Association Rule Mining

CS145

Fall 2015

Outline

- What is association rule mining?
- Methods for association rule mining
- Extensions of association rule

What Is Association Rule Mining?

- Frequent patterns: patterns (set of items, sequence, etc.) that occur frequently in a database [AIS93]
- Frequent pattern mining: finding regularities in data
  - What products were often purchased together?
    - Beer and diapers?!
  - What are the subsequent purchases after buying a car?
  - Can we automatically profile customers?

Why Essential?

- Foundation for many data mining tasks
  - Association rules, correlation, causality, sequential patterns, structural patterns, spatial and multimedia patterns, associative classification, cluster analysis, iceberg cube, …
- Broad applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, web log (click stream) analysis, …

Basics

- Itemset: a set of items
  - E.g., acm={a, c, m}
- Support of itemsets
  - Sup(acm)=3
- Given min_sup=3, acm is a frequent pattern
- Frequent pattern mining: find all frequent patterns in a database

Transaction database TDB

TID	Items bought
100	f, a, c, d, g, I, m, p
200	a, b, c, f, l,m, o
300	b, f, h, j, o
400	b, c, k, s, p
500	a, f, c, e, l, p, m, n

Frequent Pattern Mining: A Road Map

- Boolean vs. quantitative associations
  - age(x, "30..39") ^ income(x, "42..48K") → buys(x, "car") [1%, 75%]
- Single dimension vs. multiple dimensional associations
- Single level vs. multiple-level analysis
  - What brands of beers are associated with what brands of diapers?

Extensions & Applications

- Correlation, causality analysis & mining interesting rules
- Maxpatterns and frequent closed itemsets
- Constraint-based mining
- Sequential patterns
- Periodic patterns
- Structural Patterns
- Computing iceberg cubes

Frequent Pattern Mining Methods

- Apriori and its variations/improvements
- Mining frequent-patterns without candidate generation
- Mining max-patterns and closed itemsets
- Mining multi-dimensional, multi-level frequent patterns with flexible support constraints
- Interestingness: correlation and causality

Apriori: Candidate Generation-and-test

- Any subset of a frequent itemset must be also frequent — an anti-monotone property
  - A transaction containing {beer, diaper, nuts} also contains {beer, diaper}
  - {beer, diaper, nuts} is frequent → {beer, diaper} must also be frequent
- No superset of any infrequent itemset should be generated or tested
  - Many item combinations can be pruned

Apriori-based Mining

- Generate length (k+1) candidate itemsets from length k frequent itemsets, and
- Test the candidates against DB

Apriori Algorithm

- A level-wise, candidate-generation-and-test approach (Agrawal & Srikant 1994)

Data base D

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Min_sup=2

Scan D

1-candidates

Itemset	Sup
a	2
b	3
c	3
d	1
e	3

Freq 1-itemsets

Itemset	Sup
a	2
b	3
c	3
e	3

2-candidates

Itemset
ab
ac
ae
bc
be
ce

Scan D

Counting

Itemset	Sup
ab	1
ac	2
ae	1
bc	2
be	3
ce	2

Freq 2-itemsets

Itemset	Sup
ac	2
bc	2
be	3
ce	2

3-candidates

Itemset
bce

Scan D

Freq 3-itemsets

Itemset	Sup
bce	2

The Apriori Algorithm

- 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
  - 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
- return $\cup_k L_k$;

Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- How to count supports of candidates?

How to Generate Candidates?

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-join L_{k-1}

  INSERT INTO C_k

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

  FROM L_{k-1} p, L_{k-1} q

  WHERE $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$
- Step 2: pruning
  - For each itemset c in C_k do
    - For each (k-1)-subsets s of c do if (s is not in L_{k-1}) then delete c from C_k

Example of Candidate-generation

- $L_3=\{abc, abd, acd, ace, bcd\}$
- 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
- $C_4=\{abcd\}$

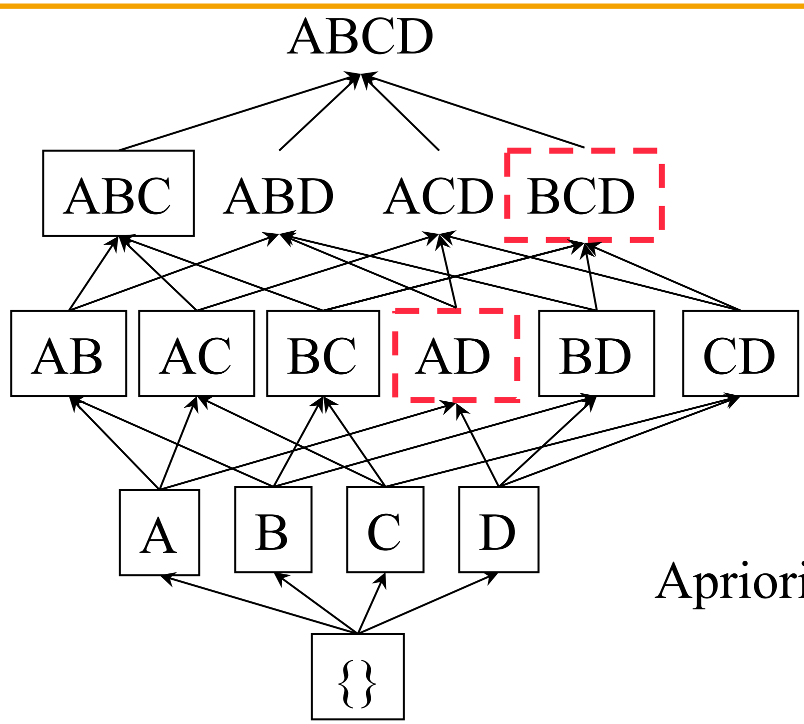
How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemsets are stored in a hash-tree
  - Leaf node of hash-tree contains a list of itemsets and counts
  - Interior node contains a hash table
  - Subset function: finds all the candidates contained in a transaction

Challenges of Frequent Pattern Mining

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

DIC: Reduce Number of Scans



Itemset lattice

S. Brin R. Motwani, J. Ullman, and S. Tsur, 1997.

- Once both A and D are determined frequent, the counting of AD can begin
- Once all length-2 subsets of BCD are determined frequent, the counting of BCD can begin