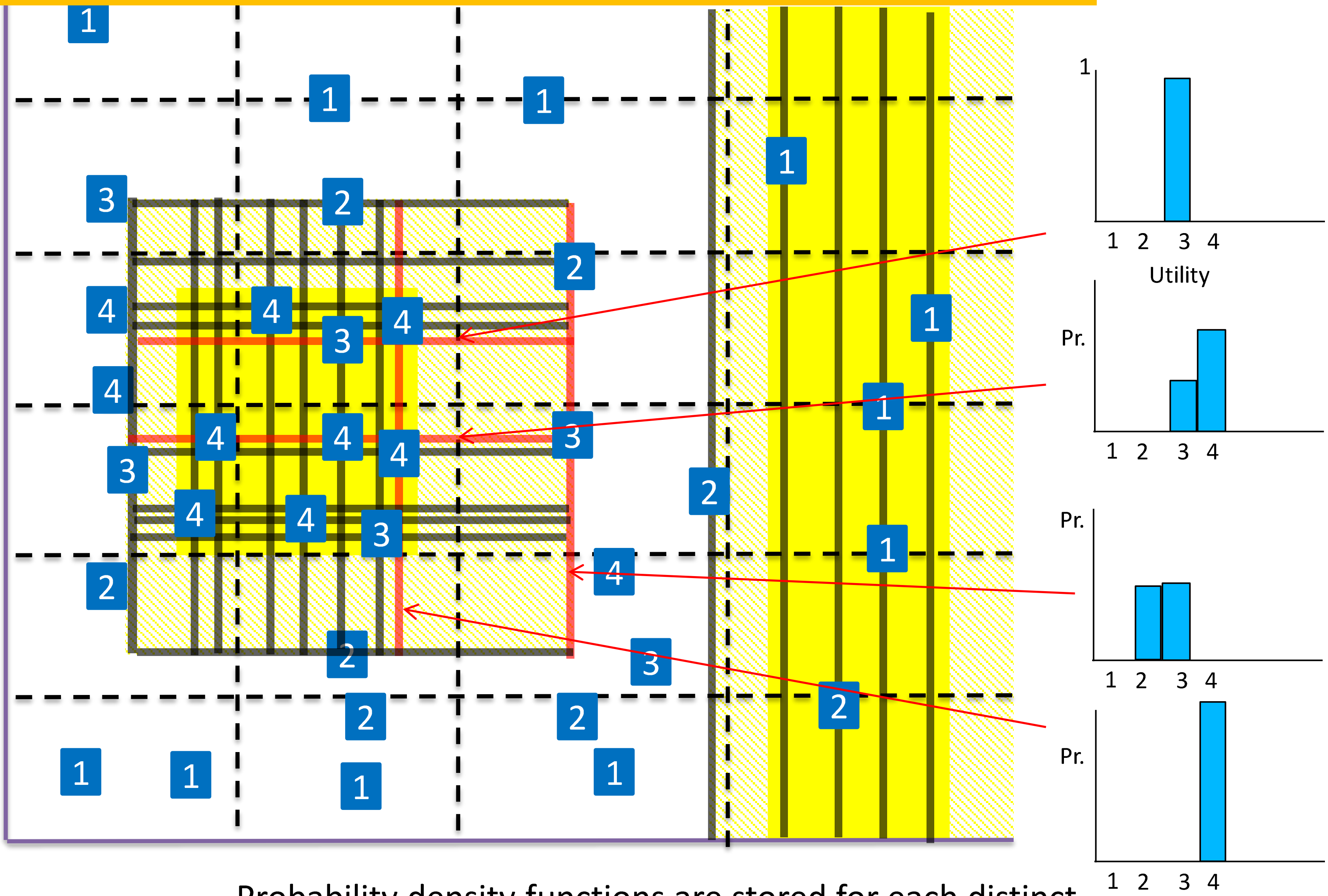
To further extend the coverage of the instance space,
Rules are extended by appending fuzzy borders

Step 3: fuzzify rules



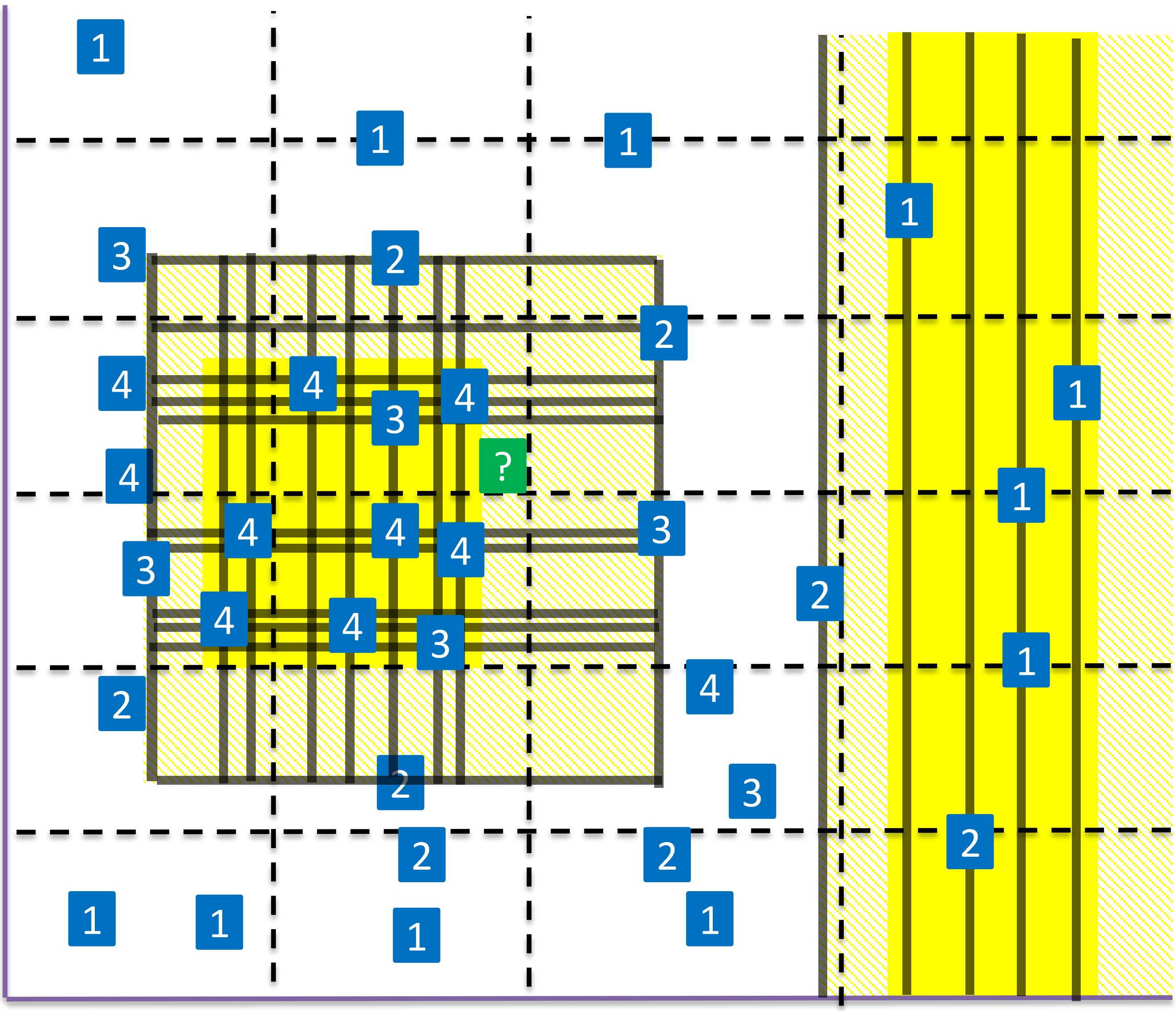
The coverage of each literal created over a cardinal attribute in the body of a rule is extended by appending a value adjacent to the lowest and highest values.

Step 4: annotate rules with probability distributions



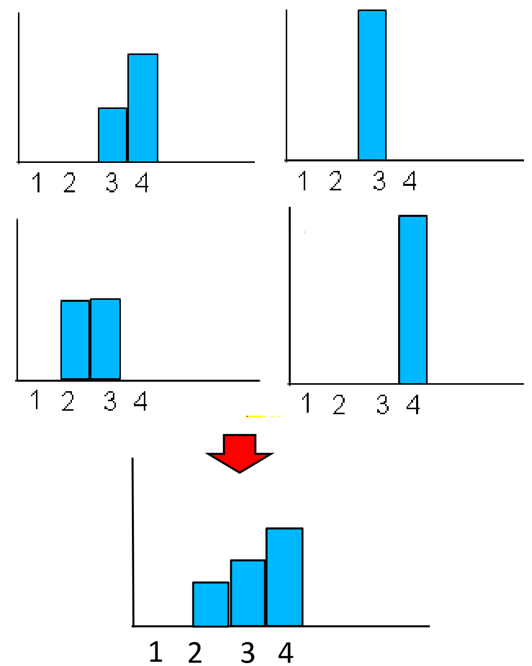
Probability density functions are stored for each distinct feature value across all points in the training data that are covered by the rule.

Step 2: aggregate distributions for matching rule

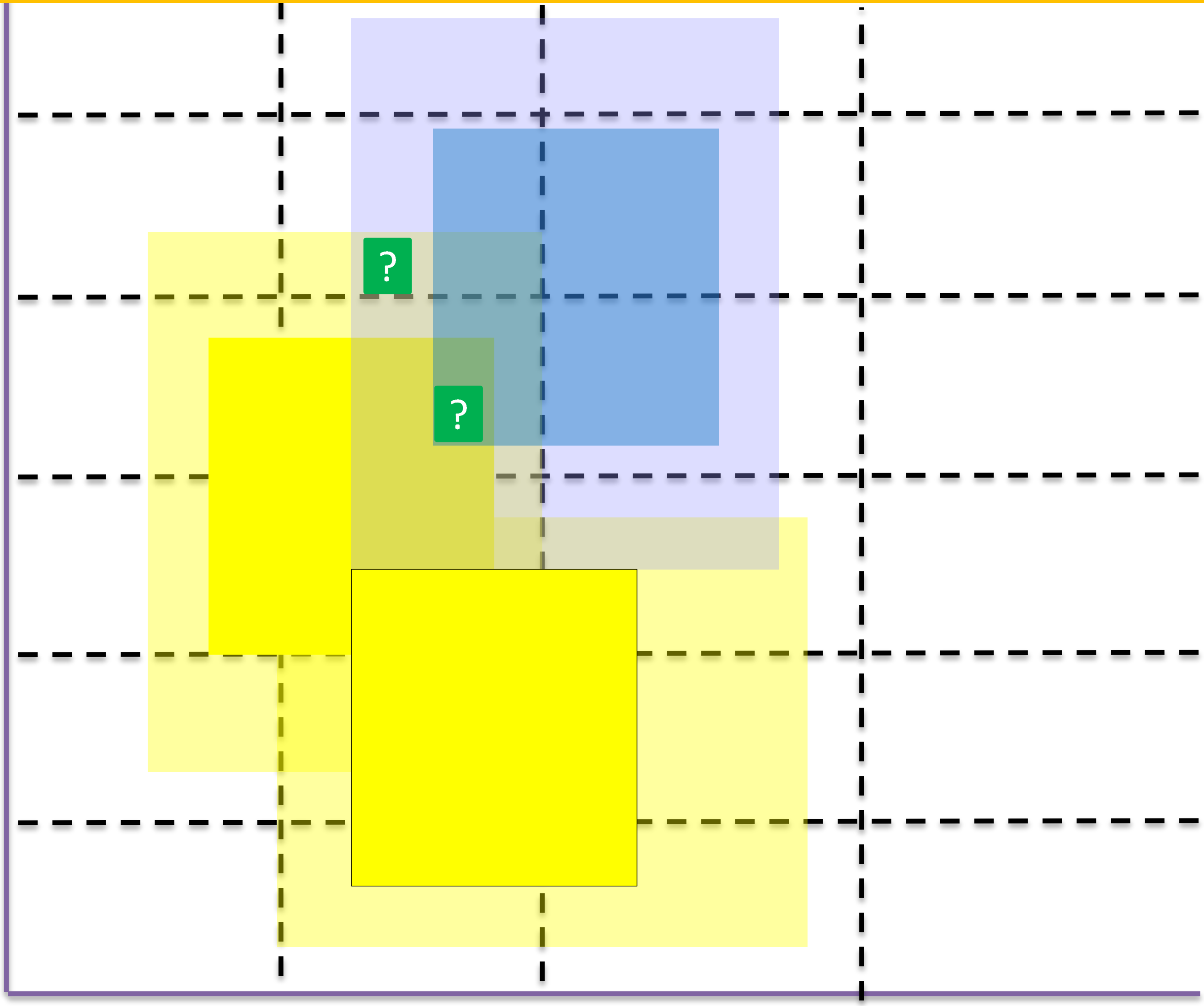


1) Locate the nearest supporting annotated line segments

2) Aggregate the probability density functions



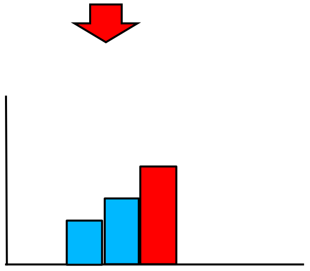
Step 2: apply model - aggregate rules



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

Preliminary experimental results

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

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

Bibliography

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

Slides http://www.comp.nus.edu.sg/~dm2/publications/kdd98slides.ps

[2] Tomáš Kliegr, Jaroslav Kuchař, Davide Sottara, Stanislav Vojíř: Learning Business Rules with Association Rule Classifiers. RuleML 2014: 236-250

[3] LUCS CBA implementation http://cgi.csc.liv.ac.uk/~frans/KDD/Software/CBA/cba.html

[4] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 487-499.

[5] J. Ross Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

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