Data Mining (EECS6412)

Frequent Pattern and Association Rule Mining

Aijun An

(Modified from Jiawei Han's slides)

Outline

- Basic concepts of association rule learning
- Apriori algorithm
- FP-Growth Algorithm
- Finding interesting rules.

Why Mining Association Rules?

- Objective:
  - Finding interesting co-occurring items (or objects, events) in a given data set.
- Examples:
  - Given a database of transactions, each transaction is a list of items (purchased by a customer in a visit), you may find:

    computer → financial_management_software [support=2%, confidence=60%]
  - From a student database, you may find
    - major(x, "CS") ^ gpa(x, "A") → has_taken(x, "DB") [1%, 75%]

3

Why Mining Association Rules? (Cont'd)

- Popular application: *Basket data analysis*
  - place items frequently bought together close to each other to increase sales of these items.
  - *?* → *iPad* (What the store should do to boost sales of the particular product, i.e., *iPad*)
  - *iPad* → *?* (What other products should the store stock up?)

4

What Kind of Databases?

Transactional database TDB

TID	Items
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

- Itemset: a set of items
- A ***transaction*** is a tuple (tid, X)
  - Transaction ID tid
  - Itemset X
- A ***transactional database*** is a set of transactions
  - In many cases, a transaction database can be treated as a set of itemsets (ignore TIDs)
- Association rule from TDB (relates two itemsets):
  - {a, c, m} → {l} [support=40%, confidence=66.7%]

5

What Kind of Databases? (*Contd*)

Relational database (RDB)

Day	Outlook	Temp	Humid	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- An attribute-value pair
  - E.g., Outlook = sunny
- Record: a set of attribute-value pairs
- Relational DB: a set of records

- Association rule from RDB (relates two sets of attribute-value pairs):

  (Outlook=sunny)∧(Temp=hot) → (PlayTennis=No) [support=14%, confidence=100%]

6

Definition of Association Rule

- An association rule is of the form:

$$X \rightarrow Y \ [\text{support, confidence}]$$

where antecedent, consequent

  - $X \subset I$, $Y \subset I$, $X \cap Y = \varnothing$ and I is a set of items (objects or attribute-value pairs).
  - support: probability that a transaction (or a record) contains X and Y, i.e.,

$$\text{support}\ (X \rightarrow Y) = P(X \cup Y)$$

  - confidence: conditional probability that a transaction (or a record) having X also contains Y, i.e.,

$$\text{confidence}(X \rightarrow Y) = P(Y|X)$$

- A rule associates one set of items (or attribute-value pairs) with another set of items (or attribute-value pairs)

7

Support and Confidence: Example

$$\text{support}(X \rightarrow Y) = P(X \cup Y)$$

$$\text{confidence}(X \rightarrow Y) = P(Y|X)$$

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Relative frequency is used to estimate the probability.

- $\{A\} \rightarrow \{C\}$ (50%, 66.7%)
- $\{C\} \rightarrow \{A\}$ (50%, 100%)
- $\{A, C\} \rightarrow \{B\}$ (25%, 50%)
- $\{A, B\} \rightarrow \{E\}$ (0%, 0%)

8

Mining Association Rules

- Problem statement

  Given a *minimum support* (*min_sup*), also called *support threshold*, and a *minimum confidence* (*min_conf*), also called *confidence threshold*, find all association rules that satisfy both *min_sup* and *min_conf* from a data set *D*.

9

Basic Concepts

- Strong rules:

  An association rule is *strong* if it satisfies both *min_sup* and *min_conf*.

- *k*-itemset: an itemset that contains *k* items.
  - {computer, financial_management_software} is a 2-itemset.
- Support count of an itemset in data set *D*: number of transactions in *D* that contain the itemset.
- Minimum support count = *min_sup* × total number of transactions in a data set.
- Frequent itemset in a data set *D*: itemset whose support count in *D* is at least the minimum support count.

10

How to Mine Association Rules

- A two-step process:
  - Find all frequent itemsets ---- the key step
  - Generate strong association rules from frequent itemsets.
- Example: given min_sup=50% and min_conf=50%

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

  - Generate strong rules:
    - {A} → {C} [support=50%, confidence=66.6%]
    - {C} → {A} [support=50%, confidence=100%]

11

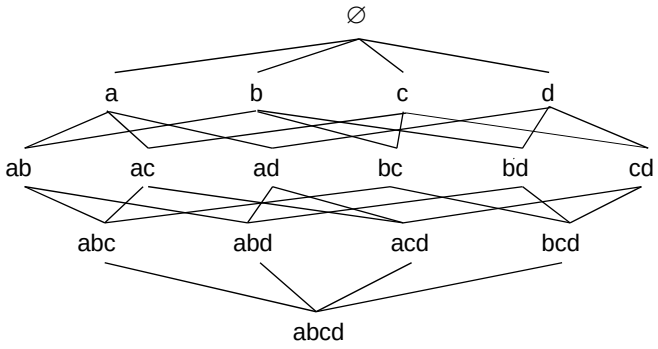
Finding Frequent Itemsets

- Objective: given a database, find all the itemsets (or sets of attribute-value pairs) that satisfy the minimum support count.
- Algorithms
  - Apriori
  - FP-Growth
  - H-Mine
  - Eclat
  - Partition
  - CLOSET
  - CHARM
  - etc.

12

Search Space for Finding All Frequent Itemsets

- Search space for DB with 4 items:



13

Apriori

- The Apriori property (an anti-monotone property), also called *downward closure* property:

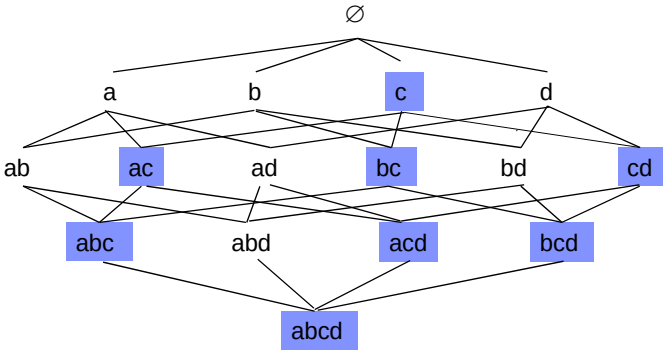
  Any nonempty subset of a frequent itemset must be frequent

  - i.e., if {*A*,*B*,*C*} is a frequent itemset, {A,B}, {A,C}, {B,C}, {*A*}, {*B*} and {C} must also be frequent.
  - This is because a transaction containing {A, B, C} also contains {A,B}, {B,C}, …. Thus, the support count of a subset is not less than that of the superset.
- No superset of any infrequent itemset should be checked.
  - Many item combinations can be pruned from the search space.
- Apriori-based mining (level-wise iterations)
  - Generate length (k+1) candidate itemsets from frequent k-itemsets
  - Test the candidates against DB

14

Pruning Search Space Using Apriori Property

- If *c* is not frequent, all the sets containing *c* are pruned:



15

The Apriori Algorithm

- Based on the Apriori property, use *iterative level-wise approach* and *candidate generation-and-test*
- Pseudo-code:

C_k: a set of candidate itemsets of size k
L_k : the set of frequent itemsets of size k

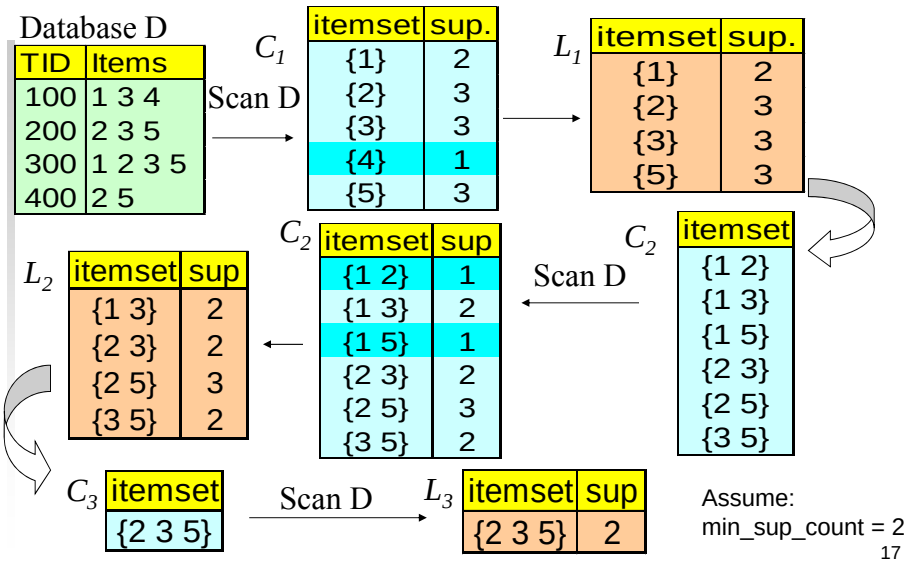
L_1 = {frequent items}; (*Scan database to find all frequent 1-itemsets*)
for (k = 1; L_k !=∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  if C_{k+1}!=∅
    for each transaction t in database do
      increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1}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

Scan database to calculate support for each itemset in C_{k+1}

Level-wise Generation process: $L_k \rightarrow C_{k+1} \rightarrow L_{k+1}$

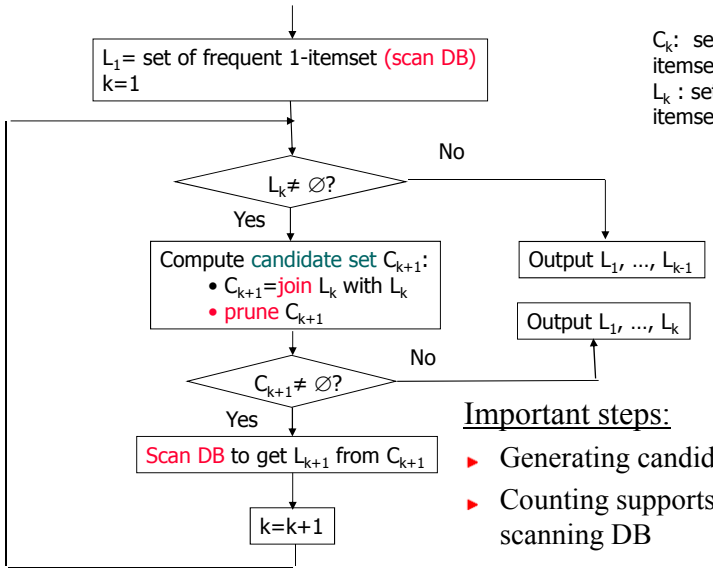
16

The Apriori Algorithm — Example



17

Apriori Algorithm (Flow Chart)



C_k: set of candidate k-itemsets
L_k : set of frequent k-itemsets

Important steps:

- Generating candidates
- Counting supports of candidates by scanning DB

18

How to Generate Candidates?
(i.e., How to Generate C_{k+1} from L_k)

- Given L_k = the set of frequent k-itemsets
- List the items in each itemset of L_k in an order

L_3 =

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

- Given L_k, generate C_{k+1} in two steps:
  - Join Step: Join L_k with L_k by joining two k-itemsets in L_k. Two k-itemsets are joinable if their first (k-1) items are the same and the last item in the first itemset is smaller than the last item in the second itemset (the condition for joining two members of L_k).

    Now, C_4={{1 2 3 4}, {1 3 4 5}}
  - Prune Step: Delete all candidates in C_{k+1} that have a non-frequent subset by checking all length-k subsets of a candidate
    - Now, C_4={{1 2 3 4}}

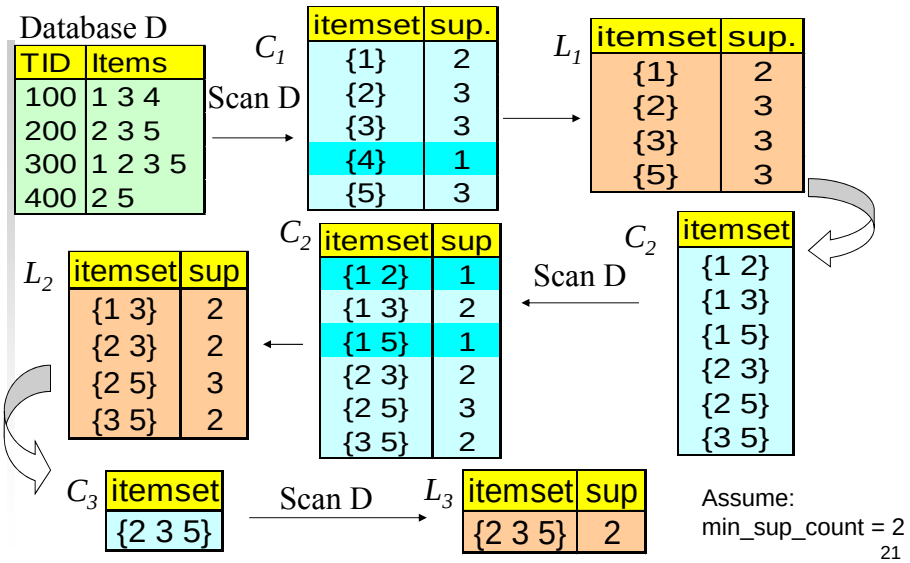
19

Example of Candidate-generation

- L_4={*abcd, abcg, abdg, abef, abeh, acdg, bcdg*}
- Self-joining: L_4*L_4
  - *abcdg* from *abcd* and *abcg*
  - *abefh* from *abef* and *abeh*
- Pruning:
  - *abefh* is removed because *abfh or aefh or befh* is not in L_4
- C_5={*abcdg*}

20

The Apriori Algorithm — Example



21

Support Counting of Candidates in DB scan

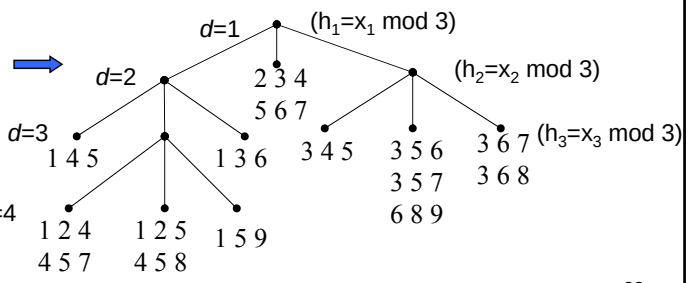
- Objective of candidate support counting
  - Find frequent itemsets L_k from a set of candidates C_k
- Naïve method: match each candidate with each transaction. Time consuming when
  - the total number of candidates is very large
  - one transaction contains many candidates
- Hash tree method:
  - Store candidate itemsets in C_k in a *hash-tree*
    - Leaf node of hash-tree contains a list of candidates and their counts
    - Interior node represents a hash function.
  - Use a *subset function* to find all the candidates contained in a transaction

22

Building a Hash Tree

- Initially treat the root as a leaf node
- Insert itemsets in C_k into the tree
- When the number of itemsets in a leaf node exceeds a specified threshold (such as 3 in the following example), convert the leaf node into an interior node by
  - hashing the itemsets according to the *d*-th item of the itemset, where *d* is the level of the node.

Given C_3 = { 124, 125, 136, 145, 159, 234, 345, 356, 357, 367, 368, 457, 458, 567, 689}



23

Subset Function

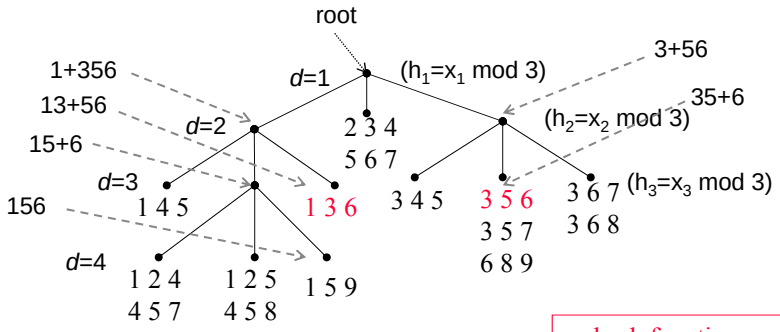
- Functionality
  - Given C_k (in a hash tree) and a transaction *t*, find all the candidates in C_k contained in *t* and increase the count of these candidates:

    Subset (C_k, *t*): candidate itemsets contained in *t*
- Method
  - Try to check each subset of *t* to see if it is contained in the tree as follows:
  - At root, hash on every item in *t* (until the *k*th item from the end) .
  - At an interior node reached by hashing on item *i*, hash on each item that comes after *i* in *t* (until the (*k*-*d*+1)th item from the end , where *d* is the level of the node in the tree), recursively apply to the nodes in the corresponding bucket
  - At a leaf, find itemsets contained in *t*
- Benefit
  - Don't have to match each candidate with each transaction.

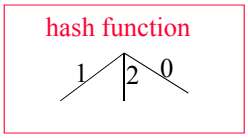
24

Example: finding the candidates contained in a transaction.

- Given a transaction {1 3 5 6} and hash tree:



It only goes to the leaf that may contain subsets of the transaction.



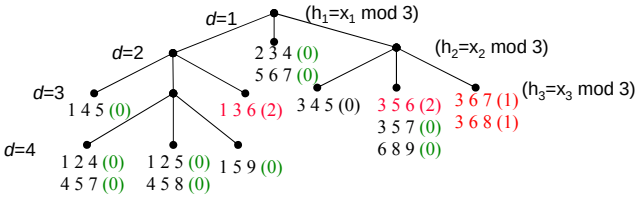
25

Scan DB to Obtain the Counts of All Length-3 Candidates

- Assume a transaction DB:

{1,3,5,6}
{2,3,5}
{1,3,6,7,8}
{3,5,6}
.....

- The counts of all the candidates in the hash tree are initialized to zero.
- Apply the subset function on each transaction and increase the count of a candidate if the transaction contains the candidate



26

How to Mine Association Rules

- A two-step process:
  - *Find all frequent itemsets* ---- *done*
  - Generate strong association rules from frequent itemsets.
- Example: given min_sup=50% and min_conf=50%

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

  - Generate strong rules:
    - {A} → {C} [support=50%, confidence=66.6%]
    - {C} → {A} [support=50%, confidence=100%]

27

Generate Association Rules from Frequent Itemsets

- Naïve algorithm:

for each frequent itemset *l* whose length ≥ 2 do
  for each **nonempty proper** subset *s* of *l* do
    if (support(*l*)/support(*s*) >= min_conf)
      output the rule $s \rightarrow l\text{-}s$, with
        support =support(l) and
        confidence=support(l)/support(s)

Note that we only need to check the confidence.
Do we need to scan the database? No. Why?

28

Generate Association Rules from Frequent Itemsets

- Example:
  - Given a frequent itemset: $l=\{A, B, C\}$
  - nonempty proper subsets of l are $\{A,B\}$, $\{A,C\}$, $\{B,C\}$, $\{A\}$, $\{B\}$, $\{C\}$
  - resulting association rules:

$$\{A,B\} \rightarrow \{C\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{A,B\})} = 50\%$$

$$\{A,C\} \rightarrow \{B\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{A,C\})} = 100\%$$

$$\{B,C\} \rightarrow \{A\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{B,C\})} = 100\%$$

$$\{A\} \rightarrow \{B,C\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{A\})} = 33\%$$

$$\{B\} \rightarrow \{A,C\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{B\})} = 29\%$$

$$\{C\} \rightarrow \{A,B\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{C\})} = 100\%$$

These confidence values are make-up numbers. This example is not continued previous examples.

If minimum confidence threshold is 70%, only 3 rules are outputted.

29

Is Apriori Fast Enough? — Performance Bottlenecks

- The core of the Apriori algorithm:
  - Use frequent *k* -itemsets to generate candidate (*k+1)*-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets to generate frequent (k+1)-itemsets from k+1 candidate set
- The bottleneck of *Apriori*: Candidate generation and testing
  - Huge candidate sets:
    - 10^4 frequent 1-itemsets will generate more than 10^7 candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate at least $2^{100} \approx 10^{30}$ candidates.
  - Multiple scans of database:
    - Needs *n* or *n+1* scans, *n* is the length of the longest frequent pattern

30