6.8 Mining frequent patterns and Association Rules :-

Frequent Patterns are patterns (such as itemsets, subsequences or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset.

- A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a frequent sequential pattern.
- A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences.
- If a substructure occurs frequently, it is called a frequent structured pattern.
- This chapter is dedicated to methods of frequent itemset mining.
- How can we find frequent itemset from large amount of data, where the data are either transactional or relational?
- How can we mine association rules in multilevel and multidimensional space?
- Which association rules are the most interesting?
- How can we help or guide the mining procedure to discover interesting associations or correlations?
- How can we take advantage of user preferences or constraints to speed up the mining process?

Market Basket Analysis : A motivational Example:

- Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets.
- A typical example of frequent itemset mining is market basket analysis.
- This process analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets".

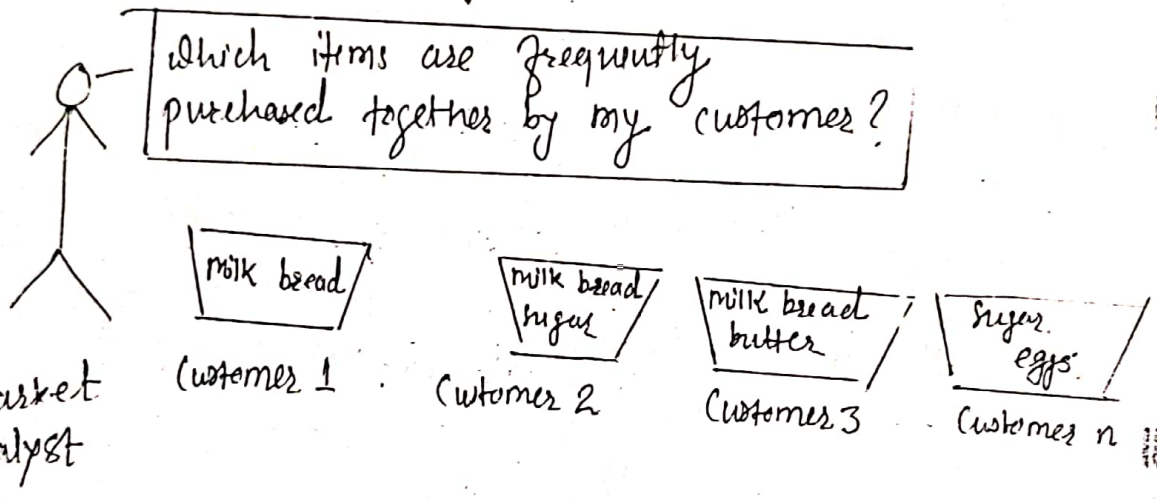


Fig. Market Basket Analysis.

- The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers.
- For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket?
- Such information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.

Market Basket Analysis :-

Suppose, as manager of an AllElectronics branch, you would like to learn more about the buying habits of your customers. Specifically, "which groups or sets of items are customers likely to purchase on a given trip to the store?

- Market Basket analysis may be performed on the retail data of customer transactions at your store.
- You can then use the results to plan marketing or advertising strategies or in the design of a new catalog.
- For instance, market basket analysis may help you design different store layouts.
- In one strategy, items that are frequently purchased together can be placed in proximity in order to further encourage the sale of such items together.
- If customers who purchase computers also tend to buy antivirus software at the same time, then placing the hardware display close to the software display may help increase the sales of both items.
- Market basket analysis can also help retailers plan which items to put on the sale at reduced prices.
- If customers tend to purchase computers and printers together, then having a sale on printers may encourage the sale of printers as well as computers.
- If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item.
- Each basket can then be represented by a Boolean vector of values assigned to these variables.
- The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently associated or purchased together.

- These patterns can be represented in the form of association rules.

- For example,
  - The information that customers who purchase computers also tend to buy antivirus software at the same time is represented in Association Rule as follows:

(1) Computer ⇒ antivirus_software [support = 2%, confidence = 60%]

- Rule support and confidence are two measures of rule interestingness.
- They respectively reflect the usefulness and certainty of discovered rules.
- A support of 2% for Association rule (1) means that 2% of all the transactions under analysis show that computer and antivirus software purchased together.
- A confidence of 60% means that 60% of customers who purchased a computer also bought the antivirus_software.
- Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold.
- Such thresholds can be set by users or domain experts.

Frequent Itemsets ; Closed Itemsets and Association Rules ⇒

- Let $I = \{I_1, I_2, \ldots I_m\}$ be a set of items.
- D - the task relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$.
- Let A be a set of items.
- A Transaction T is said to contain A if and only if $A \subseteq T$.
- An association rule is an implication of the form $A \Rightarrow B$.

$A \subset I$, $B \subset I$ and $A \cap B = \phi$

rule $A \Rightarrow B$ holds in the transaction set D with support s, where s is the percentage of transactions in D that contains $A \cup B$.

This is taken to be the probability, $P(A \cup B)$.

The rule $A \Rightarrow B$ has confidence c in the transaction set D, where c is the percentage of transactions in D containing A that also contain B.

This is taken to be the conditional probability $P(B|A)$. That is,

$$\text{Support}(A \Rightarrow B) = P(A \cup B) \quad \text{— (2)}$$

$$\text{confidence}(A \Rightarrow B) = P(B|A) \quad \text{— (3)}$$

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{Support}(A \cup B)}{\text{Support}(A)} = \frac{\text{support\_count}(A \cup B)}{\text{support\_count}(A)}$$

- In general, association rule mining can be viewed as a two-step process:

1. Find all frequent itemset: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min_sup.
2. Generate strong association rules from the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence.

The Apriori Algorithm : Finding Frequent Itemsets Using Candidate Generation :-

- Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules.
- The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties
- Apriori employs an iterative approach known as a level-wise search.
- First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support.
- The resulting set is denoted L_1.
- Next L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3 and so on, until no more frequent k-itemsets can be found.

The finding of each L_k requires one full scan of the database.

Apriori property : ALL nonempty subsets of a frequent itemset must also be frequent.

- How is the Apriori property used in the algorithm?
- How L_{k-1} is used for find L_k for $k \geq 2$.
- A two step process is followed, consisting of join and prune actions.

1) The join step :- To find L_k, a set of candidate k-itemsets is generated by joining L_{k-1} with itself. This set of candidates denoted by C_k.

2) The prune step :- C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k.

Transactional data for an AllElectronics branch.

TID	List of item IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

- There are nine transactions in this database. That is $|D| = 9$
- Suppose that the minimum support count is 2.
- Here we are illustrating the Apriori Algorithm for finding frequent itemsets in D.

C1

Scan D for count of each candidate →

Itemset	Support Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Compare candidate support count with minimum support count →

L1

Itemset	Sup. Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

I1 I2
I1 I3
I1 I4
I1 I5
I2 I3
I2 I4
I2 I5
I3 I4
I2 I5

I4 I5

$C_2 = L_1 \bowtie L_1$

Generate C_2 candidates from L_1 →

Itemset
$\{I_1, I_2\}$
$\{I_1, I_3\}$
$\{I_1, I_4\}$
$\{I_1, I_5\}$
$\{I_2, I_3\}$
$\{I_2, I_4\}$
$\{I_2, I_5\}$
$\{I_3, I_4\}$
$\{I_3, I_5\}$
$\{I_4, I_5\}$

Scan D for count of each candidate →

C_2

Itemset	Sup. Count
$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_4\}$	1
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2
$\{I_3, I_4\}$	0
$\{I_3, I_5\}$	1
$\{I_4, I_5\}$	0

Compare candidate support count with minimum support count →

L_2

Itemset	Sup. Count
$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2

Generate C_3 candidates from L_3 →

$C_3 = L_2 \bowtie L_2$

Itemset
$\{I_1, I_2, I_3\}$
$\{I_1, I_2, I_5\}$
$\{I_1, I_3, I_5\}$
$\{I_1, I_4, I_5\}$
~~$\{I_1, I_3, I_4\}$~~
$\{I_2, I_3, I_4\}$
$\{I_2, I_3, I_5\}$
$\{I_2, I_4, I_5\}$
~~$\{I_1, I_4, I_5\}$~~

$I_1\ I_2\ I_3$
$I_1\ I_2\ I_5$
$I_1\ I_2\ I_4$
$I_1\ I_3\ I_5$
$I_1\ I_4\ I_5$
$I_2\ I_3\ I_4$
$I_2\ I_3\ I_5$

Scan D for count of each candidate →

C_3

Itemset	Sup. count
$\{I_1, I_2, I_3\}$	2
$\{I_1, I_2, I_5\}$	2

Compare candidate support count with min. support count →

L_3

Itemset	Sup count
$\{I_1, I_2, I_3$	2
$\{I_1, I_2, I_5\}$	2

C_4

Itemset	Sup count
$I_1\ I_2\ I_3\ I_5$	1

Compare →

104

algorithm uses $L_3 \bowtie L_3$ to generate a candidate set of 5 itemsets C_4.

Although the join results in $\{I_1, I_2, I_3, I_5\}$, this itemset is pruned because its subset $\{I_2, I_3, I_5\}$ is not frequent. Thus $C_4 = \phi$, and the algorithm terminates, having found all of the frequent itemsets.

Algorithm: Apriori. Find frequent itemset using an iterative level wise approach based on candidate generation.

Input: D a database of transactions;

min_sup, the minimum support count threshold.

output: L frequent itemsets in D.

Method:

```
1) L1 = find_frequent_1-itemsets(D);
2) for (k=2; Lk-1 ≠ φ; k++) {
3)     Ck = apriori_gen(Lk-1);
4)     for each transaction t ∈ D {
5)         Ct = subset(Ck, t);
6)         for each candidate c ∈ Ct
7)             c.count++;
8)     }
9)     Lk = {c ∈ Ck | c.count >= min_sup}
10) }
11) return L = ∪k Lk;

procedure apriori_gen(Lk-1 : frequent (k-1)-itemsets)
1) for each itemset l1 ∈ Lk-1
2)     for each itemset l2 ∈ Lk-1
3)         if (l1[1] = l2[1] ∧ l1[2] = l2[2] ∧ ... ∧ (l1[k-2] = l2[k-2] ∧ (l1[k-1]
               < l2[k-1] then {
4)             c = l1 ⋈ l2;
```

```
5)      if has_infrequent_subset (c, Lk-1) then
6)          delete c; //prune step, remove unfruitful candidate
7)      else add c to Ck;
8)  }
9) return Ck;
procedure has_infrequent_subset (c: candidate k-itemset;
        Lk-1 : frequent (k-1)-itemsets); //use prior knowledge
1) for each (k-1)-subset s of c
2)   if s ∉ Lk-1 then
3)       return TRUE;
4)  return FALSE;
```

⇒ Generating Association Rules from frequent itemsets ⇒

- Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate a strong association rules from them.
- where strong association rules satisfy both minimum support and minimum confidence.

$$\text{Confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{Support\_count}(A \cup B)}{\text{Support\_count}(A)}$$

→ for each frequent itemset l, generate all nonempty subset of l.

→ for every nonempty subset s of l, output the rule.

$s \Rightarrow (l-s)$ if $\frac{\text{Support\_count}(l)}{\text{Support\_count}(s)} \geq$ min. conf

→ for $l = \{I_1, I_2, I_5\}$ from above example

→ what are the association rules that can be generated from l?

... nonempty subsets of I as {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}.
The resulting association rules are,

$I_1 \wedge I_2 \Rightarrow$ confidence $= 2/4 = 50\%$

$I_1 \wedge I_5 \Rightarrow I_2$ confidence $= \frac{\text{occurrence in D } (I_1 \wedge I_5 \cup I_2)}{\text{occurrence in D } (I_1 \wedge I_5)} = \frac{2}{2} = 100\%$

$I_2 \wedge I_5 \Rightarrow I_1$ confidence $= \frac{2}{2} = 100\%$

$I_1 \Rightarrow I_2 \wedge I_5$ confidence $= \frac{2}{6} = 33\%$

$I_2 \Rightarrow I_1 \wedge I_5$ confidence $= 2/7 = 29\%$

$I_5 \Rightarrow I_1 \wedge I_2$ confidence $= \frac{2}{2} = 100$

⇒ If the minimum confidence threshold is, say 70%
⇒ then the second, third and last rules above are output, because these are the only ones generated that are strong.

Q:) FP growth (finding frequent itemsets without candidate generation) ⇒

- However, Apriori algorithm suffer from two non-trivial costs
  - ⇒ It may need to generate a huge number of candidate sets.
  - ⇒ It may need to repeatedly scan the database and check a large set of candidates by pattern matching.
- frequent-pattern growth mines the complete set of frequent itemsets without candidate generation.
- FP-growth adopts a divide-and-conquer strategy.
- First it compresses the database representing frequent items into a frequent-pattern tree, as FP-tree, which retains the itemset association information.
- It then divides the compressed database into a set of conditional databases, each associated with one frequent item or "pattern fragment", and mines each such database seperately.

Example:- A given Transactional Data for an ALL Electronics Branch

TID	List of Items-IDS
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Solution ⇒ 1) frequency of items

Items	Support count
I1	6
I2	7
I3	6
I4	2
I5	2

2) Set of frequent items is sorted in the order of descending support count. (minimum support is 2).

Item	support count
I_2	7
I_1	6
I_5	6
I_4	2
I_3	2

3) Arrange According to priority

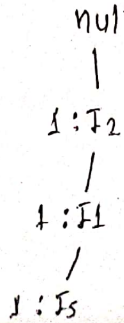
TID	List of Items
T_{100}	I_2, I_1, I_5
T_{200}	I_2, I_4
T_{300}	I_2, I_3
T_{400}	I_2, I_1, I_4
T_{500}	I_1, I_3
T_{600}	I_2, I_3
T_{700}	I_1, I_3
T_{800}	I_2, I_1, I_3, I_5
T_{900}	I_2, I_1, I_3

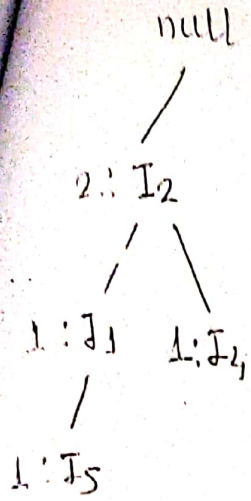
4) FP-tree is then constructed

- first create the root of the tree, labeled with "null".
- Scan database D a second time.

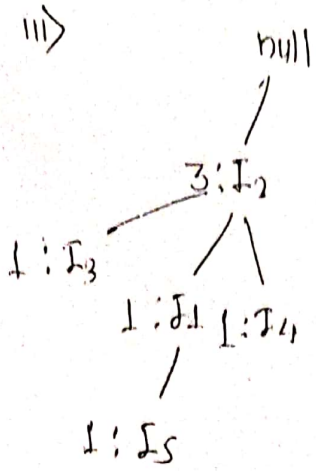
"null"

- Scan the first transaction i.e. (1).

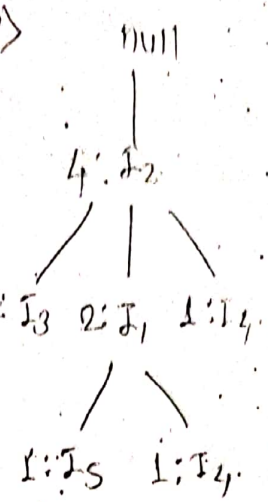




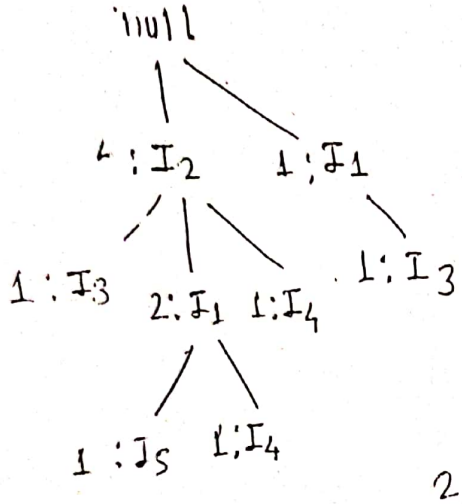
iii>



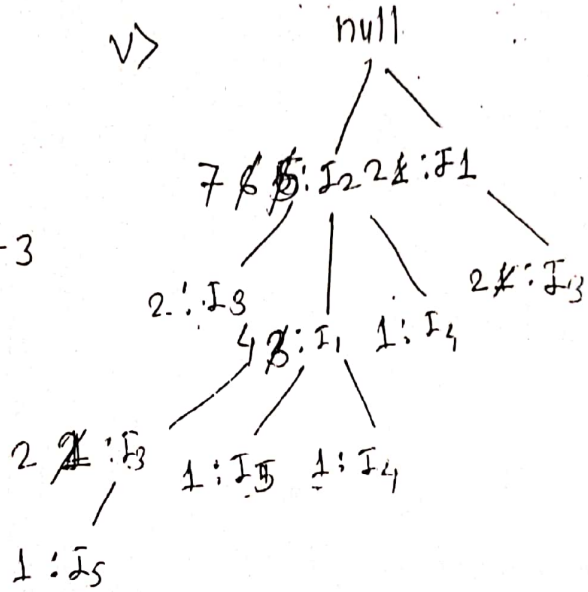
iv>



v>



v>



Item	Conditional Pattern base	Conditional FP tree	Frequent Pattern Generated
I_5	$\{\{I_2, I_1 : 1\}, \{I_2, I_1, I_3 : 1\}\}$	$\langle I_2 : 2, I_1 : 2 \rangle$	$\{I_2, I_5 : 2\}, \{I_1, I_5 : 2\}$, $\{I_2, I_1, I_5 : 2\}$
I_4	$\{\{I_2, I_1 : 1\}, \{I_2 : 1\}\}$	$\langle I_2 : 2 \rangle$	$\{I_2, I_4 : 2\}$
I_3	$\{\{I_2, I_1 : 2\}, \{I_2 : 2\}, \{I_1 : 2\}\}$	$\langle I_2 : 4, I_1 : 2 \rangle$ $\langle I_1 : 2 \rangle$	$\{I_2, I_3 : 4\}, \{I_1, I_3 : 2\}$, $\{I_2, I_1, I_3 : 2\}$
I_1	$\{I_2 : 4\}$	$\langle I_2 : 4 \rangle$	$\{I_2, I_4 : 4\}$

Final Answer : Frequent Items-sets $\{I_1, I_2, I_3 : 2\}$
$\{I_1, I_2, I_5 : 2\}$

Algorithm : FP-growth

Input : D, a transaction database;
min_sup, the minimum support count threshold.

output : The complete set of frequent patterns.

Method :

1. The FP-tree is constructed in the following steps:
   a) Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the list of frequent items.
   b) Create the root of an FP-tree, and label it as "null"
      - Sort the frequent items in Trans according to the order of L.
      - Let the sorted frequent items list in Trans be [p | P], where p is the first element and P is the remaining list.
      - call insert_tree([p|P], T).
      - if T has a child N such that N.item_name = p.item_name, then increment N's count by 1. else create a new node N and let its count be 1, its parent link be linked to T, and its node link to the nodes with the same item_name via the node_link structure.
      - if P is nonempty, call insert_tree(P, N) recursively.
2. The FP-tree is mined by calling FP_growth(FP_tree, null)

procedure FP_growth(Tree, α)
a) if Tree contains a single path P then
b) for each combination (denoted as β) of the nodes in the path P
c) generate pattern β ∪ α with support_count = minimum support count of nodes in β;
d) else for each a_i in the headers of Tree {
e) generate pattern β = a_i ∪ α with support_count = a_i.support_count;
f) Construct β's conditional pattern base and then β's conditional FP_tree Tree_β;
g) if Tree_β ≠ ∅ then
h) call FP_growth(Tree_β, β); }