CSCI 5408 Data Analytics: DM and DW Tech

(Mar 21, Week 11)

- Ass5 Due: Mar 28
  - Read Ass5-Tutorial slides
  - Ass5 Help Hours:
    - Wed, 1:00-2:30 PM, CS 134
    - Fri, 1:00-2:30 PM, CS 233
- Final Exam: Apr 20, 3:30-5:30 PM
- Write answers for review questions
- Reading: Lecture 16-17; Text: Ch6 of 3rd edition (or Ch5 of 2nd edition)

Recap: Association Rule Mining Approach (A two-phase process algorithm)

- Two major procedures:

1. **Find frequent itemsets**
   - Search all frequent itemsets from the DB
2. **Generate association rules**
   - Derive rules from each frequent itemset

Recap: **Search Space**

If a database has n unique items, and k denotes the combinations of k items (k ≤ n), the possible combinations:

$$\sum^{N}_{k=1} C^{K}, \quad \text{where } C^{K} = n! / ((n-k)! * k!)$$

E.g., when n = 4, the sum of

k=1,	$4!/((4-1)!*1!) = 4$	A,B,C,D
k=2,	$4!/((4-2)!*2!) = 6$	AB,AC,AD,BC,BD,CD
k=3,	$4!/((4-3)!*3!) = 4$	ABC,ABD,ACD,BCD
k=4,	$4!/((4-4)!*4!) = 1$	ABCD

$\sum^{4}_{k=1} C^{4}$, i.e. the number of itemsets = $\mathbf{2^4 - 1} =$ $\boxed{15.}$

When n = 100, the number of itemsets = ?

$2^{100} = 1.27 \times 10^{30}$

How big is it?

Recap: How many itemsets **actually occurred in D (vs. the Lattice)**?

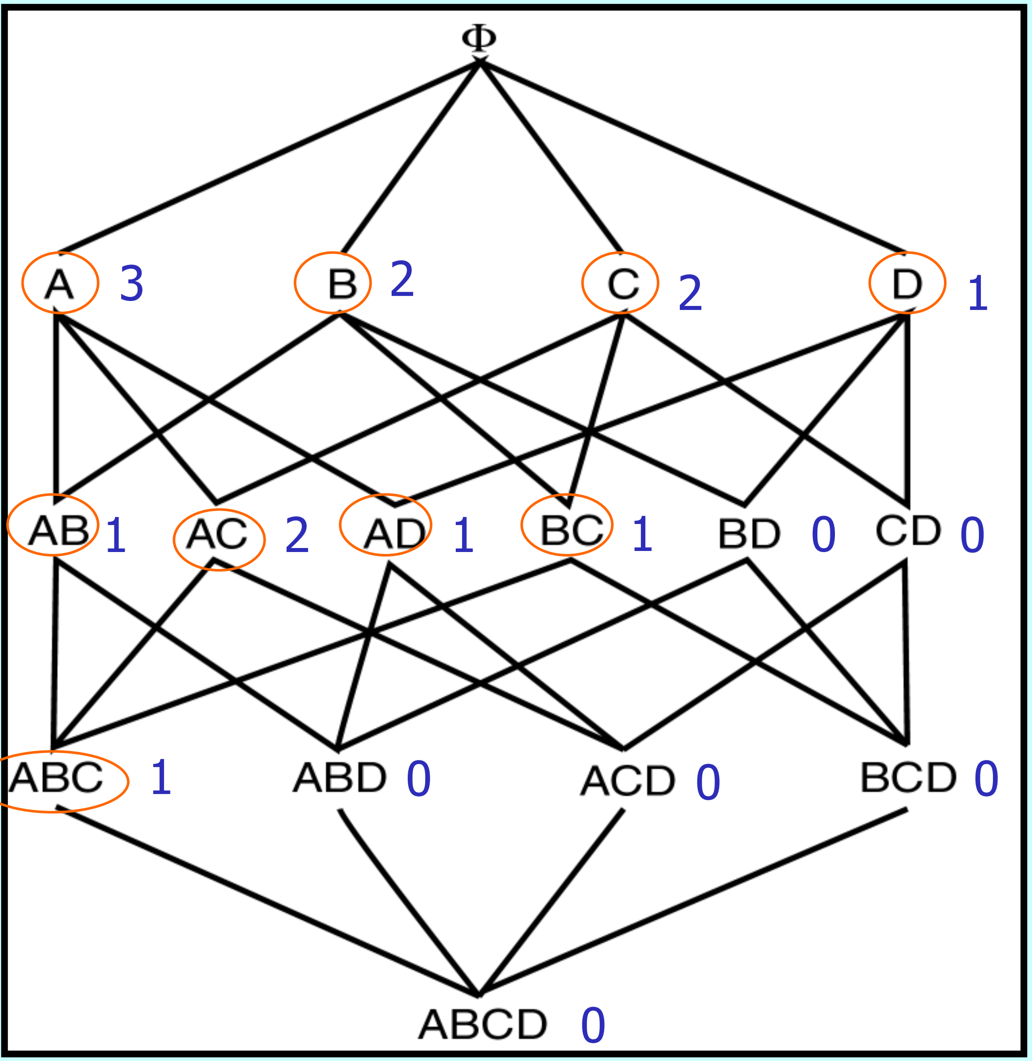
D:

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B

What is the ground truth?
* The unique itemsets contained in D are only a subset of the lattice (with various counts).

Do we need to search whole space, and why?

Lattice of itemsets for I = {A, B, C, D}



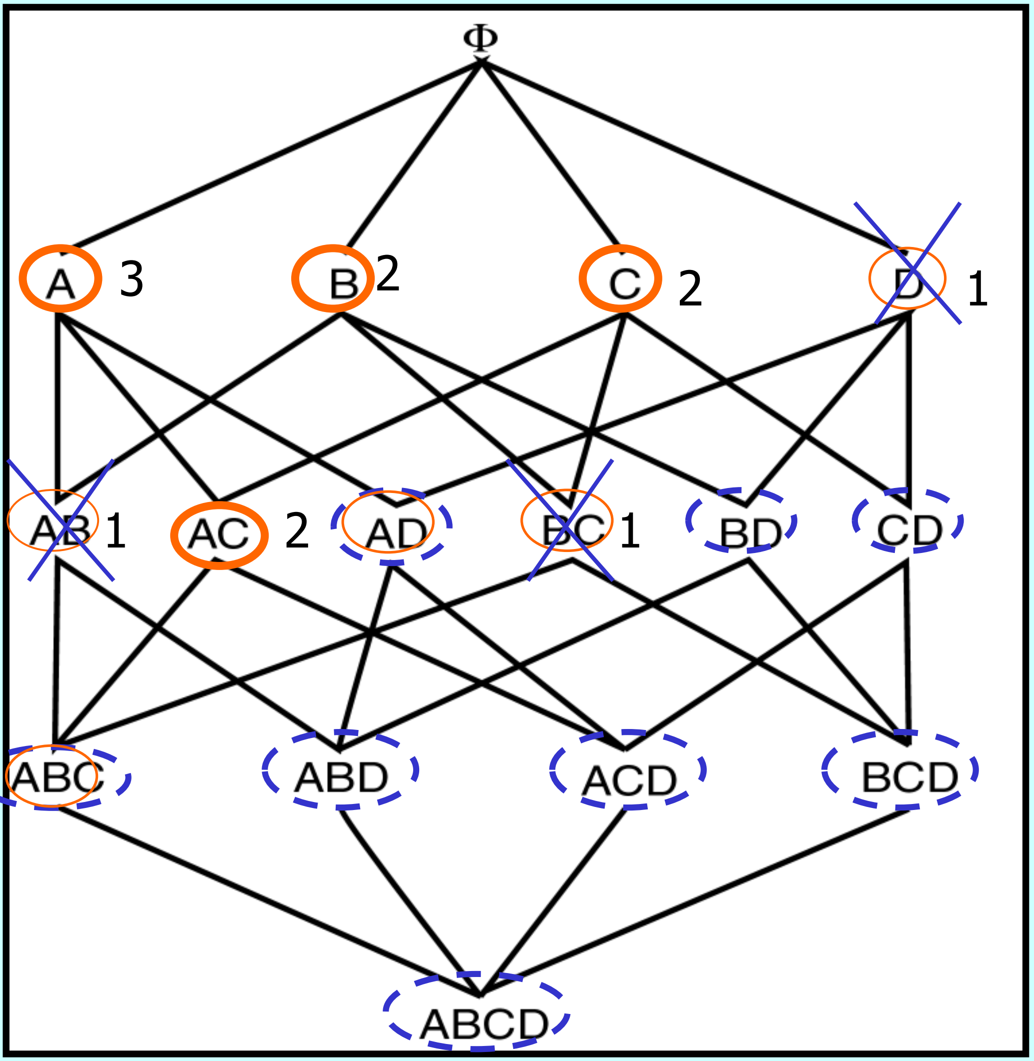
Recap: **Search Pruning**

Sup_rate = 30%

D:

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B

Lattice of itemsets for I = {A, B, C, D}



Heuristic: <u>If a itemset is not frequent, all of its supersets are not frequent,</u> i.e. **no need to search them (i.e. generate them).**

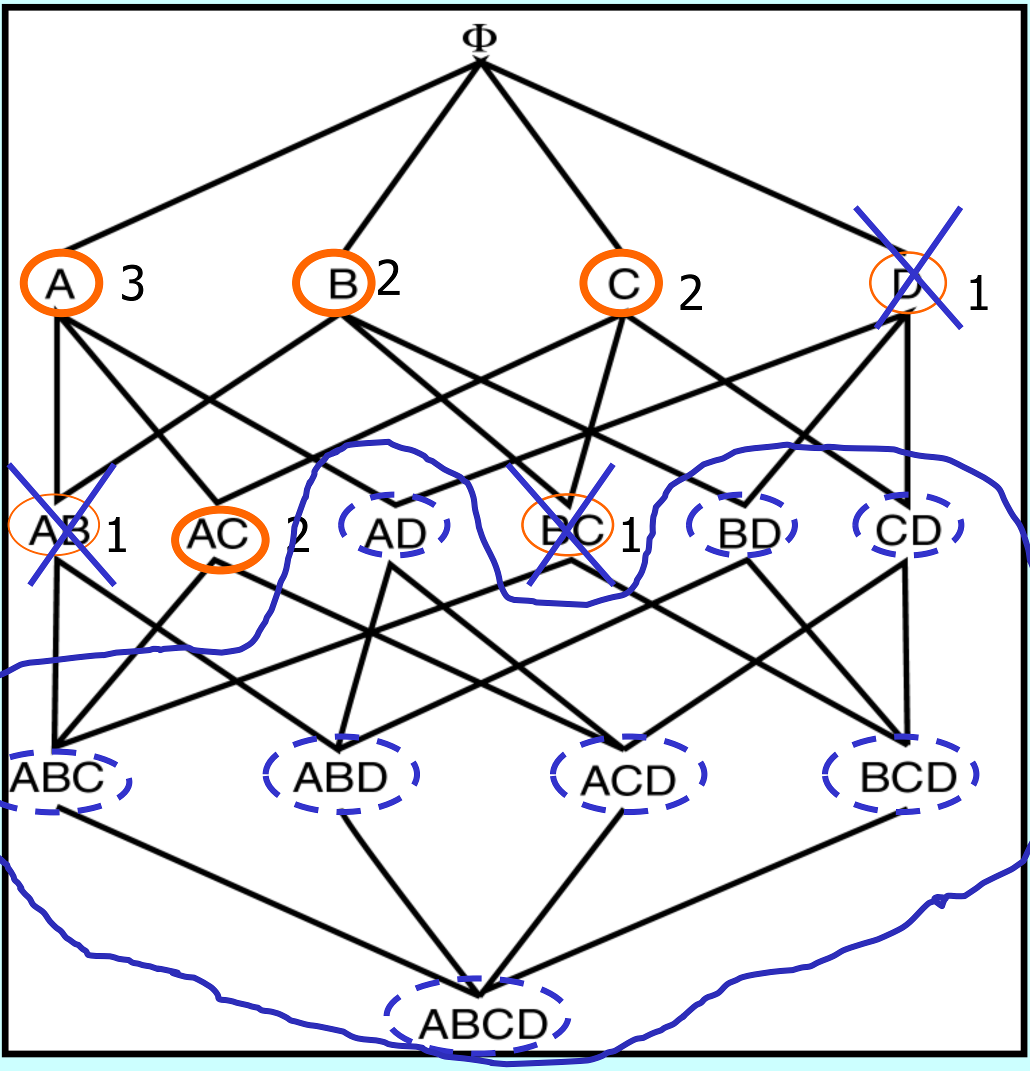
If an itemset is not frequent, none of its supersets is frequent.

D: Sup_rate = 30%

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B

Heuristic: If a itemset is not frequent, all of its supersets are not frequent, i.e. **no need to search them** (i.e. no need to generate them).

Lattice of itemsets for I = {A,B,C,D}



Pruned search space

A priori knowledge: Apriori Property

- Frequent Itemset Property:

  Any subset of a frequent itemset is frequent.

- Contra-positive:

  If an itemset is not frequent, none of its supersets is frequent.

How can this knowledge be applied in designing a more efficient algorithm?

Tracing Apriori Algorithm: given a TDB & Sup≥50%

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1st scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

Apriori Algorithm: A Candidate Generation-and-Test Approach

- **Key steps:**
  - Initially, scan DB once to get frequent 1-itemsets
  - **Generate** (k+1) **candidate** itemsets from k frequent itemsets
  - **Test the candidates (pruning)**
  - Terminate when no frequent or candidate set can be generated

Apriori Algorithm (Pseudo-code)

Input: Dataset D, min_sup
Output: L   // frequent itemsets in D

```
$C_k$ : Candidate k-itemsets;
$L_k$ : Frequent k-itemsets;

$L_1$ = frequent_1-itemsets(D, min_sup);
for ($k$ = 2; $L_{k-1}$ !=$\emptyset$; $k$++) {               // search in order
    $C_k$ = apripri_gen($L_{k-1}$);                    // candidates generation
    for each transaction $t \in$ D {    // scan D for counts (candidate
                                        testing)
        $C_t$ = subset($C_k$ , t);                   // candidate subsets
        for each candidate c $\in C_t$
        c.count++;                  }
    $L_k$ = {c$\in C_k$ | $c.count \geq min\_sup$}
return $\cup_k L_k$ ;
```

- Identify each part of candidate “Generation” and “Testing”.
- In the algorithm, where the Apriori knowledge is applied?
- How the knowledge is applied in each identified place?

Input: Database D, min_supp

Output: L, frequent itemsets in D

```
1.   $C_k$ : Candidate itemsets of size k;
2.   $L_k$ : frequent itemsets of size k;
3.   $L_1$ = frequent_1-itemsets(D, min_sup);
4.   for (k = 2; $L_{k-1}$ !=$\varnothing$; k++) {          // search in order
5.       $C_k$ = apripri_gen($L_{k-1}$);            // candidate generation
6.       for each transaction t $\in$ D {  // candidate testing
7.           $C_t$ = subset($C_k$, t);              // candidate subsets
8.           for each candidate c $\in$ $C_t$
9.           c.count++;                 }
10.      $L_k$ = {c$\in$ $C_k$ | c.count $\geq$ min_sup}
11.  return $\cup_k$ $L_k$ ;
```

- Identify each part of candidate "Generation" and "Testing".
- In the algorithm, where the Apriori knowledge is applied?
- How the knowledge is applied in each identified place?

Input: Database D, min_supp

Output: L, frequent itemsets in D

```
   1.  $C_k$ : Candidate itemsets of size k;
   2.  $L_k$ : frequent itemsets of size k;
⇨  3.  $L_1$ = frequent_1-itemsets(D, min_sup);
   4.  for (k = 2; $L_{k-1}$ !=$\emptyset$; k++) {          // search in order
⇨  5.      $C_k$ = apripri_gen($L_{k-1}$);           // **candidate generation**
   6.      for each transaction t $\in$ D {  // **candidate testing**
   7.         $C_t$ = subset($C_k$, t);           // candidate subsets
   8.         for each candidate c $\in$ $C_t$
   9.         c.count++;               }
⇨  10.     $L_k$  = {c$\in$ $C_k$  | c.count $\geq$ min_sup}
   11. return $\cup_k$ $L_k$ ;
```

Candidate Generation: $C_k = L_{k-1} \bowtie L_{k-1}$

```
apriori_gen($L_{k-1}$, min_sup)
    for each itemset $l_1$ in $L_{k-1}$ {
        for each itemset $l_2$ in $L_{k-1}$
            if ( ($l_1[1] = l_2[1]$) ∧ ($l_1[2] = l_2[2]$) ∧ … ∧ ($l_1[k-1] = l_2[k-1]$) ∧
                ($l_1[k] < l_2[k]$) ) {
                c = $l_1 \bowtie l_2$;                // join for generating candidates
                if has_infrequent_subset($c$, $L_{k-1}$) {
                    delete $c$;                       // **prune unfruitful candidates**
                    else add $c$ to $C_k$; }
            }
        }
    }
    return $C_k$;
```

Candidate Generation: **apriori_gen(L_{k-1})**

- Join Operation: $C_k = L_{k-1} \bowtie L_{k-1}$
- E.g., $k = 2$ and $L_1 = \{\{1\},\{2\},\{3\},\{4\}\}$

$$\boxed{\mathbf{C_2 = L_1 \bowtie L_1 = ?}}$$

1. $L_1 \times L_1 = \{\{1\},\{2\},\{3\},\{4\}\} \times \{\{1\},\{2\},\{3\},\{4\}\}$

$= ?$

2. Remove redundant and irrelevant itemsets.

Candidate Generation: apriori_gen(L_{k-1})

- The Join Operation: $C_k = L_{k-1} \bowtie L_{k-1}$
- E.g., $k = 4$ and L_3 = {{1 2 3},{1 2 4},{1 3 4},{1 3 5},{2 3 4}}

$$\boxed{C_4 = L_3 \bowtie L_3}$$

= {{$\underline{1\ 2\ 3}$}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}

$\bowtie$

{{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}} =

{1 2 3} $\bowtie$ {1 2 3} = {123123} = {123} X // Irrelevant
{1 2 3} $\bowtie$ {1 2 4} = {123124} = {1234} √
{1 2 3} $\bowtie$ {1 3 4} = {123134} = {1234} X // Redundant

…

Test condition for: $C_k = L_{k-1} \bowtie L_{k-1}$

```
apriori_gen($L_{k-1}$, min_supp)
    for each itemset $l_1 \in L_{k-1}$ {
        for each itemset $l_2 \in L_{k-1}$
            if ( ($l_1[1] = l_2[1]$) ∧ ($l_1[2] = l_2[2]$) ∧ … ∧ ($l_1[k-1] = l_2[k-1]$) ∧
                ($l_1[k] < l_2[k]$) ) {
                c = $l_1 \bowtie l_2$;        // join for generating candidates
    …
```

$l_1[1]$

E.g., $C_k = L_{k-1} \bowtie L_{k-1}$ = {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}
{{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}

$l_2[1]$

= ?

$E.g., C_4 = L_3 \bowtie L_3$

$= \{\{1\ 2\ 3\}, \{1\ 2\ 4\}, \{1\ 3\ 4\}, \{1\ 3\ 5\}, \{2\ 3\ 4\}\} \bowtie$
$\{\{1\ 2\ 3\}, \{1\ 2\ 4\}, \{1\ 3\ 4\}, \{1\ 3\ 5\}, \{2\ 3\ 4\}\}$

$\{1\ 2\ 3\} \bowtie L_3 = \{1\ 2\ 3\ 1\ 2\ 4\} = \{1\ 2\ 3\ 4\}$
$\{1\ 2\ 4\} \bowtie L_3 = \{empty\}$ *// No new k-itemset*
$\{1\ 3\ 4\} \bowtie L_3 = \{1\ 3\ 4\ 1\ 3\ 5\} = \{1\ 3\ 4\ 5\}$
$\{1\ 3\ 5\} \bowtie L_3 = \{empty\}$ *// No new k-itemset*
$\{2\ 3\ 4\} \bowtie L_3 = \{empty\}$ *// No new k-itemset*

After the join step, $C_4 = \{\{1\ 2\ 3\ 4\}, \{1\ 3\ 4\ 5\}\}$.

```
for each itemset $l_1 \in L_{k-1}$ {
        for each itemset $l_2 \in L_{k-1}$
            if ( ($l_1[1] = l_2[1]$) ∧ ($l_1[2] = l_2[2]$) ∧ … ∧ ($l_1[k-1] = l_2[k-1]$) ∧
                ($l_1[k] < l_2[k]$) ) { // ensure no duplications are generated
              c = $l_1 \bowtie l_2$;     // join for generating candidates
```

Apriori Algorithm (Pseudo-code)

Input: Database *D*, *min_sup*
Output: *L*, frequent itemsets in *D*

```
$C_k$ : list of candidate itemsets of size k;
$L_k$ : list of frequent itemsets of size k;

$L_1$ = frequent_1-itemsets(D, min_sup);
for ($k$ = 2; $L_{k-1}$ !=$\varnothing$; $k$++) {          // search in order
    $C_k$ = apriori_gen($L_{k-1}$);                      // candidates generation
    for each transaction $t \in$ D {                   // scan D for counts (candidate
                                                          testing)
        $C_t$ = subset($C_k$, t);                        // subset of candidates from
                                                          a single transaction t
        for each candidate c $\in C_t$
            c.count++;                             }
    $L_k$ = {c$\in C_k$ | $c.count \geq min\_sup$}
return $L = \cup_k L_k$;
```

Candidate Generation: $C_k = L_{k-1} \bowtie L_{k-1}$

apriori_gen(L_{k-1}, min_sup)

```
    for each itemset $l_1$ in $L_{k-1}$ {
        for each itemset $l_2$ in $L_{k-1}$
            if ( ($l_1[1] = l_2[1]$) /\ ($l_1[2] = l_2[2]$) /\ … /\ ($l_1[k-1] = l_2[k-1]$) /\
                ($l_1[k] < l_2[k]$) ) {
                c = $l_1 \bowtie l_2$;                    // join for generating candidates
                if has_infrequent_subset($c$, $L_{k-1}$) {
                  delete $c$;                          // prune unfruitful candidates
                  else add $c$ to $C_k$; }
            }
        }
    }
    return $C_k$;
```

E.g., $C_4 = L_3 \bowtie L_3$ = {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}

{{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}

= {{1 2 3 4}, ~~{1 3 4 5}~~ }

= {{1 2 3 4}

C_4 = {1 2 3 4} . Why {1 3 4 5} was eliminated?

Check infrequent subset for pruning

```
has_infrequent_subset($c$, $L_{k-1}$)        // $c$ : candidate k-itemset
                                             // apply aprior knowledge
    for each (k-1)-subset $s$ of $c$
        if ($s \notin L_{k-1}$ )
            return TRUE;
    return FALSE;
```

E.g., $C_4 = L_3 \bowtie L_3$ = {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}} {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}

= **{{1 2 3 4}, {1 3 4 5}}**

{1 3 4 5} was eliminated because its subsets {1 4 5} and {3 4 5} are not in L_3.

apripri_gen(L_{k-1}, min_supp)

L_{k-1} ---> apriori_gen( ) ---> C_k :

- JOIN process for generating C_k
- Apply the Apriori knowledge to filter the candidates with sup < min_sup but without counting from DB: *has_infrequent_subset(c, L_{k-1})*

How to Generate Rules from L

1. Generate candidate rules from each $k \geq 2$ itemset l_k in L.
2. Test for strong rules by filtering the rules with conf < min_conf.

Generate Rules from L

1. Generate candidate rules from each k ≥ 2 itemset l_k in L.
2. Test for strong rules by filtering the rules with conf < min_conf.

- E.g., Given L and min_sup = 50%, min_conf= 70%:

  The input: L= {L_1, L_2, L_3}:
  L_1 = { {A}, {B}, {C}, {E} }
  L_2 = { {A,C}, {B,C}, {B,E}, {C,E} }
  L_3 = { {B,C,E} }

Generate Candidate Rules from L

Input: L= {L1, L2, L3}

L1 = {{A},{B},{C},{E}}
L2 ={{A,C},{B,C},{B,E,},{C,E}}
L3 = {{B,C,E}}

- Generate candidate rules from each k ≥ 2 itemset l_k in L
  - For each itemset l_k generate <u>all non-empty subsets</u> of l_k, e.g.
    - L2 = {{A,C},{B,C},{B,E,},{C,E}}
      Subsets of {A,C}: {{A},{C}}
      …
    - L3 = {{B,C,E}}
      Subsets of {B,C,E}: {{B},{C},{E},{B,C},{B,E},{C,E}}
  - For every non-empty subset s of itemset l_k generate $\boxed{s \Rightarrow (l_k - s)}$
    - E.g., For l_3 = {B,C,E}, the candidate rules:
      {B} ⇒ {C,E},
      {C} ⇒ {B,E},
      {E} ⇒ {B,C},
      {B,C} ⇒ {E},
      {B,E} ⇒ {C},
      {C,E} ⇒ {B}

Generate Strong Rules from Candidates

- Test for **strong rules** with conf ≥ min_conf
  - Confidence measure: conf(X⇒Y) = $\boxed{P(Y|X)}$
    = $\boxed{sup(X,Y) / sup(X)}$

* No new count is needed.

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

L_1 = { {A}, {B}, {C}, {E} }
L_2 = { {A,C}, {B,C}, {B,E,}, {C,E} }
L_3 = { {B,C,E} }

E.g., For l_3 = {B,C,E}, the candidate rules:
{B}⇒{C, E}, {C} ⇒ {B, E}, {E} ⇒{B, C},
{B, C}⇒{E}, {B, E}⇒{C}, {C, E}⇒{B}

Rules with conf=66.6%:
{B}⇒{C, E}, (conf = 2/3, i.e.
$\boxed{\text{sup\{B,C,E\}} = 50\% / \text{sup\{B\}} = 75\%}$)

{C}⇒{B, E}, {E}⇒{B, C}, {B, E}⇒ C}

Rules with conf=100%:
{B, C}⇒{E}, {C, E}⇒{B}, {B}⇒{E}, …

Generate Rules from *L*

Algorithm: rule_gen (L, min_conf)

// Generate strong association rules, i.e., conf(rule) > min_conf, from the frequent itemsets L.

```
for each itemset $l_k$ in $L$ with $k \geq 2$ {
    S = subsets_gen ($l_k$);        // Generate a list S containing
                                       all subsets of $l_k$
    for each subset s in S {
        conf = sup($l_k$) / sup(s);
        if (conf $\geq$ min_conf ) then
            print the rule: "s $\Rightarrow$ ($l_k$ - s)", with sup & conf;
    }
}
```

Any Concern on Rule Replication?

Given the frequent itemset {A,B,C}, can we generate {A,B} $\Rightarrow$ {A,C} ?

Given the frequent itemset {A,B,C}, can we generate {A,B} => {A,C} ?

Wrong! It conflicts with the condition:
for X $\Rightarrow$ Y, X INTERSECT Y = Ø.

The rule {A,B} $\Rightarrow$ {C} for the itemset {A,B,C}
shouldn't be replicated by

{A,B} $\Rightarrow$ {A,C}
{A,B} $\Rightarrow$ {B,C}
{A,B} $\Rightarrow$ {A,B,C}

Given the frequent itemset {A,B,C}, can we generate {A,B} $\Rightarrow$ {A,C} ?

Wrong! It conflicts with the condition:
for X $\Rightarrow$ Y, X INTERSECT Y = Ø.

"s $\Rightarrow$ (l - s)" guarantees that X $\cap$ Y = Ø.

Any room for improving efficiency?

- E.g., If we know "$\{A,C\} \Rightarrow \{B\}$" < min_conf,
  do we need to search for $\{A\} \Rightarrow \{B,C\}$ and
  $\{C\} \Rightarrow \{A,B\}$, Why?

Any room for improving efficiency?

- E.g., If we know "$\{A,C\} \Rightarrow \{B\}$" < min_conf,
  do we need to search for $\{A\} \Rightarrow \{B,C\}$ and
  $\{C\} \Rightarrow \{A,B\}$, why?

Observation: {A, B, C}

a) $\{A,C\} \Rightarrow \{B\}$, conf = sup{A,B,C} / sup{A,C}

b) $\{A\} \Rightarrow \{B,C\}$, conf = sup{A,B,C} / sup{A}

c) $\{C\} \Rightarrow \{A,B\}$, conf = sup{A,B,C} / sup{C}

Any room for improving efficiency?

- E.g., If we know "{A,C} $\Rightarrow$ {B}" < min_conf,
  do we need to search for {A} $\Rightarrow$ {B,C} and
  {C} $\Rightarrow$ {A,B}, why?

Observation: {A, B, C}

a) {A,C} $\Rightarrow$ {B}, conf = sup{A,B,C} / sup{A,C}

b) {A} $\Rightarrow$ {B,C}, conf = sup{A,B,C} / sup{A}

c) {C} $\Rightarrow$ {A,B}, conf = sup{A,B,C} / sup{C}

Observation: What can be generalized?

a) $\{A,C\} \Rightarrow \{B\}$, conf = sup{A,B,C} / sup{A,C}

b) $\{A\} \Rightarrow \{B,C\}$, conf = sup{A,B,C} / sup{A}

c) $\{C\} \Rightarrow \{A,B\}$, conf = sup{A,B,C} / sup{C}

We know: sup{A,C} $\le$ sup(A) and

sup{A,C} $\le$ sup(C).

Where is the room for improvement?

Where is the room for improvement?

- If we know $\{A,C\} \Rightarrow \{B\}$ < min_conf, do we need to search $\{A\}\Rightarrow\{B,C\}$ and $\{C\}\Rightarrow\{A,B\}$, why?

  a) $\{A,C\} \Rightarrow \{B\}$, conf = sup{A,B,C} / sup{A,C}
  b) $\{A\} \Rightarrow \{B,C\}$, conf = sup{A,B,C} / sup{A}
  c) $\{C\} \Rightarrow \{A,B\}$, conf = sup{A,B,C} / sup{C}

- If conf(rule a)) < min_conf, do we need to search rules b) and c)?
  - Because sup{A} $\geq$ sup{A,C} and sup{C} $\geq$ sup{A,C}, so that conf(b) $\leq$ conf(a), conf(c) $\leq$ conf(a)

    without the need of searching rules b) and c).

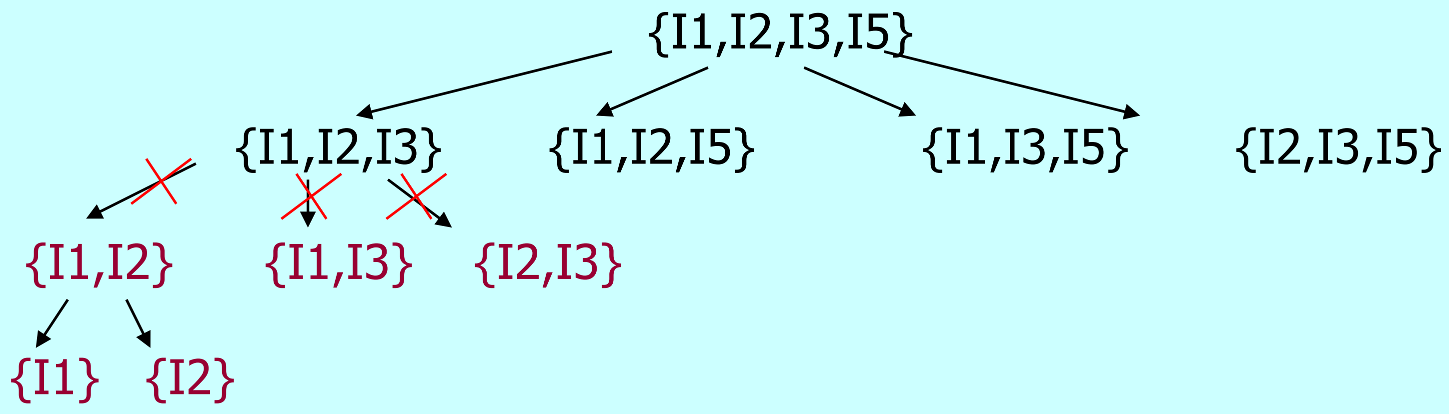
Formalize Pruning Rule

- **Improve the search for generating sub-itemsets**
- **The pruning rule ('efficiency trick rule'):**
  - If s is a subset of l, we know sup(s) ≥ sup(l)
    - E.g., l = {I2,I3,I4}, we have sup{I2,I3} ≥ sup{I2,I3,I4}.
  - **If s' is a subset of s, then conf(s⇒(l-s)) ≥ conf(s'⇒(l-s'))**
    - E.g., For {I2,I4}⇒{I3}, s={I2,I4}, s'= {I2} or s'= {I4}, then we have candidate rules: {I2}⇒{I3,I4}, {I4}⇒{I2,I3},

If conf({I2,I4}⇒{I3}) < min_conf, then neither of the rules, {I2}⇒{I3,I4}, {I4}⇒{I2,I3}, can pass the min_conf.

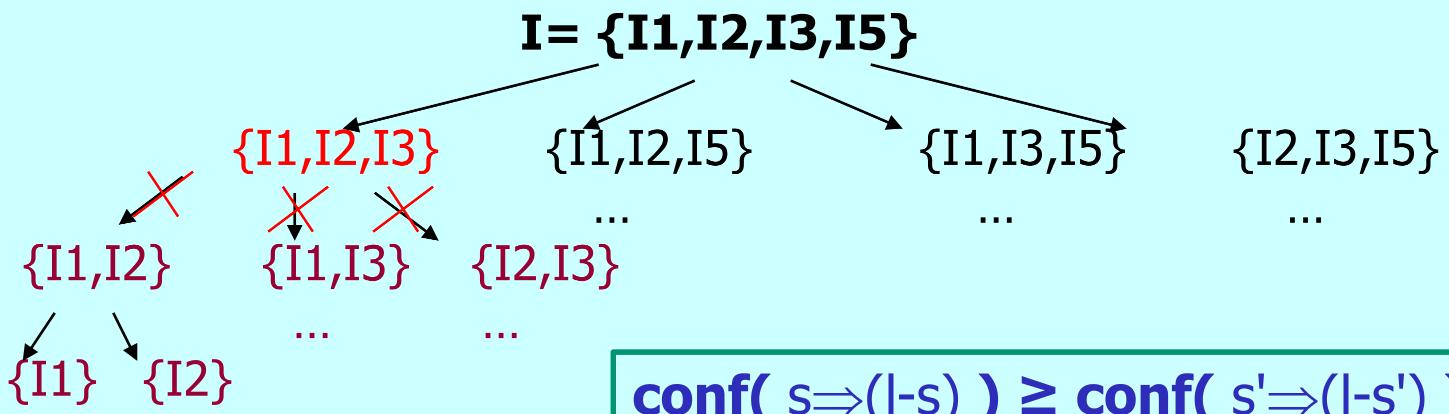
Prune search tree while generate rules

- E.g. If "{I1,I2,I3}⇒{I5}" < min_conf then the search stops to generate the rules from {I1,I2,I3}, e.g. "{I1,I2}⇒{I3,I5}'', "{I1}⇒ {I2,I3,I5}'', etc. can be pruned.
- Depth-first search for generating subsets of the frequent itemset l.
  E.g., for {I1,I2,I3,I5}, generate {I1,I2,I3}, then generate {I1,I2}, ...



Pruning while generating rules

(* Ass3 Bonus: +20% for proper implementation, and describing it in README)



conf(s⇒(l-s) **) ≥ conf(** s'⇒(l-s') **),**
where s' is a subset of s.

E.g., According the pruning rule:

If "{I1,I2,I3} => {I5}" < min_conf then stop to search for the following rules:

{I1,I2} => {I3,I5}, {I1,I3} => {I2,I5}, {I2,I3} => {I1,I5},

{I1} => {I2,I3,I5}, {I2} => {I1,I3,I5}, and {I3} => {I1,I3,I5}

Association Rule Mining for Relational Data

- **Given a relational dataset, find all associations of the form:**
  - Transaction DB: $X \Rightarrow Y$
  - Relational DB: {attr-value pairs} $\Rightarrow$ {attr-value pairs}
- **Examples:**
  - {wind=weak,outlook=sunny,humidity=high} $\Rightarrow$ {play=no}
  - {wind=weak,play=no} $\Rightarrow$ {outlook=sunny, humidity=high}
  - {age={30, …,39},income={42K,…,48K}} $\Rightarrow$ {buys=DVD player}
- **Main issues:**
  - How to convert relational data into items
  - How to convert quantitative attributes to categorical attributes for forming items

Transaction Data vs. Relational Data

- Transaction DB:

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

$\xrightarrow{\text{Relational table}}$

- Each attribute is a binary variable, and
- only value "1" should be considered as "item".

Tid	A	B	C	D	E
10	1	0	1	1	0
20	0	1	1	0	1
30	1	1	1	0	1
40	0	1	0	0	1

- Relational DB:

Id	Age	Married	NumCars
100	23	No	1
200	25	Yes	1
300	25	No	0
400	34	Yes	2
500	39	Yes	2

- An attribute can have a domain of binary, or multiple/continued values.
- All domain values should be considered as "items".

E.g., Association rule mining with min_sup = 40% and min_conf = 60%

Id	Age	Married	NumCars
100	**23**	**No**	**1**
200	**25**	**Yes**	**1**
300	**25**	**No**	**0**
400	**34**	**Yes**	**2**
500	**39**	**Yes**	**2**

C_1 = { Age=23 [1], Age=25 [2], Age=34 [1],
Age=39 [1], Married=No [2],
Married=Yes [3], NumCars=0 [1],
NumCars=1 [2], NumCars=2 [2] }

L_1 = { Age=25 [2], Married=No [2], Married=Yes [3],
NumCars=1 [2], NumCars=2 [2] }

L_2 = { ? }

* An itemset can not have two items share a same attribute name.

Given supp= 40% and conf = 60%:

Rules	Supp	Conf
{ Married=Yes } $\Rightarrow$ { NumCars=2 }	40%	66.6%
{ NumCars=2 } $\Rightarrow$ { Married=Yes }	40%	100%

E.g., Association rule mining with min_sup = 40% and min_conf = 60%

Id	Age	Married	NumCars
100	23	No	1
200	25	Yes	1
300	25	No	0
400	34	Yes	2
500	39	Yes	2

C_1 = { Age=23 [1], Age=25 [2], Age=34 [1], Age=39 [1], Married=No [2], Married=Yes [3], NumCars=0 [1], NumCars=1 [2], NumCars=2 [2] }

L_1 = { Age=25 [2], Married=No [2], Married=Yes [3], NumCars=1 [2], NumCars=2 [2] }

L_2 = { ? }

* An itemset can not have two items share a same attribute name.

Given supp= 40% and conf = 60%:

Rules	Supp	Conf
{ Married=Yes } ⇒ { NumCars=2 }	40%	66.6%
{ NumCars=2 } ⇒ { Married=Yes }	40%	100%
* { Age={34,39}, Married=Yes } ⇒ { NumCars=2 }	40%	100%

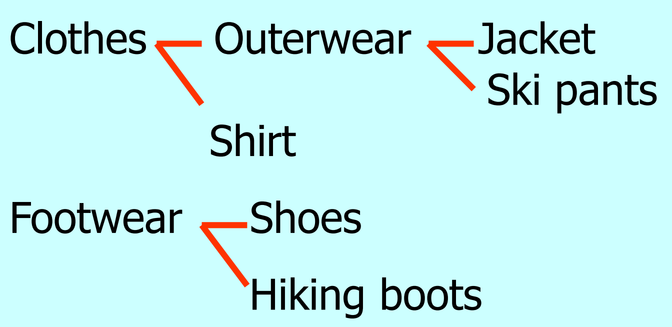
* Grouping: "Age" was identified for grouping: { Age={?}, Married=Yes} ⇒ { NumCars=2 }

How to mine multilevel association rules?

E.g., Transaction DB with category concepts:

ID	Items
100	Shirt
200	Jacket, Hiking boots
300	Ski pants, Hiking boots
400	Shoes
500	Shoes
600	Jacket

Concept Hierarchies:



ID	Items
100	Shirt
200	Jacket, Hiking boots
300	Ski pants, Hiking boots
400	Shoes
500	Shoes
600	Jacket

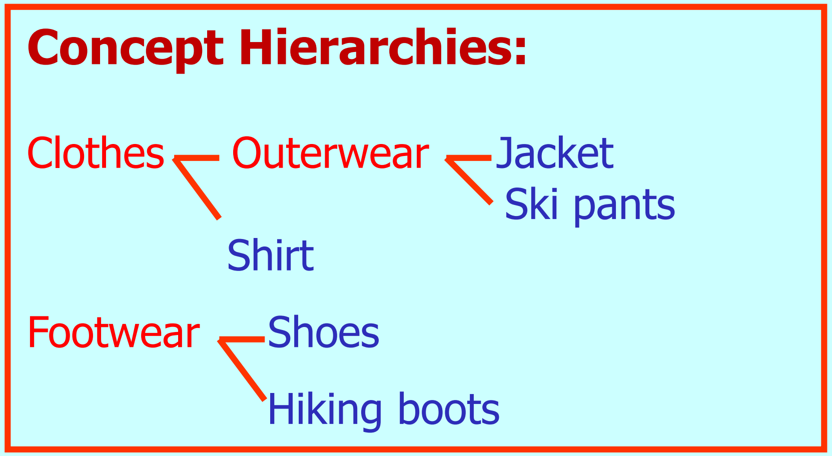
1-Itemset counts:

Itemset	Sup
{Shirt}	1
{Jacket}	2
{Hiking boots}	2
{Ski pants}	1
{Shoes}	2

Frequent itemsets: (min_sup=30%)

Itemset	Sup
{Jacket}	2
{Shoes}	2
{Hiking boots}	2

ID	Items
100	Shirt
200	Jacket, Hiking boots
300	Ski pants, Hiking boots
400	Shoes
500	Shoes
600	Jacket



Count 1-itemsets:

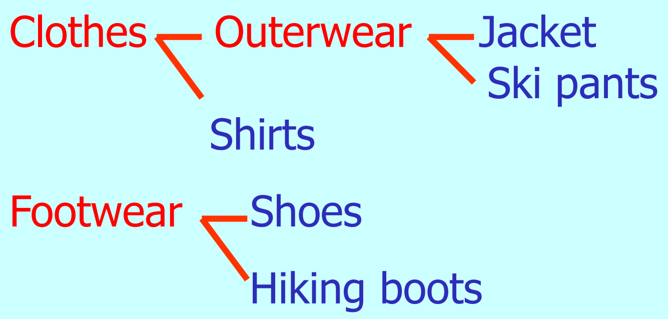
1-Itemset	Sup
{Shirt}	1
{Jacket}	2
{Hiking boots}	2
{Ski pants}	1
{Shoes}	2
{Clothes}	4
{Outerwear}	3
{Footwear}	4

Frequent 1-itemsets: (min_sup=30%)

Itemset	Sup
{Jacket}	2
{Shoes}	2
{Hiking boots}	2
{Clothes}	4
{Outerwear}	3
{Footwear}	4

ID	Items
100	Shirt
200	Jacket, Hiking boots
300	Ski pants, Hiking boots
400	Shoes
500	Shoes
600	Jacket

Concept Hierarchies:

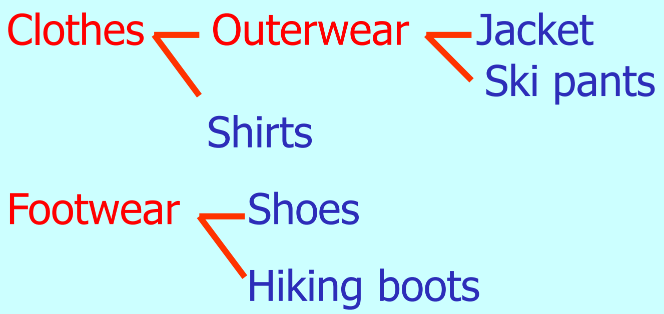


Frequent itemsets: (min_sup=30%)

Itemset	Sup
{Jacket}	2
{Outerwear}	3
{Clothes}	4
{Shoes}	2
{Hiking boots}	2
{Footwear}	4
{Outerwear, Hiking boots}	2
{Clothes, Hiking boots}	2
{Outerwear, Footwear}	2
{Clothes, Footwear}	2

ID	Items
100	Shirt
200	Jacket, Hiking boots
300	Ski pants, Hiking boots
400	Shoes
500	Shoes
600	Jacket

Concept Hierarchies:



Frequent itemsets: (min_sup=30%)

Itemset	Sup
{Jacket}	2
{Outerwear}	3
{Clothes}	4
{Shoes}	2
{Hiking boots}	2
{Footwear}	4
{Outerwear, Hiking boots}	2
{Clothes, Hiking boots}	2
{Outerwear, Footwear}	2
{Clothes, Footwear}	2

Set min_sup = 30%, min_conf = 50%:

Rule	Sup	Conf
{Clothes} $\Rightarrow$ {Hiking boots}	33%	50%
{Outerwear} $\Rightarrow$ {Footwear}	33%	66.6%
{Hiking boots} $\Rightarrow$ {Outerwear}	33%	100%
{Hiking boots} $\Rightarrow$ {Clothes}	33%	100%

Constraint-based Association Mining

- Do we need to find all the association patterns in a database?
  - Some time unrealistic & unnecessary: The patterns could be too many but not focused
- Data mining program should allow the user to specify a set of mining conditions (i.e. various constraints):
  - User directs what to be mined using a data mining interface (such as a query language, or a graphical user interface etc.)

- Constraint-based association pattern mining
  - **User flexibility:** Provides constraints on what to be mined.
  - **Efficient mining:** Apply prior application information: constraints for obtaining more relevant association patterns quickly.
  - **Interface:** Design user friendly interface.
  - E.g., Pre-define one side of the rules to be mined:
    1. {shirt} $\Rightarrow$ {?} (sup, conf)
    2. {?} $\Rightarrow$ {age={20-40}, income $\geq$ \$50,000} (sup, conf)
    3. {?} $\Rightarrow$ {PlayTennis=yes} (sup, conf)

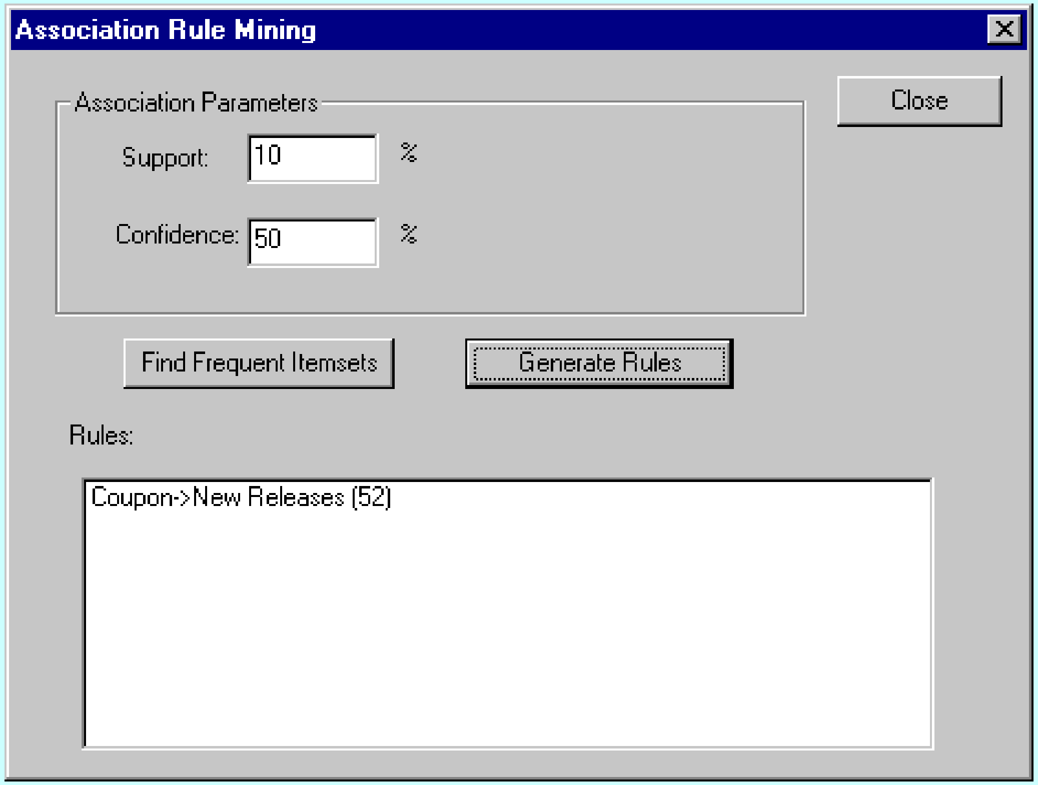
Application Example

- **Association rule mining for video store data analysis:** (Doc/Thesis-Project/HBthesisPothier99.pdf)
  - The goal is to find predictable association rules to analyze which items that occur within a transaction imply that other items will occur within that same transaction.
  - This information has obvious value to the business owner since they can encourage customers to buy additional products using the cross-selling information discovered from the data.

- The potential items that exist within the application database are listed:

Item Names		
Calendar	Used Movie	Small Slush
Nintendo	CD	New Releases
Birthday Movie	Poster	Regular Movies
Free Movie	Photocopies	Childrens
Coupon	Popcorn	Free Rentals
Sega Games	DVD	DVD Player
Nintendo 64	SNES Machine	SNES Games
Adult	N64 Machine	
Gift Certificate	Large Slush	

- *“The process of mining association rules is extremely dependent on the support and confidence values that are chosen for mining. If we use a small test set with support = 10% and confidence = 50%, we get the only one association rule:”*



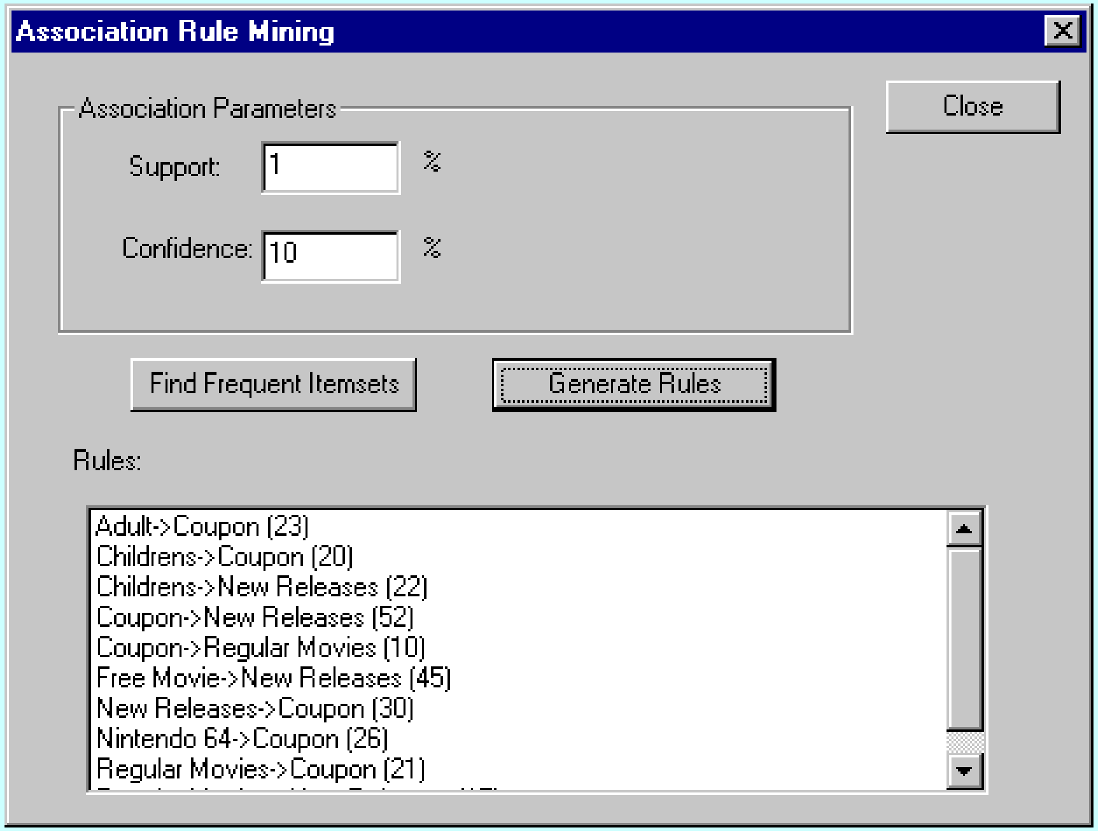
Support: 10 %

Confidence: 50 %

Rules:

```
Coupon->New Releases (52)
```

- *“It can been seen that the diverse rules that have been generated from the data set. Again, even with the lowered thresholds, the coupon seems to be the most common item in the formation of frequent item sets and therefore the derived association rules:”*



Association Parameters

Support: 1 %

Confidence: 10 %

Rules:

```
Adult->Coupon (23)
Childrens->Coupon (20)
Childrens->New Releases (22)
Coupon->New Releases (52)
Coupon->Regular Movies (10)
Free Movie->New Releases (45)
New Releases->Coupon (30)
Nintendo 64->Coupon (26)
Regular Movies->Coupon (21)
```

Summary

- **Association rule mining is the general process of finding which things go together and the prediction rules between subsets.**
  - Typical example: market basket analysis.
  - Results can serve both purposes of description and predication.
- **The approach is able to find relationships patterns in large databases without restrictions and the need of training data.**
- **Caution must be given in the interpretation of association rules since many discovered relationships turn out to be trivial.**
- **Apriori algorithm is the most well studied method and used in most commercial products**
  - uses the generate-and-test strategy that applied apriori knowledge for pruning the search space for finding all frequent itemsets
- **Association analysis on different types of data**

Review Questions

1. Given an input dataset and the Apriori algorithm, how to trace the algorithm for intermediate results?
2. How to derive strong rules from the given frequent itemsets ***L*** and a min_*conf rate*?
3. How to improve the efficiency of the rule generation procedure by applying the apriori property?
4. What are the two general purposes of DM, use some examples of mined association patterns to explain for each purpose?
5. How can the association mining process be mapped to the empirical cycle model of scientific research?