Unit II : ASSOCIATION RULES

Basic Concepts - Market Basket Analysis - Frequent Itemsets, Closed Itemsets and Association Rules - Frequent Itemset Mining Methods – Apriori Algorithm – Generating Association Rules - Frequent pattern growth - Mining Various Kinds of Association Rules

Mining Frequent Patterns, Association and Correlations

- Basic concepts and a road map
- Efficient and scalable frequent itemset mining methods
- Mining various kinds of association rules
- From association mining to correlation analysis
- Constraint-based association mining
- Summary

What Is Frequent Pattern Analysis?

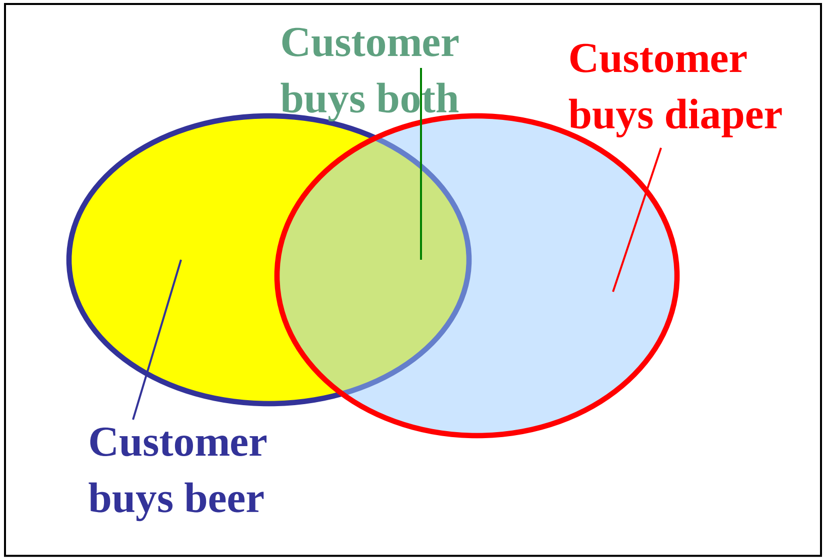
- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- First proposed by Agrawal, Imielinski, and Swami [AIS93] in the context of frequent itemsets and association rule mining
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

Why Is Freq. Pattern Mining Important?

- Discloses an intrinsic and important property of data sets
- Forms the foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Classification: associative classification
  - Cluster analysis: frequent pattern-based clustering
  - Data warehousing: iceberg cube and cube-gradient
  - Semantic data compression: fascicles
  - Broad applications

Basic Concepts: Frequent Patterns and Association Rules

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



- Itemset X = $\{x_1, \ldots, x_k\}$
- Find all the rules $X \rightarrow Y$ with minimum support and confidence
  - support, *s*, probability that a transaction contains $X \cup Y$
  - confidence, *c*, conditional probability that a transaction having X also contains *Y*

Let $sup_{min} = 50\%$, $conf_{min} = 50\%$

Freq. Pat.: {A:3, B:3, D:4, E:3, AD:3}

Association rules:

$A \rightarrow D$ (60%, 100%)

$D \rightarrow A$ (60%, 75%)

Closed Patterns and Max-Patterns

- A long pattern contains a combinatorial number of sub-patterns, e.g., $\{a_1, \ldots, a_{100}\}$ contains $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100} - 1 = 1.27*10^{30}$ sub-patterns!
- Solution: *Mine closed patterns and max-patterns instead*
- An itemset X is closed if X is *frequent* and there exists *no super-pattern* Y $\supset$ X, *with the same support* as X (proposed by Pasquier, et al. @ ICDT'99)
- An itemset X is a max-pattern if X is frequent and there exists no frequent super-pattern Y $\supset$ X (proposed by Bayardo @ SIGMOD'98)
- Closed pattern is a lossless compression of freq. patterns
  - Reducing the # of patterns and rules

Closed Patterns and Max-Patterns

- Exercise. DB = {<a_1, …, a_{100}>, < a_1, …, a_{50}>}
  - Min_sup = 1.
- What is the set of closed itemset?
  - <a_1, …, a_{100}>: 1
  - < a_1, …, a_{50}>: 2
- What is the set of max-pattern?
  - <a_1, …, a_{100}>: 1
- What is the set of all patterns?
  - !!

Mining Frequent Patterns, Association and Correlations

- Basic concepts and a road map
- Efficient and scalable frequent itemset mining methods
- Mining various kinds of association rules
- From association mining to correlation analysis
- Constraint-based association mining
- Summary

Scalable Methods for Mining Frequent Patterns

- The downward closure property of frequent patterns
  - Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- Scalable mining methods: Three major approaches
  - Apriori (Agrawal & Srikant@VLDB'94)
  - Freq. pattern growth (FPgrowth—Han, Pei & Yin @SIGMOD'00)
  - Vertical data format approach (Charm—Zaki & Hsiao @SDM'02)

Apriori: A Candidate Generation-and-Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested! (Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94)
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no frequent or candidate set can be generated

The Apriori Algorithm—An Example

$Sup_{min} = 2$

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

The Apriori Algorithm

- 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 $\cup_k L_k$;

Important Details of Apriori

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

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

  insert into $\boldsymbol{C_k}$

  select ***p.item$_1$, p.item$_2$, …, p.item$_{k-1}$, q.item$_{k-1}$***

  from ***L_{k-1} p, L_{k-1} q***

  where ***p.item$_1$=q.item$_1$, …, p.item$_{k-2}$=q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$***
- Step 2: pruning

  forall ***itemsets c in C_k*** do

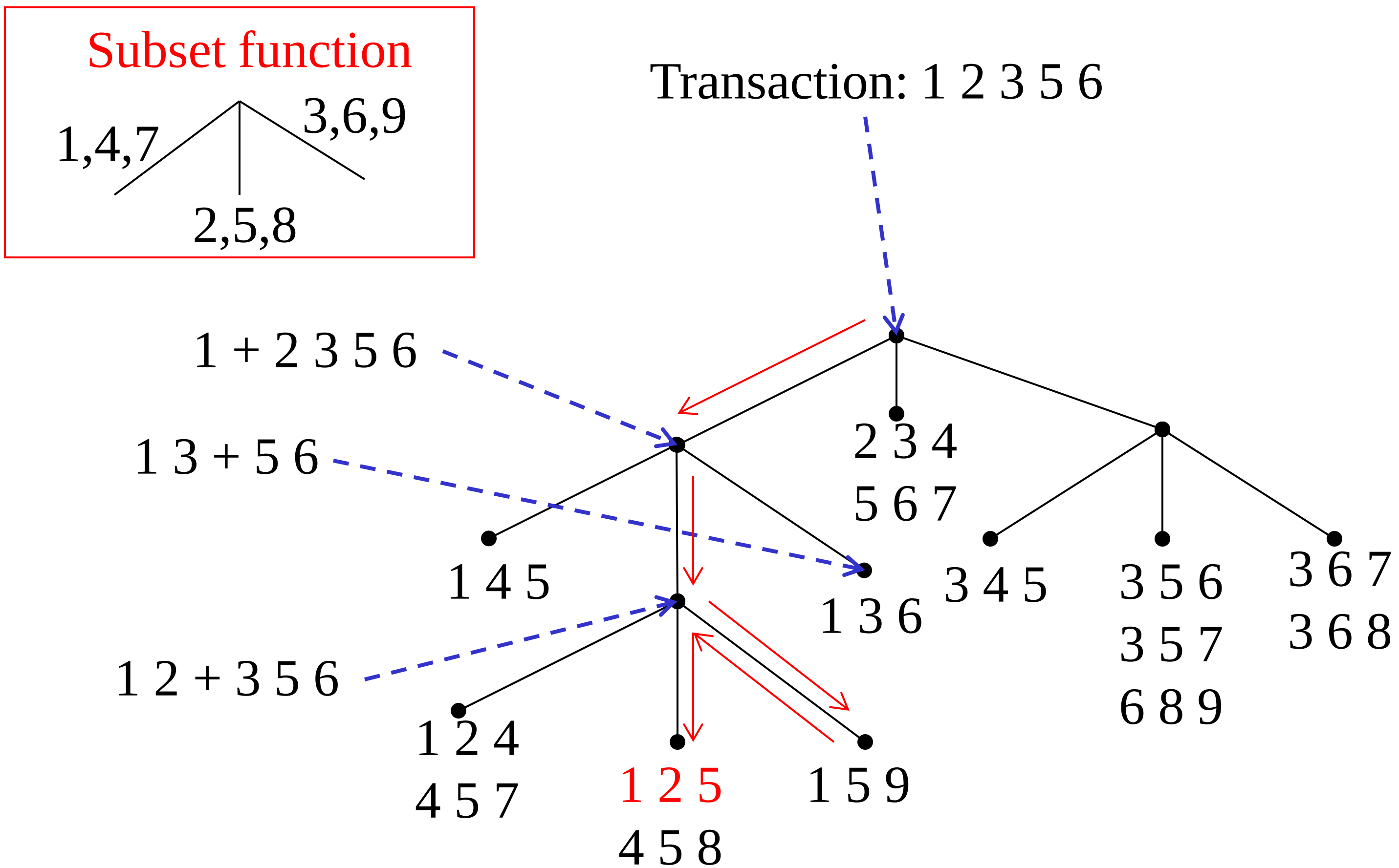
  forall ***(k-1)-subsets s of c*** do

  if *(s is not in L_{k-1})* **then delete** *c* **from** C_k

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

Example: Counting Supports of Candidates



Efficient Implementation of Apriori in SQL

- Hard to get good performance out of pure SQL (SQL-92) based approaches alone
- Make use of object-relational extensions like UDFs, BLOBs, Table functions etc.
  - Get orders of magnitude improvement
- S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In SIGMOD'98

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 passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Partition: Scan Database Only Twice

- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns
- A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In *VLDB'95*

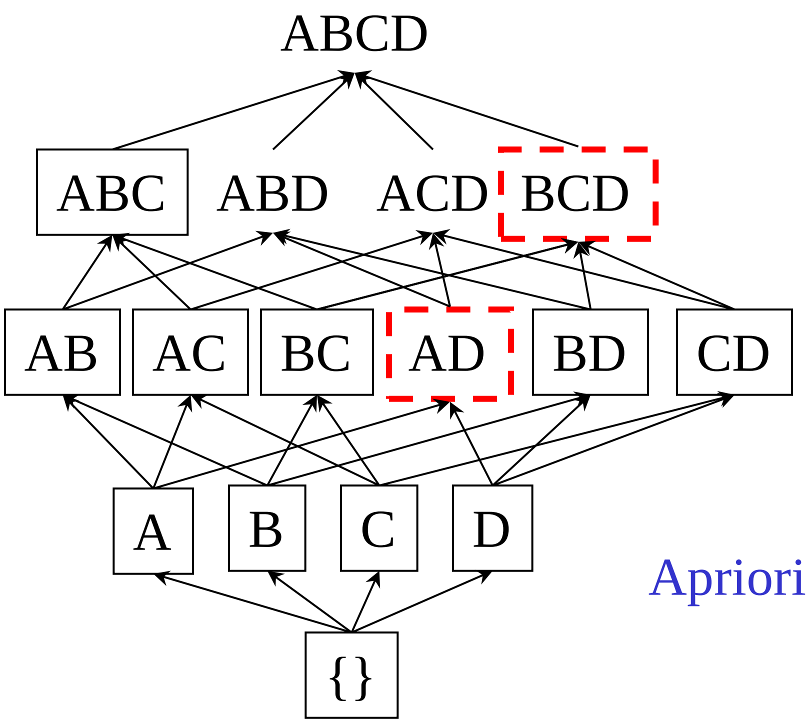
DHP: Reduce the Number of Candidates

- A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Candidates: a, b, c, d, e
  - Hash entries: {ab, ad, ae} {bd, be, de} …
  - Frequent 1-itemset: a, b, d, e
  - ab is not a candidate 2-itemset if the sum of count of {ab, ad, ae} is below support threshold
- J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules.

Sampling for Frequent Patterns

- Select a sample of original database, mine frequent patterns within sample using Apriori
- Scan database once to verify frequent itemsets found in sample, only *borders* of closure of frequent patterns are checked
  - Example: check *abcd* instead of *ab, ac, …, etc.*
- Scan database again to find missed frequent patterns
- H. Toivonen. Sampling large databases for association rules.

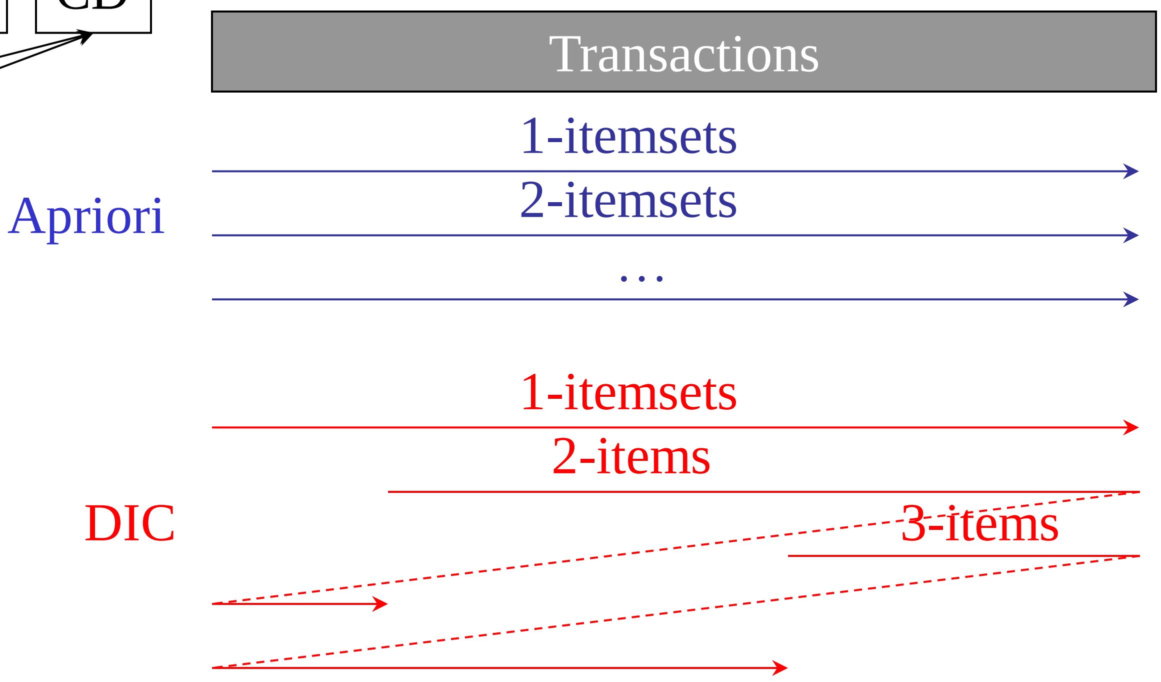
DIC: Reduce Number of Scans



Itemset lattice

Dynamic itemset counting and implication rules for market basket data.

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



Bottleneck of Frequent-pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100} - 1 = 1.27 \times 10^{30}$!
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?