UNIT-3

Mining frequent patterns, Associations and Correlations- Basic Concepts, Frequent itemset Mining methods- Apriori Algorithm, Generating association rules from frequent itemsets, improving the efficiency of Apriori, A pattern growth approach for mining frequent itemsets. Which patterns are interesting- pattern evaluation methods

Basic Concepts:

- **Market Basket Analysis: A Motivating Example**
- **Frequent Itemsets, Closed Itemsets, and Association Rules**

Frequent patterns are patterns (e.g., itemsets, subsequences, or substructures) that appear frequently in a data set.

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.

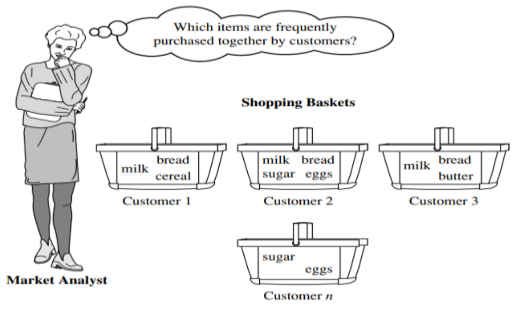
Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks.

Market Basket Analysis: A Motivating Example

Frequent item set mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. With massive amounts of data continuously being collected and stored, many industries are becoming interested in mining such patterns from their databases. The discovery of interesting correlation relationships among huge amounts of business transaction records can help in many business decision-making processes such as catalog design, cross-marketing, and customer shopping behavior analysis.

An 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". The discovery of these 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? This information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.



Market basket analysis

Let's look at an example of how market basket analysis can be useful.

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

To answer your question, 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 to further encourage the combined sale of such items. 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.

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 the following **association rule:**

computer ⍰ antivirus software [support = 2%,confidence = 60%]

Rule support and confidence are two measures of rule interestingness.

A support of 2% for Rule means that 2% of all the transactions under analysis show that computer and antivirus software are purchased together.

A confidence of 60% means that 60% of the customers who purchased a computer also bought the software.

Association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. These thresholds can be a set by users or domain experts.

Frequent Itemsets, Closed Itemsets, and Association Rules:

Each transaction is associated with an identifier, called a TID. Let A be a set of items. A transaction T is said to contain A if $A \subseteq T$.

An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, $A \neq \emptyset$, $B \neq \emptyset$, and $A \cap B = \varphi$. The rule $A \Rightarrow B$ holds in the transaction set D with support s, where s is the percentage of transactions in D that contain $A \cup B$ (i.e., the union of sets A and B say, or, both A and 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

$\text{support}(A \Rightarrow B) = P(A \cup B)$

$\text{confidence}(A \Rightarrow B) = P(B|A)$.

Rules that satisfy both a minimum support threshold (min sup) and a minimum confidence threshold (min conf) are called strong. support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0.

If the relative support of an itemset I satisfies a prespecified minimum support threshold (i.e., the absolute support of I satisfies the corresponding minimum support count threshold), then I is a frequent itemset.

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

A set of items is referred to as an **itemset.**

An itemset that contains k items is a k-itemset. The set {computer, antivirus software} is a 2-itemset. The occurrence frequency of an itemset is the number of transactions that contain the itemset. This is also known, simply, as the frequency, support count, or count of the itemset

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

1. Find all frequent itemsets: 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.

An itemset X is closed frequent Itemset only if

- X is frequent
- No immediate superset of X has sam support of X.

An itemset X is a **maximal frequent itemset** (or max-itemset) in a data set D if X is frequent, and there exists no super-itemset Y such that X ⊂ Y and Y is frequent in D.

Frequent Itemset Mining Methods:

- **Apriori Algorithm**: Finding Frequent Itemsets by Confined Candidate Generation
- **A Pattern-Growth Approach** for Mining Frequent Itemsets OR **FP-growth** (finding frequent itemsets without candidate generation).

Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

- Generating Association Rules from Frequent Itemsets
- Improving the Efficiency of Apriori

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, where k-itemsets are used to explore (k + 1)-itemsets.

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 by L1.

Next, L1 is used to find L2, the set of frequent 2-itemsets, which is used to find L3, and so on, until no more frequent k-itemsets can be found. The finding of each Lk requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property is used to reduce the search space.

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

The Apriori property is based on the following observation. By definition, if an itemset I does not satisfy the minimum support threshold, min sup, then I is not frequent, that is, P(I) < min sup. If an item A is added to the itemset I, then the resulting itemset (i.e., I ∪A) cannot occur more frequently than I. Therefore, I ∪A is not frequent either, that is, P(I ∪A) < min sup.

This property belongs to a special category of properties called **antimonotonicity** in the sense that if a set cannot pass a test, all of its supersets will fail the same test as well.

"How is the Apriori property used in the algorithm?"

To understand this, let us look at how Lk−1 is used to find Lk for k ≥ 2.

A two-step process is followed, consisting of join and prune actions.

1. **The join step:** This step generates (K+1) item set from K-itesets by joining each item with itself. To find L_k , a set of candidate k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted Ck . Let l1 and l2 be itemsets in L_{k-1}.
2. **The Prune step:** This step scans the count of each item in the database. If the candidate item does not meet minimum support then it is regarded as infrequent and thus it is removed. This step is performed to reduce the size of the candidate itemsets.

Example:

There are nine transactions in this database, that is, |D| = 9 i.e Transactional Data for an AllElectronics Branch. illustrate the Apriori algorithm for finding frequent itemsets in D**.** min sup = 2, minimum confidence threshold is, say, 70%.

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C1. The algorithm simply scans all of the transactions to count the number of occurrences of each item.

2. Suppose that the minimum support count required is 2, that is, min sup = 2. (The corresponding relative support is 2/9 = 22%.) The set of frequent 1-itemsets, L1, can then be determined. It consists of

the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in C1 satisfy minimum support.

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

Table 6.1:Transactional Data for an AllElectronics Branch

4.To discover the set of frequent 2-itemsets, L2, the algorithm uses the join L1 ⋈ L1 to generate a candidate set of 2-itemsets, C2.

5.The set of frequent 2-itemsets, L2, is then determined, consisting of those candidate 2-itemsets in C2 having minimum support.

6.Generation and pruning of candidate 3-itemsets, C3, from L2 using the Apriori property:

The generation of the set of the candidate 3-itemsets, C3. From the join step, we first get

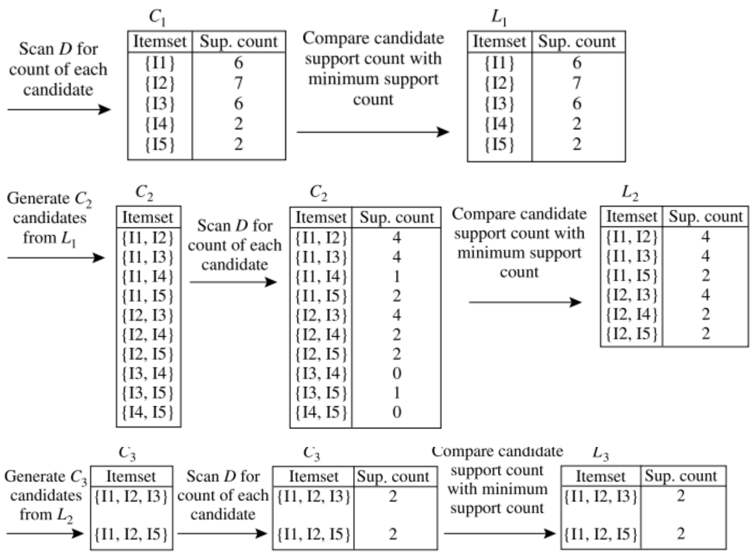
C3 = L2 ⋈ L2 = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}. Based on the Apriori property that all subsets of a frequent itemset must also be frequent

(a) Join: $C_3 = L_2 \bowtie L_2 = \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$\bowtie \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$= \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}.$

(b) Prune using the Apriori property: All nonempty subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?

- The 2-item subsets of {I1, I2, I3} are {I1, I2}, {I1, I3}, and {I2, I3}. All 2-item subsets of {I1, I2, I3} are members of L2. Therefore, keep {I1, I2, I3} in C3.
- The 2-item subsets of {I1, I2, I5} are {I1, I2}, {I1, I5}, and {I2, I5}. All 2-item subsets of {I1, I2, I5} are members of L2. Therefore, keep {I1, I2, I5} in C3., etc

(c) Therefore, C3 = {{I1, I2, I3}, {I1, I2, I5}} after pruning**.**


Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_2 candidates from L_1
Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_3 candidates from L_2
Scan D for count of each candidate
Compare candidate support count with minimum support count

C_1

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

L_1

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

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Generation of the candidate itemsets and frequent itemsets, where the minimum support count is 2.

7.The transactions in D are scanned to determine L3, consisting of those candidate 3-itemsets in C3 having minimum support.

8. The algorithm uses L3 ⋈ L3 to generate a candidate set of 4-itemsets, C4. Although the join results in {{I1, I2, I3, I5}}, itemset {I1, I2, I3, I5} is pruned because its subset {I2, I3, I5} is not frequent. Thus, C4 = φ, and the algorithm terminates, having found all of the frequent itemsets.

Pseudocode for the Apriori algorithm:

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approacl on candidate generation.

Input:

- D, a database of transactions;
- min_sup, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)    $L_1$ = find_frequent_1-itemsets(D);
(2)    for ($k = 2; L_{k-1} \neq \phi; k{+}{+}$) {
(3)        $C_k$ = apriori_gen($L_{k-1}$);
(4)        for each transaction $t \in D$ { // scan $D$ for counts
(5)            $C_t$ = subset($C_k$, $t$); // get the subsets of $t$ that are candidates
(6)            for each candidate $c \in C_t$
(7)                c.count++;
(8)        }
(9)        $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10)   }
(11)   return $L = \cup_k L_k$;

procedure apriori_gen($L_{k-1}$:frequent $(k-1)$-itemsets)
(1)    for each itemset $l_1 \in L_{k-1}$
(2)        for each itemset $l_2 \in L_{k-1}$
(3)            if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2])$
                   $\wedge ... \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ then {
(4)                $c = l_1 \bowtie l_2$; // join step: generate candidates
(5)                if has_infrequent_subset($c$, $L_{k-1}$) then
(6)                    delete $c$; // prune step: remove unfruitful candidate
(7)                else add $c$ to $C_k$;
(8)            }
(9)    return $C_k$;

procedure has_infrequent_subset($c$: candidate $k$-itemset;
           $L_{k-1}$: frequent $(k-1)$-itemsets); // use prior knowledge
(1)    for each $(k-1)$-subset $s$ of $c$
(2)        if $s \notin L_{k-1}$ 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 strong association rules from them (where strong association rules satisfy both minimum support and minimum confidence).

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

The conditional probability is expressed in terms of itemset support count, where support count(A ∪B) is the number of transactions containing the itemsets A ∪B, and support count(A) is the number of transactions containing the itemset A.

Based on this equation, association rules can be generated as follows:

- For each frequent itemset l, generate all nonempty subsets of l.
- For every nonempty subset s of l, output the rule "s ⇒ (l − s)"
  if support count(l)⁄ support count(s) ≥ min conf, where min conf is the minimum confidence threshold.

Example:

Generating association rules. Let's try an example based on the transactional data for AllElectronics shown before in Table 6.1. The data contain frequent itemset X = {I1, I2, I5}. What are the association rules that can be generated from X? The nonempty subsets of X are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}.

The resulting association rules are as shown below, each listed with its confidence:

{I1, I2} ⇒ I5, confidence = 2/4 = 50%

{I1, I5} ⇒ I2, confidence = 2/2 = 100%

{I2, I5} ⇒ I1, confidence = 2/2 = 100%

I1 ⇒ {I2, I5}, confidence = 2/6 = 33%

I2 ⇒ {I1, I5}, confidence = 2/7 = 29%

I5 ⇒ {I1, I2}, confidence = 2/2 = 100%

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

Improving the Efficiency of Apriori:

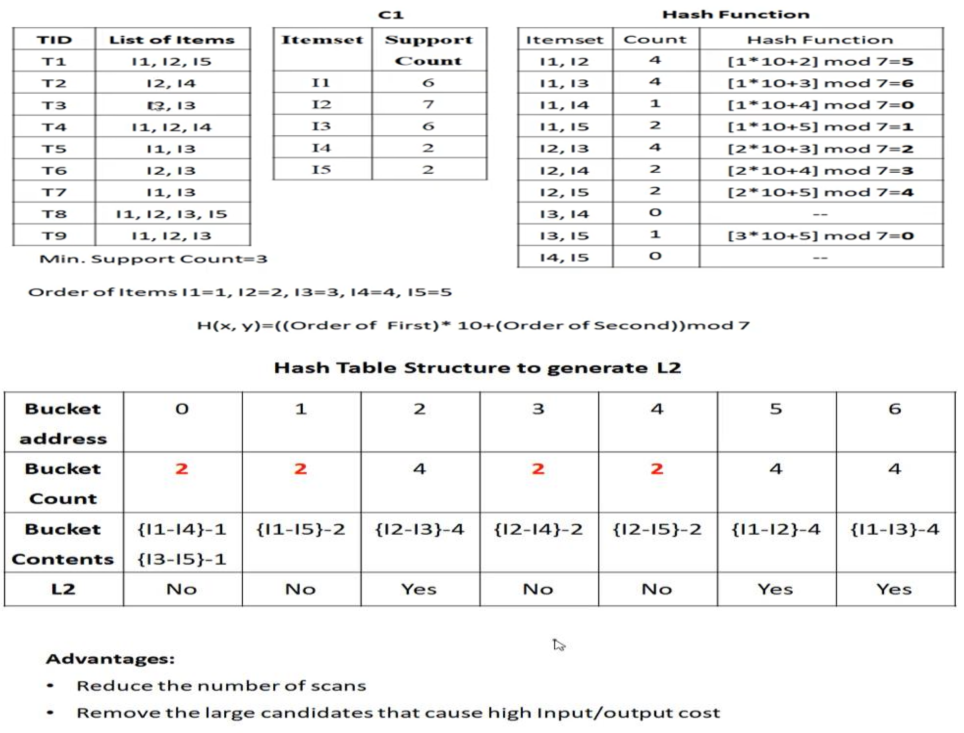
"How can we further improve the efficiency of Apriori-based mining?" Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm. Several of these variations are summarized as follows

Hash-based technique (hashing itemsets into corresponding buckets):

A hash-based technique can be used to reduce the size of the candidate k-itemsets, Ck , for k > 1.

- For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L1, we can generate all the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a hash table structure, and increase the corresponding bucket counts (Figure 6.5).
- A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set.
- Such a hash-based technique may substantially reduce the number of candidate k-itemsets examined (especially when k = 2).

Example:



TID	List of Items
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

Min. Support Count=3

C1

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

Hash Function

Itemset	Count	Hash Function
I1, I2	4	[1*10+2] mod 7=**5**
I1, I3	4	[1*10+3] mod 7=**6**
I1, I4	1	[1*10+4] mod 7=**0**
I1, I5	2	[1*10+5] mod 7=**1**
I2, I3	4	[2*10+3] mod 7=**2**
I2, I4	2	[2*10+4] mod 7=**3**
I2, I5	2	[2*10+5] mod 7=**4**
I3, I4	0	--
I3, I5	1	[3*10+5] mod 7=**0**
I4, I5	0	--

Order of Items I1=1, I2=2, I3=3, I4=4, I5=5

H(x, y)=((Order of First)* 10+(Order of Second))mod 7

Hash Table Structure to generate L2

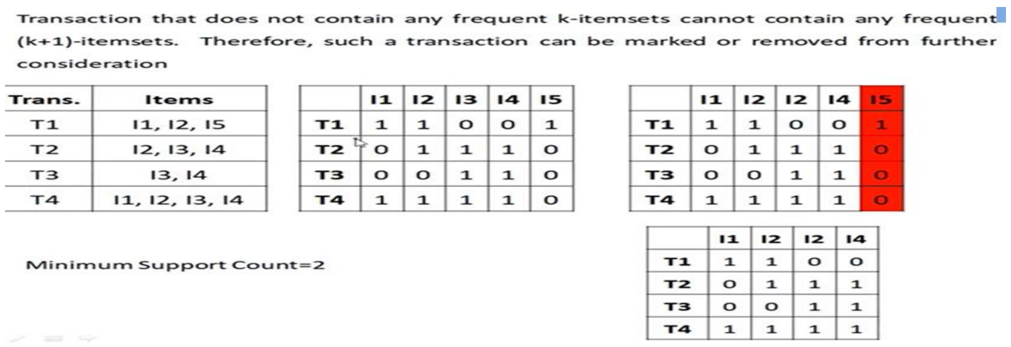
Bucket address	0	1	2	3	4	5	6
Bucket Count	2	2	4	2	2	4	4
Bucket Contents	{I1-I4}-1 {I3-I5}-1	{I1-I5}-2	{I2-I3}-4	{I2-I4}-2	{I2-I5}-2	{I1-I2}-4	{I1-I3}-4
L2	No	No	Yes	No	No	Yes	Yes

Advantages:

- Reduce the number of scans
- Remove the large candidates that cause high Input/output cost

Transaction reduction (reducing the number of transactions scanned in future iterations):

- A transaction that does not contain any frequent k-item sets cannot contain any frequent (k + 1)-item sets.
- Therefore, such a transaction can be marked or removed from further consideration because subsequent database scans for j-item sets, where j > k, will not need to consider such a transaction.



Transaction that does not contain any frequent k-itemsets cannot contain any frequent (k+1)-itemsets. Therefore, such a transaction can be marked or removed from further consideration

Trans.	Items
T1	I1, I2, I5
T2	I2, I3, I4
T3	I3, I4
T4	I1, I2, I3, I4

	I1	I2	I3	I4	I5
T1	1	1	0	0	1
T2	0	1	1	1	0
T3	0	0	1	1	0
T4	1	1	1	1	0

	I1	I2	I2	I4	I5
T1	1	1	0	0	1
T2	0	1	1	1	0
T3	0	0	1	1	0
T4	1	1	1	1	0

Minimum Support Count=2

	I1	I2	I2	I4
T1	1	1	0	0
T2	0	1	1	1
T3	0	0	1	1
T4	1	1	1	1

Partitioning (partitioning the data to find candidate item sets):

- A partitioning technique can be used that requires just two database scans to mine the frequent item sets.
- It consists of two phases.
- In phase I, the algorithm divides the transactions of D into n non overlapping partitions.
- In phase II a second scan of D is conducted in which the actual support of each candidate is assessed to determine the global frequent itemsets.

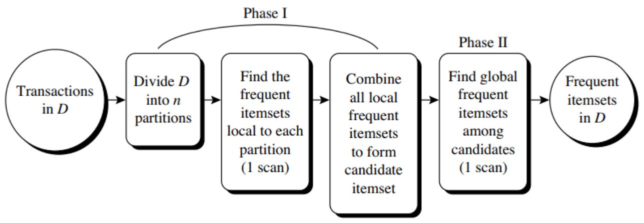
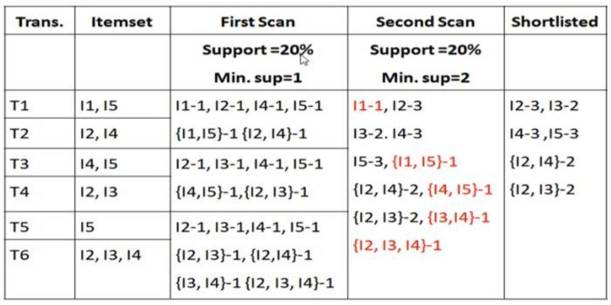

Phase I
Transactions in D
Divide D into n partitions
Find the frequent itemsets local to each partition (1 scan)
Combine all local frequent itemsets to form candidate itemset
Phase II
Find global frequent itemsets among candidates (1 scan)
Frequent itemsets in D

Fig: Mining by partitioning the data.

Example:



Trans.	Itemset	First Scan	Second Scan	Shortlisted
		Support =20% Min. sup=1	Support =20% Min. sup=2	
T1	I1, I5	I1-1, I2-1, I4-1, I5-1	I1-1, I2-3	I2-3, I3-2
T2	I2, I4	{I1,I5}-1 {I2, I4}-1	I3-2. I4-3	I4-3 ,I5-3
T3	I4, I5	I2-1, I3-1, I4-1, I5-1	I5-3, {I1, I5}-1	{I2, I4}-2
T4	I2, I3	{I4,I5}-1,{I2, I3}-1	{I2, I4}-2, {I4, I5}-1	{I2, I3}-2
T5	I5	I2-1, I3-1,I4-1, I5-1	{I2, I3}-2, {I3,I4}-1	
T6	I2, I3, I4	{I2, I3}-1, {I2,I4}-1 {I3, I4}-1 {I2, I3, I4}-1	{I2, I3, I4}-1	

Sampling (mining on a subset of the given data):

- The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D.
- In this way, we trade off some degree of accuracy against efficiency.
- The S sample size is such that the search for frequent itemsets in S can be done in main memory, and so only one scan of the transactions in S is required overall.
- Because we are searching for frequent itemsets in S rather than in D, it is possible that we will miss some of the global frequent itemsets.

Dynamic itemset counting (adding candidate itemsets at different points during a scan):

- A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points.
- In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan.
- The technique uses the count-so-far as the lower bound of the actual count.
- If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates. This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

Apriori suffer from two nontrivial costs:

- It may still need to generate a huge number of candidate sets. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets.
- It may need to repeatedly scan the whole database and check a large set of candidates by pattern matching. It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

A Pattern-Growth Approach for Mining Frequent Itemsets:

"Can we design a method that mines the complete set of frequent itemsets without such a costly candidate generation process?"

- The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations, thus improving performance.
- It uses a divide-and-conquer strategy.
- The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information**.**

This algorithm works as follows:

- First, it compresses the input database creating an FP-tree instance to represent frequent items.
- After this first step, it divides the compressed database into a set of conditional databases, each associated with one frequent pattern.
- Finally, each such database is mined separately

The frequent-pattern tree (FP-tree) is a compact data structure that stores quantitative information about frequent patterns in a database. Each transaction is read and then mapped onto a path in the FP-tree.

A frequent Pattern Tree is made with the initial item sets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the item set.

The root node represents null, while the lower nodes represent the item sets. The associations of the nodes with the lower nodes, that is, the item sets with the other item sets, are maintained while forming the tree.

Example: FP-growth (finding frequent itemsets without candidate generation).

We reexamine the mining of transaction database, D, of Table 6.1 in Example 6.3 using the frequent pattern growth approach**.**

The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies).

Let the minimum support count be 2.

The set of frequent items is sorted in the order of descending support count. This resulting set or list is denoted by L.

Thus, we have **L = {{I2: 7}, {I1: 6}, {I3: 6}, {I4: 2}, {I5: 2}}.**

FP-tree as the tree structure given below:

One root is labelled as "null" with a set of item-prefix subtrees as children and a frequent-item-header table.

Each node in the item-prefix subtree consists of three fields:

- Item-name: registers which item is represented by the node;
- Count: the number of transactions represented by the portion of the path reaching the node;
- Node-link: links to the next node in the FP-tree carrying the same item name or null if there is none.

Each entry in the frequent-item-header table consists of two fields:

- Item-name: as the same to the node;

Head of node-link: a pointer to the first node in the FP-tree carrying the item name

- First, create the root of the tree, labeled with "null." Scan database D a second time. The items in each transaction are processed in L order (i.e., sorted according to descending support count), and a branch is created for each transaction.
- For example, the scan of the first transaction, "T100: I1, I2, I5," which contains three items (I2, I1, I5 in L order), leads to the construction of the first branch of the tree with three nodes, hI2: 1i, hI1: 1i, and hI5: 1i, where I2 is linked as a child to the root, I1 is linked to I2, and I5 is linked to I1.
- The second transaction, T200, contains the items I2 and I4 in L order, which would result in a branch where I2 is linked to the root and I4 is linked to I2. However, this branch would share a common prefix, I2, with the existing path for T100.

In this way, the problem of mining frequent patterns in databases is transformed into that of mining the FP-tree

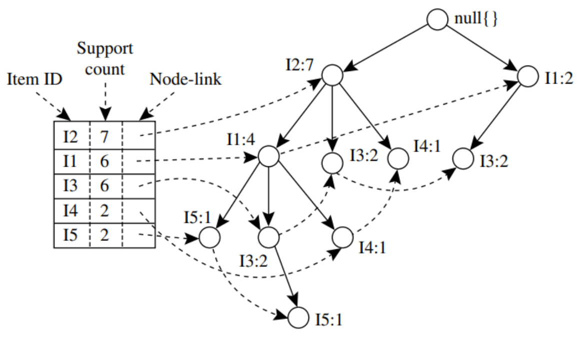


Figure 6.7 An FP-tree registers compressed, frequent pattern information.

The FP-tree is mined as follows:

Start from each frequent length-1 pattern (as an initial suffix pattern), construct its conditional pattern base (a "sub-database," which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then construct its (conditional) FP-tree, and perform mining recursively on the tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Mining of the FP-tree is summarized in Table 6.2

Table 6.2 Mining the FP-Tree by Creating Conditional (Sub-)Pattern Bases

Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns Generated*
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

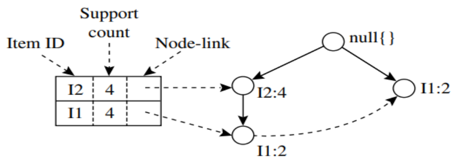
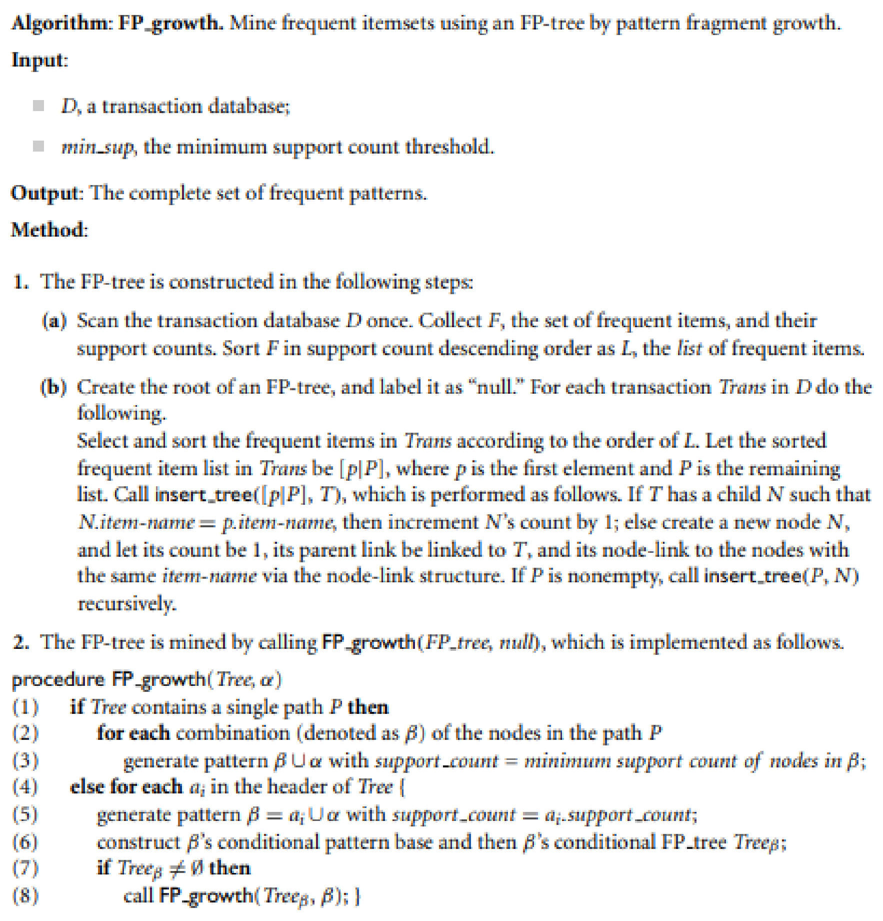


Fig: The conditional FP-tree associated with the conditional node I3.

FP-growth algorithm for discovering frequent itemsets without candidate generation:



Algorithm: FP_growth. Mine frequent itemsets using an FP-tree by pattern fragment 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." For each transaction *Trans* in *D* do the following.
Select and sort the frequent items in *Trans* according to the order of *L*. Let the sorted frequent item list in *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call insert_tree($[p|P]$, T), which is performed as follows. If T has a child N such that $N.item\text{-}name = p.item\text{-}name$, 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*), which is implemented as follows.

```
procedure FP_growth(Tree, $\alpha$)
(1)   if Tree contains a single path P then
(2)       for each combination (denoted as $\beta$) of the nodes in the path P
(3)           generate pattern $\beta \cup \alpha$ with support_count = minimum support count of nodes in $\beta$;
(4)   else for each $a_i$ in the header of Tree {
(5)       generate pattern $\beta = a_i \cup \alpha$ with support_count = $a_i$.support_count;
(6)       construct $\beta$'s conditional pattern base and then $\beta$'s conditional FP_tree $Tree_\beta$;
(7)       if $Tree_\beta \neq \emptyset$ then
(8)           call FP_growth($Tree_\beta$, $\beta$); }
```

Which Patterns Are Interesting?—Pattern Evaluation Methods:

Several metrics and criteria, including support, confidence, and lift, are used in this evaluation method to statistically evaluate the patterns. Finding patterns, trends, and correlations in data allows for the discovery of hidden information that can help with decision−making and problem−solving. An essential step in this process is pattern evaluation, which involves systematically evaluating the identified patterns to ascertain their utility, importance, and quality.

- Strong Rules Are Not Necessarily Interesting
- From Association Analysis to Correlation Analysis
- A Comparison of Pattern Evaluation Measures

Strong Rules Are Not Necessarily Interesting:

Whether or not a rule is interesting can be assessed either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting, and this judgment, being subjective, may differ from one user to another. However, objective interestingness measures, based on the statistics —behind the data.

"How can we tell which strong association rules are really interesting?" Let's examine the following example.

Example: A misleading "strong" association rule.

Suppose we are interested in analyzing transactions at AllElectronics with respect to the purchase of computer games and videos.

Let game refer to the transactions containing computer games, and video refer to those containing videos. Of the 10,000 transactions analyzed, the data show that 6000 of the customer transactions included computer games, while 7500 included videos, and 4000 included both computer games and videos.

Suppose that a data mining program for discovering association rules is run on the data, using a minimum support of, say, 30% and a minimum confidence of 60%.

The following association rule is discovered:

- buys(X, “computer games”) ⇒ buys(X, “videos”)
  [support = 40%, confidence = 66%].

The above rule is a strong association rule and would therefore be reported, since its
support value of 4000 /10,000 = 40% and
confidence value of 4000/ 6000 = 66%
satisfy the minimum support and minimum confidence thresholds, respectively.
However, Rule is misleading because the probability of purchasing videos is 75%, which is even larger than 66%.

From Association Analysis to Correlation Analysis:

As we have seen so far, the support and confidence measures are insufficient at filtering out uninteresting association rules. To tackle this weakness, a correlation measure can be used to augment the support–confidence framework for association rules.
This leads to correlation rules of the form
A ⇒ B [support, confidence, correlation]
There are many different correlation measures from which to choose.

Following are correlation measures to determine which would be good for mining large data sets:

- Lift
- χ 2 measure

Lift:

Lift is a simple correlation measure that is given as follows. The occurrence of itemset A is independent of the occurrence of itemset B if P(A ∪B) = P(A)P(B); otherwise, itemsets A and B are dependent and correlated as events. This definition can easily be extended to more than two itemsets. The lift between the occurrence of A and B can be measured by computing

$$lift(A, B) = \frac{P(A \cup B)}{P(A)P(B)}.$$

- If the resulting value above Eq. is less than 1, then the occurrence of A is negatively correlated with the occurrence of B, meaning that the occurrence of one likely leads to the absence of the other one.
- If the resulting value is greater than 1, then A and B are positively correlated, meaning that the occurrence of one implies the occurrence of the other.
- If the resulting value is equal to 1, then A and B are independent and there is no correlation between them.

Example: Correlation analysis using lift

We need to study how the two itemsets, A and B, are correlated. Let $\overline{game}$ refer to the transactions that do not contain computer games, and $\overline{video}$ refer to those that do not contain videos. The transactions can be summarized in a contingency table, as shown in Table 6.6

	game	$\overline{game}$	Σ_{row}
video	4000	3500	7500
$\overline{video}$	2000	500	2500
Σ_{col}	6000	4000	10,000

Table 6.6 2 × 2 Contingency Table

From the table, we can see that the
probability of purchasing a computer game is P({game}) = 0.60,
probability of purchasing a video is P({video}) = 0.75, and
probability of purchasing both is P({game, video}) = 0.40.

lift of Rule is

P({game, video})/(P({game}) × P({video})) = 0.40/(0.60 × 0.75) = 0.89.
Because this value is less than 1, there is a negative correlation between the occurrence of {game} and {video}.

The numerator is the likelihood of a customer purchasing both, while the denominator is what the likelihood would have been if the two purchases were completely independent.
Such a negative correlation cannot be identified by a support–confidence framework.
χ 2 measure: To compute the χ 2 value, we take the squared difference between the observed and expected value for a slot (A and B pair) in the contingency table, divided by the expected value.
This amount is summed for all slots of the contingency table.
Let's perform a χ 2 analysis
Example: Correlation analysis using χ 2 .
To compute the correlation using χ 2 analysis for nominal data, we need the observed value and expected value (displayed in parenthesis) for each slot of the contingency table, as shown in Table 6.7.
From the table, we can compute the χ 2 value as follows:

Table 6.7 Table 6.6 Contingency Table, Now with the Expected Values

	game	$\overline{game}$	Σ_{row}
video	4000 (4500)	3500 (3000)	7500
$\overline{video}$	2000 (1500)	500 (1000)	2500
Σ_{col}	6000	4000	10,000

$$\chi^2 = \Sigma\frac{(observed - expected)^2}{expected} = \frac{(4000-4500)^2}{4500} + \frac{(3500-3000)^2}{3000} + \frac{(2000-1500)^2}{1500} + \frac{(500-1000)^2}{1000} = 555.6.$$

Because the χ 2 value is greater than 1, and the observed value of the slot (game, video) = 4000, which is less than the expected value of 4500, buying game and buying video are negatively correlated
A Comparison of Pattern Evaluation Measures:
Four such measures:

- All confidence,
- Max confidence,
- Kulczynski, and cosine.

We'll then compare their effectiveness with respect to one another and with respect to the lift and χ 2 measures.

- **Given two itemsets, A and B, the all confidence measure of A and B is defined as**

$$all\_conf(A,B) = \frac{sup(A \cup B)}{max\{sup(A), sup(B)\}} = min\{P(A|B), P(B|A)\},$$

where max{sup(A), sup(B)} is the maximum support of the itemsets A and B. Thus, all conf(A,B) is also the minimum confidence of the two association rules related to A and B, namely, "A ⇒ B" and "B ⇒ A."

- **Given two itemsets, A and B, the max confidence measure of A and B is defined as**
  max conf(A, B) = max{P(A|B),P(B|A)}.

The max conf measure is the maximum confidence of the two association rules, "A ⇒ B" and "B ⇒ A."

- **Given two itemsets, A and B, the Kulczynski measure of A and B (abbreviated as Kulc) is defined as**

$$Kulc(A,B) = \frac{1}{2}(P(A|B) + P(B|A)).$$

- **Finally, given two itemsets, A and B, the cosine measure of A and B is defined as**

$$cosine(A,B) = \frac{P(A \cup B)}{\sqrt{P(A) \times P(B)}}$$

The cosine measure can be viewed as a harmonized lift measure: The two formulae are similar except that for cosine, the square root is taken on the product of the probabilities of A and B.
Each of these four measures defined has the following property: Its value is only influenced by the supports of A, B, and A ∪B, or more exactly, by the conditional probabilities of P(A|B) and P(B|A), but not by the total number of transactions. Another common property is that each measure ranges from 0 to 1, and the higher the value, the closer the relationship between A and B.
Now, together with lift and χ 2 , we have introduced in total six pattern evaluation measures. You may wonder, **"Which is the best in assessing the discovered pattern relationships?" To answer this**

question, we examine their performance on some typical data sets.
Example: Comparison of six pattern evaluation measures on typical data sets. The relationships between the purchases of two items, milk and coffee, can be examined by summarizing their purchase history in Table 6.8, a 2 × 2 contingency table, where an entry such as mc represents the number of transactions containing both milk and coffee.

Table 6.8 2 × 2 Contingency Table for Two Items

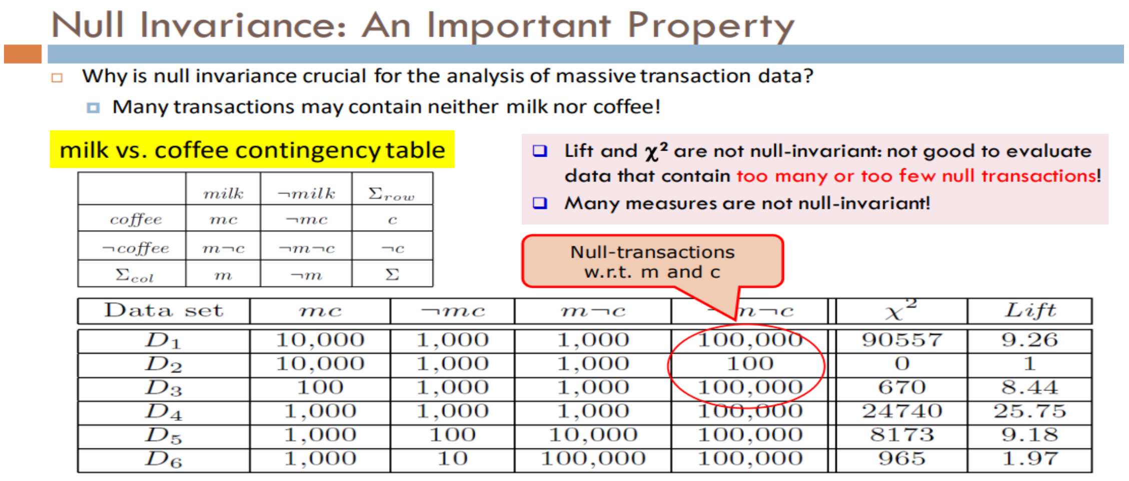
	milk	$\overline{milk}$	Σ_{row}
coffee	mc	$\overline{m}c$	c
$\overline{coffee}$	$m\overline{c}$	$\overline{mc}$	$\overline{c}$
Σ_{col}	m	$\overline{m}$	Σ

Table 6.9 shows a set of transactional data sets with their corresponding contingency tables and the associated values for each of the six evaluation measures.
Let's first examine the first four data sets, D1 through D4.
From the table, we see that m and c are positively associated in D1 and D2, negatively associated in D3, and neutral in D4. For D1 and D2, m and c are positively associated because mc (10,000) is considerably greater than mc (1000) and mc (1000).
for people who bought milk (m = 10,000 + 1000 = 11,000), it is very likely that they also bought coffee (mc/m = 10/11 = 91%), and vice versa

Table 6.9 Comparison of Six Pattern Evaluation Measures Using Contingency Tables for a Variety of Data Sets

Data Set	mc	$\overline{m}c$	$m\overline{c}$	$\overline{mc}$	χ^2	*lift*	*all_conf.*	*max_conf.*	*Kulc.*	*cosine*
D_1	10,000	1000	1000	100,000	90557	9.26	0.91	0.91	0.91	0.91
D_2	10,000	1000	1000	100	0	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100,000	670	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100,000	24740	25.75	0.5	0.5	0.5	0.5
D_5	1000	100	10,000	100,000	8173	9.18	0.09	0.91	0.5	0.29
D_6	1000	10	100,000	100,000	965	1.97	0.01	0.99	0.5	0.10

The results of the four newly introduced measures show that m and c are strongly positively associated in both data sets by producing a measure value of 0.91. However, lift and χ 2 generate dramatically different measure values for D1 and D2 due to their sensitivity to $\overline{mc}$.



Null Invariance: An Important Property

- Why is null invariance crucial for the analysis of massive transaction data?
  - Many transactions may contain neither milk nor coffee!

milk vs. coffee contingency table

	$milk$	$\neg milk$	Σ_{row}
$coffee$	mc	$\neg mc$	c
$\neg coffee$	$m\neg c$	$\neg m\neg c$	$\neg c$
Σ_{col}	m	$\neg m$	Σ

- Lift and χ^2 are not null-invariant: not good to evaluate data that contain too many or too few null transactions!
- Many measures are not null-invariant!

Null-transactions w.r.t. m and c

Data set	mc	$\neg mc$	$m\neg c$	$\neg m\neg c$	χ^2	$Lift$
D_1	10,000	1,000	1,000	100,000	90557	9.26
D_2	10,000	1,000	1,000	100	0	1
D_3	100	1,000	1,000	100,000	670	8.44
D_4	1,000	1,000	1,000	100,000	24740	25.75
D_5	1,000	100	10,000	100,000	8173	9.18
D_6	1,000	10	100,000	100,000	965	1.97

"Among the all confidence, max confidence, Kulczynski, and cosine measures, which is best at indicating interesting pattern relationships?"
To answer this question, we introduce the imbalance ratio (IR), which assesses the imbalance of two itemsets, A and B, in rule implications. It is defined as

$$IR(A,B) = \frac{|sup(A) - sup(B)|}{sup(A) + sup(B) - sup(A \cup B)},$$

where the numerator is the absolute value of the difference between the support of the itemsets A and B, and the denominator is the number of transactions containing A or B. If the two directional implications between A and B are the same, then IR(A,B) will be zero. Otherwise, the larger the

difference between the two, the larger the imbalance ratio. This ratio is independent of the number of null-transactions and independent of the total number of transactions.
In summary, the use of only support and confidence measures to mine associations may generate a large number of rules, many of which can be uninteresting to users.
Instead, we can augment the support–confidence framework with a pattern interestingness measure, which helps focus the mining toward rules with strong pattern relationships.
The added measure substantially reduces the number of rules generated and leads to the discovery of more meaningful rules.
Besides those introduced in this section, many other interestingness measures have been studied in the literature.
Because large data sets have many null-transactions, it is important to consider the nullinvariance property when selecting appropriate interestingness measures for pattern evaluation.
Among the four null-invariant measures studied here, namely all confidence, max confidence, Kulc, and cosine, we recommend using Kulc in conjunction with the imbalance ratio.

UNIT WISE IMPORTANT QUESTIONS: -

1. Can we design method that mines the complete set of frequent item sets without candidate generation? Explain with example.
2. Prove that all nonempty subsets of a frequent item set must also be frequent.
3. How are association rules generated from frequent item sets? Illustrate.
4. Apply FP-Growth algorithm to the following transactional data to find frequent item sets. List all frequent item sets with their minimum support count of 2 and confidence = 50%.

TID	List of Item IDs
1	i1,i3,i5,i7
2	i2,i4,i6,i8
3	i1,i3,i5,i7
4	i9,i7,i5,i1
5	i2,i4,i6,i7
6	i1,i2,i3,i4
7	i3,i4,i5,i6
8	i7,i8,i6,i1
9	i8,i5,i3,i2
10	i1,i3,i4,i6

5. Find the frequent itemsets and strong association rules for the following transactional database table using Apriori algorithm. Consider the thresholds as support = 30% and confidence= 40%

TID	List of Item IDs
1	i1,i3,i5,i7
2	i2,i4,i6,i8
3	i1,i3,i5,i7
4	i9,i7,i5,i1
5	i2,i4,i6,i7
6	i1,i2,i3,i4
7	i3,i4,i5,i6
8	i7,i8,i6,i1
9	i8,i5,i3,i2
10	i1,i3,i4,i6

6. Explain the terms minimum support threshold and a minimum confidence threshold.
7. Illustrate FP-Growth algorithm with suitable example.
8. Categorize various kinds of association rules with examples.

9. Explain pattern evaluation measures in detail.
10. What are the correlation measures? Explain in detail with examples.
11. A database has five transactions. Let min sup = 60% and min conf = 80%

TID	*items_bought*
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y }
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

(a) Find all frequent itemsets using Apriori and FP-growth, respectively. Compare the efficiency of the two mining processes.
(b) List all the strong association rules (with support s and confidence
c) matching the following metarule, where X is a variable representing customers, and itemi denotes variables representing items (e.g., "A," "B,"):
$\forall x \in$ transaction, buys(X,item1) $\wedge$ buys(X,item2) $\Rightarrow$ buys(X,item3) [s,c]