Classification

CS145

Fall 2014

Classification based on Association

- Classification rule mining versus Association rule mining
  - Aim
    - A small set of rules as classifier
    - All rules according to minsup and minconf
  - Syntax
    - $X \rightarrow y$
    - $X \rightarrow Y$

Why & How to Integrate

- Both classification rule mining and association rule mining are indispensable to practical applications.
- The integration is done by focusing on a special subset of association rules whose right-hand-side are restricted to the classification class attribute.
  - CARs: class association rules

CBA: Three Steps

- Discretize continuous attributes, if any
- Generate all class association rules (CARs)
- Build a classifier based on the generated CARs.

Our Objectives

- To generate the complete set of CARs that satisfy the user-specified minimum support (minsup) and minimum confidence (minconf) constraints.
- To build a classifier from the CARs.

Three Contributions

- It proposes a new way to build accurate classifiers.
- It makes association rule mining techniques applicable to classification tasks.
- It helps to solve a number of important problems with the existing classification systems, including:
    - *understandability* problem
    - discovery of *interesting* or *useful* rules
    - Disk v.s. Memory

Rule Generator: Basic Concepts

- Ruleitem

  <condset, y> :condset is a set of items,   y is a class label

  Each ruleitem represents a rule: condset->y

- condsupCount
  - The number of cases in D that contain condset
- rulesupCount
  - The number of cases in D that contain the condset and are labeled with class y
- Support=(rulesupCount/|D|)*100%
- Confidence=(rulesupCount/condsupCount)*100%

RG: Basic Concepts (Cont.)

- Frequent ruleitems
  - A ruleitem is frequent if its support is above *minsup*
- Accurate rule
  - A rule is accurate if its confidence is above *minconf*
- Possible rule
  - For all ruleitems that have the same condset, the ruleitem with the highest confidence is the possible rule of this set of ruleitems.
- The set of class association rules (CARs) consists of all the **possible** rules (PRs) that are both **frequent** and **accurate**.

RG: An Example

- A ruleitem:<{(A,1),(B,1)},(class,1)>
  - assume that
    - the support count of the condset *(condsupCount)* is 3,
    - the support of this ruleitem *(rulesupCount)* is 2, and
    - |D|=10
  - then (A,1),(B,1) -> (class,1)
    - supt=20% (rulesupCount/|D|)*100%
    - confd=66.7% (rulesupCount/condsupCount)*100%

RG: The Algorithm

```
F_1 = {large 1-ruleitems};
CAR_1 = genRules (F_1);
prCAR_1 = pruneRules (CAR_1); //count the item and class occurrences to
                                determine the frequent 1-ruleitems and prune it
for (k = 2; F_k-1 ≠ Ø; k++) do
    C_k = candidateGen (F_k-1); //generate the candidate ruleitems C_k
                                  using the frequent ruleitems F_k-1
  for each data case d ∈ D do //scan the database
    C_d = ruleSubset (C_k, d); //find all the ruleitems in C_k whose condsets
                                 are supported by d
    for each candidate c ∈ C_d do
      c.condsupCount++;
      if d.class = c.class then
        c.rulesupCount++; //update various support counts of the candidates in C_k
    end
  end
```

RG: The Algorithm(cont.)

$F_k = \{c \in C_k \mid$ c.rulesupCount $\geq minsup\}$;
//select those new frequent ruleitems to form F_k

CAR_k = genRules(F_k); //select the ruleitems both accurate and frequent

$prCAR_k$ = pruneRules(CAR_k);

end

$CARs = \cup_k CAR_k$;

$prCARs = \cup_k prCAR_k$;

Class Builder M1: Basic Concepts

- Given two rules r_i and r_j, define: $r_i \succ r_j$ if
  - The confidence of r_i is greater than that of r_j, or
  - Their confidences are the same, but the support of r_i is greater than that of r_j, or
  - Both the confidences and supports are the same, but r_i is generated earlier than r_j.
- Our classifier is of the following format:
  - $<r_1, r_2, \ldots, r_n$, default_class$>$,
    - where $r_i \in R$, $r_a \succ r_b$ if $b>a$

M1: Three Steps

The basic idea is to choose a set of high precedence rules in R to cover D.

- Sort the set of generated rules R
- Select rules for the classifier from R following the sorted sequence and put in C.
  - Each selected rule has to correctly classify at least one additional case.
  - Also select default class and compute errors.
- Discard those rules in C that don't improve the accuracy of the classifier.
  - Locate the rule with the lowest error rate and discard the rest rules in the sequence.

Example

A	B	C	D	E	Class
0	0	1	1	0	Y
0	0	0	1	1	N
0	1	1	1	0	Y
1	1	1	1	0	Y
0	1	0	0	1	N

RuleItemsets	Support
BY	40%
CY	60%
DY	60%
EN	40%
BCY	40%
BDY	40%
CDY	60%
BCDY	40%

Min_support = 40%   Min_conf = 50%

Example

Rules	Confidence	Support
B→Y	66.7%	40%
C→Y	100%	60%
D→Y	75%	60%
E→N	100%	40%
BC→Y	100%	40%
BD→Y	100%	40%
CD→Y	100%	60%
BCD→Y	100%	40%

Example

Rules	Confidence	Support
$C \rightarrow Y$	100%	60%
$CD \rightarrow Y$	100%	60%
$E \rightarrow N$	100%	40%
$BC \rightarrow Y$	100%	40%
$BD \rightarrow Y$	100%	40%
$BCD \rightarrow Y$	100%	40%
$D \rightarrow Y$	75%	60%
$B \rightarrow Y$	66.7%	40%

Example

A	B	C	D	E	Class
0	0	1	1	0	Y
0	0	0	1	1	N
0	1	1	1	0	Y
1	1	1	1	0	Y
0	1	0	0	1	N

Rules	Confidence	Support
C→Y	100%	60%
CD→Y	100%	60%
E→N	100%	40%
BC→Y	100%	40%
BD→Y	100%	40%
BCD→Y	100%	40%
D→Y	75%	60%
B→Y	66.7%	40%

Example

A	B	C	D	E	Class
0	0	1	1	0	Y
0	0	0	1	1	N
0	1	1	1	0	Y
1	1	1	1	0	Y
0	1	0	0	1	N

Rules	Confidence	Support
C→Y	100%	60%
CD→Y	100%	60%
E→N	100%	40%
BC→Y	100%	40%
BD→Y	100%	40%
BCD→Y	100%	40%
D→Y	75%	60%
B→Y	66.7%	40%

Default classification accuracy 60%

Example

A	B	C	D	E	Class
0	0	1	1	0	Y
0	0	0	1	1	N
0	1	1	1	0	Y
1	1	1	1	0	Y
0	1	0	0	1	N

Rules	Confidence	Support	
C→Y	100%	60%	✓
CD→Y	100%	60%	
E→N	100%	40%	
BC→Y	100%	40%	
BD→Y	100%	40%	
BCD→Y	100%	40%	
D→Y	75%	60%	
B→Y	66.7%	40%	

Example

A	B	C	D	E	Class
0	0	1	1	0	Y
0	0	0	1	1	N
0	1	1	1	0	Y
1	1	1	1	0	Y
0	1	0	0	1	N

Rules	Confidence	Support	
C→Y	100%	60%	✓
CD→Y	100%	60%	✗
E→N	100%	40%	
BC→Y	100%	40%	
BD→Y	100%	40%	
BCD→Y	100%	40%	
D→Y	75%	60%	
B→Y	66.7%	40%	

Example

A	B	C	D	E	Class
0	0	1	1	0	Y
0	0	0	1	1	N
0	1	1	1	0	Y
1	1	1	1	0	Y
0	1	0	0	1	N

Rules	Confidence	Support	
C→Y	100%	60%	✓
CD→Y	100%	60%	✗
E→N	100%	40%	✓
BC→Y	100%	40%	
BD→Y	100%	40%	
BCD→Y	100%	40%	
D→Y	75%	60%	
B→Y	66.7%	40%	

Example

A	B	C	D	E	Class
0	0	1	1	0	Y
0	0	0	1	1	N
0	1	1	1	0	Y
1	1	1	1	0	Y
0	1	0	0	1	N

Rules	Confidence	Support	
C→Y	100%	60%	✓
CD→Y	100%	60%	✗
E→N	100%	40%	✓
BC→Y	100%	40%	✗
BD→Y	100%	40%	✗
BCD→Y	100%	40%	✗
D→Y	75%	60%	✗
B→Y	66.7%	40%	✗

M1: Algorithm

- R = sort(R); //Step1:sort R according to the relation “$\succ$”
- **for** each rule $r \in R$ in sequence **do**
- *temp* = Ø;
- **for** each case $d \in D$ **do** //go through D to find those cases covered by each rule r
- **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; //r will be a potential rule because it can correctly classify at least one case d
- delete all the cases with the ids in *temp* from D;
- selecting a default class for the current C; //the majority class in the remaining data
- compute the total number of errors of C;
- **end**
- **end** // Step 2
- 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). //Step 3

M1: Two conditions it satisfies

- Each training case is covered by the rule with the highest precedence among the rules that can cover the case.
- Every rule in C correctly classifies at least one remaining training case when it is chosen.

M1: Conclusion

- The algorithm is simple, but inefficient especially when the database is not resident in the main memory. It needs too many passes over the database.
- The improved algorithm M2 takes slightly more than one pass.