

CSE5334 DATA MINING

Lecture 11: Association Rule Mining (1) CSE4392/5334 Data Mining, Fall 2009
Department of Computer Science and Engineering, University of Texas at Arlington
Chengkai Li (Slides courtesy of Vipin Kumar and Jiawei Han)

Association Rule Mining

Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

{Diaper} → {Beer},
 {Milk, Bread} → {Eggs,Coke},
 {Beer, Bread} → {Milk},

Implication means co-occurrence, not causality!

Definition: Frequent Itemset

- Itemset**
 - A collection of one or more items
 - Example: {Milk, Bread, Diaper}
 - k-itemset
 - An itemset that contains k items
- Support count (σ)**
 - Frequency of occurrence of an itemset
 - E.g. σ({Milk, Bread, Diaper}) = 2
- Support**
 - Fraction of transactions that contain an itemset
 - E.g. s({Milk, Bread, Diaper}) = 2/5
- Frequent Itemset**
 - An itemset whose support is greater than or equal to a *minsup* threshold

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Definition: Association Rule

- Association Rule**
 - An implication expression of the form X → Y, where X and Y are itemsets
 - Example: {Milk, Diaper} → {Beer}
- Rule Evaluation Metrics**
 - Support (s)**
 - Fraction of transactions that contain both X and Y
 - Confidence (c)**
 - Measures how often items in Y appear in transactions that contain X

Example: {Milk, Diaper} ⇒ Beer

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

Association Rule Mining Task

Given a set of transactions T, the goal of association rule mining is to find all rules having

- support ≥ *minsup* threshold
- confidence ≥ *minconf* threshold

Brute-force approach:

- List all possible association rules
- Compute the support and confidence for each rule
- Prune rules that fail the *minsup* and *minconf* thresholds

⇒ **Computationally prohibitive!**

Mining Association Rules

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Rules:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
 {Milk,Beer} → {Diaper} (s=0.4, c=1.0)
 {Diaper,Beer} → {Milk} (s=0.4, c=0.67)
 {Beer} → {Milk,Diaper} (s=0.4, c=0.67)
 {Diaper} → {Milk,Beer} (s=0.4, c=0.5)
 {Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Observations:

- All the above rules are binary partitions of the same itemset: {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- Two-step approach:
 - Frequent Itemset Generation**
 - Generate all itemsets whose support \geq minsup
 - Rule Generation**
 - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset
- Frequent itemset generation is still computationally expensive

Frequent Itemset Generation

Frequent Itemset Generation

Given d items, there are 2^d possible candidate itemsets

Frequent Itemset Generation

- Brute-force approach:
 - Each itemset in the lattice is a **candidate** frequent itemset
 - Count the support of each candidate by scanning the database

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

← Transactions (N rows, W columns) → List of Candidates (M candidates)

- Match each transaction against every candidate
- Complexity $\sim O(NMw) \Rightarrow$ Expensive since $M = 2^d$!!!

Computational Complexity

- Given d unique items:
 - Total number of itemsets = 2^d
 - Total number of possible association rules:

$$R = \sum_{k=1}^{d-1} \binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j}$$

$$= 3^d - 2^{d+1} + 1$$

If $d=6$, $R = 602$ rules

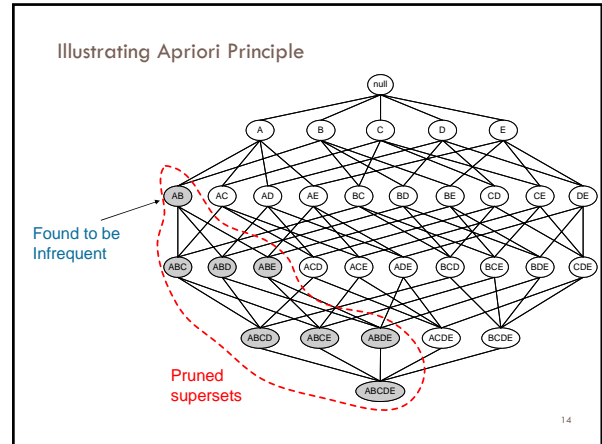
Frequent Itemset Generation Strategies

- Reduce the **number of candidates** (M)
 - Complete search: $M=2^d$
 - Use pruning techniques to reduce M
- Reduce the **number of transactions** (N)
 - Reduce size of N as the size of itemset increases
 - Used by DHP and vertical-based mining algorithms
- Reduce the **number of comparisons** (NM)
 - Use efficient data structures to store the candidates or transactions
 - No need to match every candidate against every transaction

Reducing Number of Candidates

- Apriori principle:
 - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$
 - Support of an itemset never exceeds the support of its subsets
 - This is known as the **anti-monotone** property of support



Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread, Milk}	3
{Bread, Beer}	2
{Bread, Diaper}	3
{Milk, Beer}	2
{Milk, Diaper}	3
{Beer, Diaper}	3

Pairs (2-itemsets)
(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered, ${}^4C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning, $6 + 6 + 1 = 13$

Itemset	Count
{Bread, Milk, Diaper}	3

Triplets (3-itemsets)

Another Example

Database TDB

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

$SUP_{min} = 2$

1st scan

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

2nd scan

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

3rd scan

Itemset	Sup
{A, C, E}	2

Apriori Algorithm

- Let $k=1$
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
 - Generate candidate $(k+1)$ -itemsets from frequent k -itemsets
 - Prune candidate $(k+1)$ -itemsets containing some infrequent k -itemset
 - Count the support of each candidate by scanning the DB
 - Eliminate infrequent candidates, leaving only those that are frequent

1. Generate Candidate $(k+1)$ itemsets

$SUP_{min} = 2$

Input: frequent k -itemsets L_k
Output: frequent $(k+1)$ -itemsets L_{k+1}

Procedure:

- Candidate generation, by self-join $L_k \bowtie L_k$
 - For each pair of $P = \{p_1, p_2, \dots, p_k\} \in L_k, Q = \{q_1, q_2, \dots, q_k\} \in L_k$
 - if $p_1=q_1, \dots, p_{k-1}=q_{k-1}, p_k < q_k$, add $\{p_1, \dots, p_k, q_k\}$ into C_{k+1}

Example: $L_2 = \{AB, AC, AE, CE\}$

- AB and AC \Rightarrow ABC
- AB and AE \Rightarrow ABE
- AC and AE \Rightarrow ACE

2. Prune Candidates

Sup_{min} = 2

Input: frequent k-itemsets L_k
 Output: frequent (k+1)-itemsets L_{k+1}

Procedure:

- Candidate generation, by self-join $L_k * L_k$
 - For each pair of $P = \{p_1, p_2, \dots, p_k\} \in L_k, Q = \{q_1, q_2, \dots, q_k\} \in L_k$
 - if $p_1=q_1, \dots, p_{k-1}=q_{k-1}, p_k < q_k$, add $\{p_1, \dots, p_{k-1}, p_k, q_k\}$ into C_{k+1}
- Prune candidates that contain infrequent k-itemsets

Example: $L_2 = \{AB, AC, AE, CE\}$

- AB and AC \Rightarrow ABC, pruned because BC is not frequent
- AB and AE \Rightarrow ABE, pruned because BE is not frequent
- AC and AE \Rightarrow ACE

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

Itemset	sup
{A, B, C}	0
{A, B, E}	0
{A, C, E}	2

3. Count support of candidates and 4. Eliminate infrequent candidates

Sup_{min} = 2

Input: frequent k-itemsets L_k
 Output: frequent (k+1)-itemsets L_{k+1}

Procedure:

- Candidate generation, by self-join $L_k * L_k$
 - For each pair of $P = \{p_1, p_2, \dots, p_k\} \in L_k, Q = \{q_1, q_2, \dots, q_k\} \in L_k$
 - if $p_1=q_1, \dots, p_{k-1}=q_{k-1}, p_k < q_k$, add $\{p_1, \dots, p_{k-1}, p_k, q_k\}$ into C_{k+1}
- Prune candidates that contain infrequent k-itemsets
- Count the support of each candidate by scanning the DB
- Eliminate infrequent candidates

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

Itemset	sup
{A, C, E}	2

Reducing Number of Comparisons

Candidate counting:

- Scan the database of transactions to determine the support of each candidate itemset
- To reduce the number of comparisons, store the candidates in a hash structure
 - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Hash Structure

Buckets

Generate Hash Tree

Suppose you have 15 candidate itemsets of length 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

You need:

- Hash function
- Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)
- An order on the items (e.g., 1 .. 9, Beer, Bread, Coke, Diaper, Egg, Milk)

Hash function

1,4,7 hashed on the 1st item

2,5,8 hashed on the 2nd item

3,6,9 hashed on the 3rd item

Association Rule Discovery: Hash tree

Hash Function

Candidate Hash Tree

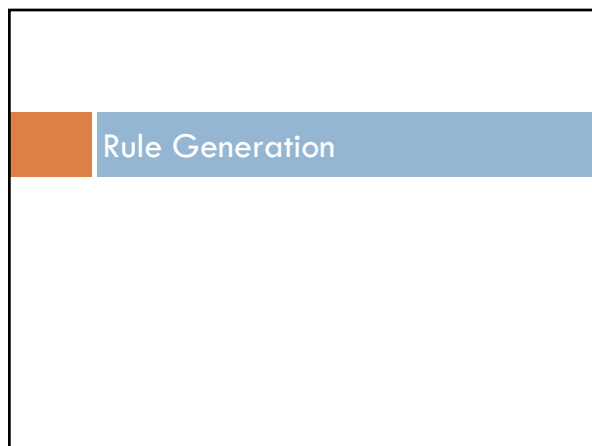
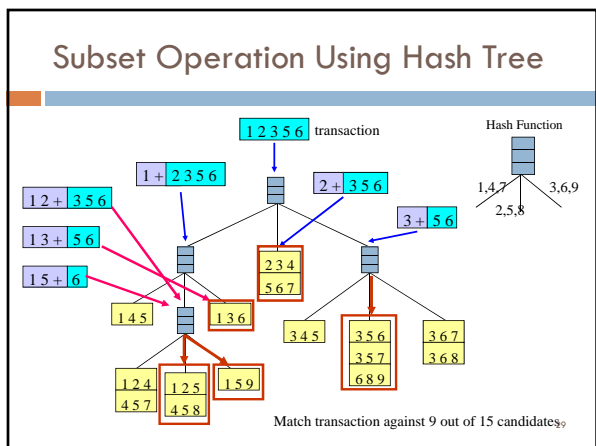
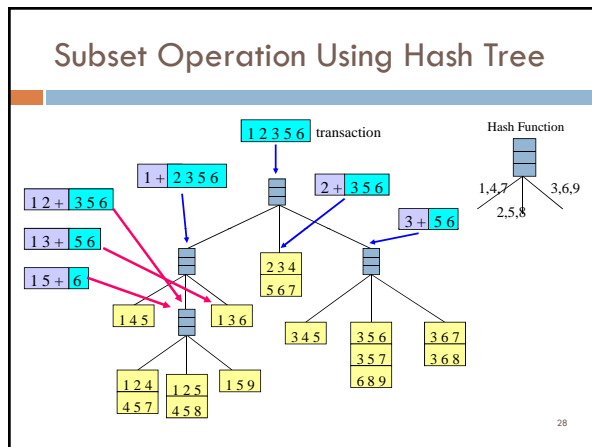
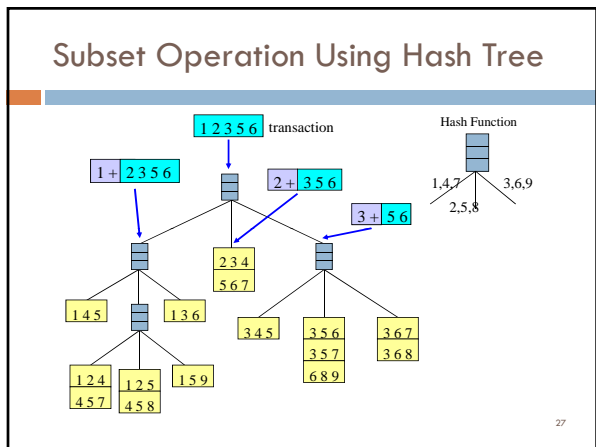
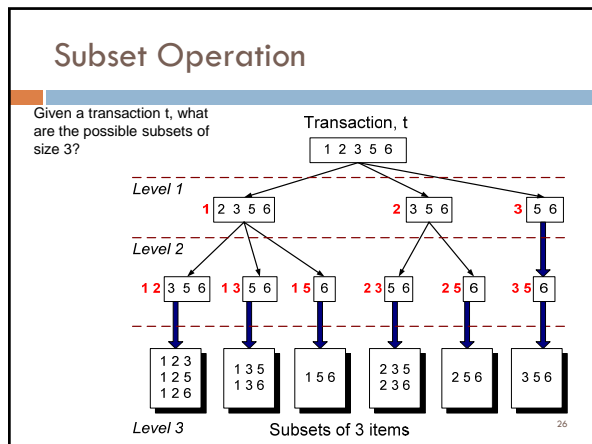
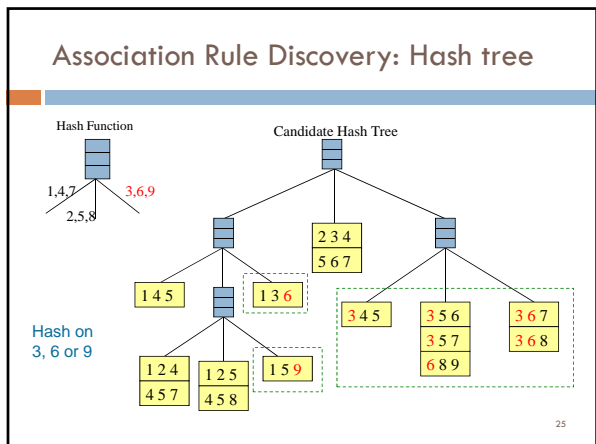
Hash on 1, 4 or 7

Association Rule Discovery: Hash tree

Hash Function

Candidate Hash Tree

Hash on 2, 5 or 8



Rule Generation

- Given a frequent itemset L , find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement
 - If $\{A,B,C,D\}$ is a frequent itemset, candidate rules:

$ABC \rightarrow D,$	$ABD \rightarrow C,$	$ACD \rightarrow B,$	$BCD \rightarrow A,$
$A \rightarrow BCD,$	$B \rightarrow ACD,$	$C \rightarrow ABD,$	$D \rightarrow ABC$
$AB \rightarrow CD,$	$AC \rightarrow BD,$	$AD \rightarrow BC,$	$BC \rightarrow AD,$
$BD \rightarrow AC,$	$CD \rightarrow AB,$		
 - If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \emptyset$ and $\emptyset \rightarrow L$)

31

Rule Generation

- How to efficiently generate rules from frequent itemsets?
 - In general, confidence does not have an anti-monotone property
 - $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$
 - But confidence of rules generated from the same itemset has an anti-monotone property
 - e.g., $L = \{A,B,C,D\}$:
 - $c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$
- Confidence is anti-monotone w.r.t. the RHS of the rule

32

Rule Generation for Apriori Algorithm

Lattice of rules

33

Rule Generation for Apriori Algorithm

- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent
 - $\text{join}(CD \Rightarrow AB, BD \Rightarrow AC)$ would produce the candidate rule $D \Rightarrow ABC$
- Prune rule $D \Rightarrow ABC$ if its subset $AD \Rightarrow BC$ does not have high confidence

34