Data Mining -- Association Rules

Instructor: Jen-Wei Huang

Office: 92528 in the EE building
jwhuang@mail.ncku

Association Rules

- Finding association, correlation or causal structures among sets of items or objects in transactional, relational DB
- Examples
  - bread $\wedge$ milk -> butter
  - age("25~35") $\wedge$ income("35,000~40,000) -> buyer(Lancer)

Example

Tid	Items
100	A, C, D
200	B, C, E
300	A, B, C, E
400	B, E

min_support = 2
min_conf = 2/3

- Frequent itemsets
  - {A}, {B}, {C}, {E}, {A,C}, {B,C}, {B,E}, {C,E}, {B,C,E}
- Strong rules
  - {B, E}→C (2/3)
  - C→A (2/3)
  - A→C (2/2)

Definitions

- $I = \{i_1, i_2, i_3 \ldots i_n\}$: the set of all items
  - Itemset: a set of items
- Association rule: $A \rightarrow B$,
  - where $A \subset I$, $B \subset I$, $A \cap B = \emptyset$
- support $(A \rightarrow B)$ = Prob.$(A \cup B)$
- confidence$(A \rightarrow B)$ = Prob.$(A \cup B / A)$
  - Strong rule: satisfy both minimum support & confidence

Definitions

- $I = \{i_1, i_2, i_3 \ldots i_n\}$: the set of all items
- $T \subseteq I$: a transaction
- D: a set of T, transaction DB
- itemset: a set of items
- k-itemset: an itemset that contains k items

Tid	Items
100	A, C, D
200	B, C, E
300	A, B, C, E
400	B, E

Frequent Pattern

- First proposed by Agrawal [1]
- A pattern that occurs frequently in a data set
- Finding inherent regularities in data
- Foundation for many essential data mining tasks

- In association rule mining, we want to find frequent itemsets, i.e., itemsets whose support are no less than a min_supp threshold.

Apriori Algorithm [2]

- A candidate generation and test approach
- Two steps:
  - Finding all frequent itemsets
  - Deriving valid association rules

- Downward closure property
  - Any subset of a frequent itemset must be frequent
  - E.g.) If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - If there is any itemset which is infrequent, its superset should not be frequent

Apriori Algorithm

- Scan DB once to get frequent 1-itemset
- For frequent k-itemsets, repeat followings
  - Generate length (k+1) candidate itemsets from frequent-k itemsets
  - Test the candidate itemsets against DB
  - Terminate when no frequent or candidate set can be generated
- Compute confidences from all frequent k-itemsets (k>1)

An Example

min_support = 2

Database DB

Tid	Items
100	A, C, D
200	B, C, E
300	A, B, C, E
400	B, E

1^{st} 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}

2^{nd} 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}

3^{rd} scan

L_3

Itemset	sup
{B, C, E}	2

Candidate Generation

- Step 1: self-joining L_k
- Step 2: pruning
- E.g.)
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: $L_3 * L_3$
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - $C_4 = \{abcd\}$

Pseudo-Code

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
    increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
return $\cup_k L_k$;

Association Rules Computation

for each large itemset m do
  for each subset p of m do
    if (sup(m)/sup(m-p)>= minconf) then
      output the rule (m-p)=>p with conf= sup(m)/sup(m-p) and support=sup(m)

Example

- Frequent k-itemsets (k>1) generated from the previous step:
  - {A, C}, {B, C}, {B, E}, {C, E}, {B, C, E}
- Scan DB to test if the confidences of the corresponding ARs are valid.
  - A->C, C->A
  - B->C, C->B
  - B->E, E->B
  - C->E, E->C
  - B->CE, C->BE, E->BC, BC->E, BE->C, CE->B

Redundant Rules

- For the same support and confidence, if we have a rule {a,d}->{c,e,f,g}, do we need
  - {a,d}->{c,e,f}
  - {a}->{c,e,f,g}
  - {a,d,c}->{e,f,g}
  - {a}->{d,c,e,f,g} ?
- Maximal association rules

Interestingness Measure

- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading
  - The overall % of students eating cereal is 75% > 66.7%.
- *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is more accurate, although with lower support and confidence
- Measure of dependent/correlated events: lift

$$lift = \frac{P(A \cup B)}{P(A)P(B)}$$

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

$$lift(B,C) = \frac{2000/5000}{3000/5000 * 3750/5000} = 0.89$$

$$lift(B,\neg C) = \frac{1000/5000}{3000/5000 * 1250/5000} = 1.33$$

Improvements of Apriori

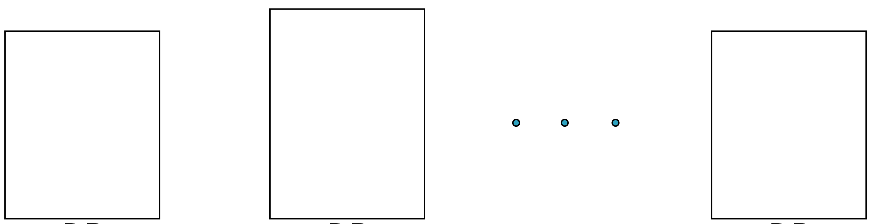
- Major computational challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Scan Reduction

- Reduce Scans of database
- Compute candidate k-itemsets from candidate (k-1)-itemsets instead of frequent (k-1)-itemsets
- Two scan methods:
  - Scan DB the first time for frequent 1-itemsets
  - Compute all candidate k-frequent itemsets from frequent 1-itemsets
  - Scan DB the second time to test if candidate k-itemsets are frequent

Partition Database

- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB [3]
  - Step 1: partition database and find local frequent patterns
  - Step 2: consolidate global frequent patterns



DB_1 + DB_2 + … + DB_k = DB

$sup_1(i) < \sigma DB_1$ $sup_2(i) < \sigma DB_2$ $sup_k(i) < \sigma DB_k$ $sup(i) < \sigma DB$

Hash-based Algorithm

- Algorithm DHP [4]: Direct Hashing and Pruning
- Hash table scheme
  - Eliminate infrequent candidate itemsets in the early phase
- Transaction items pruning
  - Eliminate infrequent items from the database

Candidate Itemsets Pruning

TID	Items
100	A,C,D
200	B,C,E
300	A,B,C,E
400	B,E

L_1=<A,B,C,E> Minimum Support Frequency=2

<AC>,<AD>,<CD>
<BC>,<BE>,<CE>
<AB>,<AC>,<AE>,<BC>,<BE>,<CE>
<BE>

h(x,y)=((ord(x)*10+ord(y)) mod 7

<CE>				<BE>		<AC>
<CE>		<BC>		<BE>		<CD>
<AD>	<AE>	<BC>		<BE>	<AB>	<AC>
3	1	2	0	3	1	3
0	1	2	3	4	5	6

H_2

$C_2=L_1*L_1$

Itemset
<AB>
<AC>
<AE>
<BC>
<BE>
<CE>

→

C_2'

Itemset
<AC>
<BC>
<BE>
<CE>

pruning <AB>, <AE>

Hash table building

Candidate pruning

Transaction Items Pruning

- A transaction should contain at least k+1 k-itemsets to support (k+1)-itemsets
  - Each item should appear at least k times

C_2=<<AC>, <BC>, <BE>, <CE>>

TID	Items
100	A,C,D
200	B,C,E
300	A,B,C,E
400	B,E

Before trimming

a[0]	a[1]	a[2]	a[3]
1	1	0	
2	2	2	
1	2	3	2
1	1		

Counter array

Trimming information collecting

	TID	Items
Trim A,C,D	100	
	200	B,C,E
Trim A	300	B,C,E
Trim B,E	400	

After trimming

Transaction trimming

References

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. SIGMOD'93
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. VLDB'94
- [3] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. VLDB'95.
- [4] J. S. Park, M. S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. SIGMOD'95

References

- Slides from Prof. J.-W. Han, UIUC
- Slides from Prof. M.-S. Chen, NTU
- Slides from Prof. W.-Z. Peng, NCTU

HW1

- Compute strong association rules from the following DB with
  - min_supp = 50%
  - min_conf = 66%
- DB:
  - 100 A, C, D
    200 B, C, E
    300 A, B, C, E
    400 B, E
    500 A, C, E
    600 B, C, D