Mining Frequent Patterns, Associations, and Correlations: Basic Concepts and Methods

Introduction

- Imagine that you are a sales manager at AllElectronics, and you are talking to a customer who recently bought a PC and a digital camera from the store.
- What should you recommend to her next?
- Information about which products are frequently purchased by your customers following their purchases of a PC and a digital camera in sequence would be very helpful in making your recommendation.

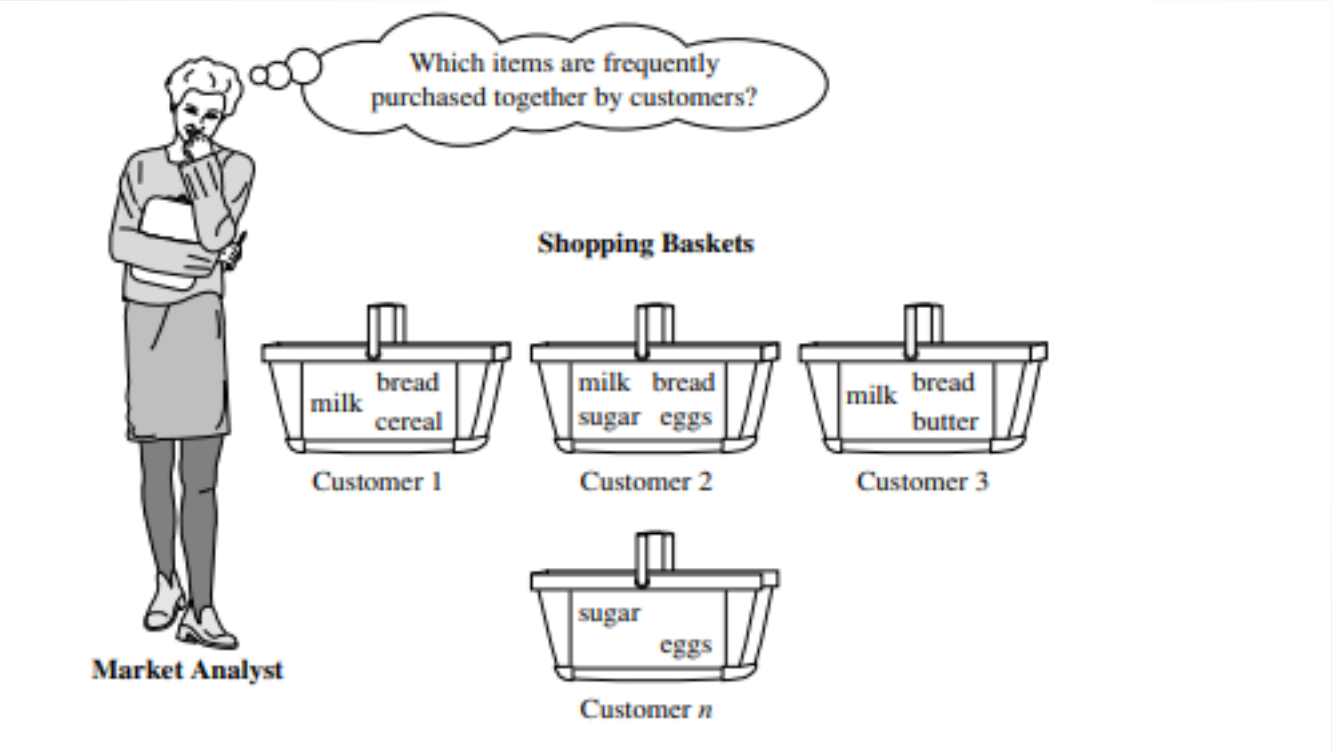
- Frequent patterns and association rules are the knowledge that you want to mine in such a scenario.
- Frequent patterns are patterns that appear frequently in a data set.
- For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset.
- 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.
- Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

- **Basic Concepts**
- Frequent pattern mining searches for recurring relationships in a given data set.

- **Market Basket Analysis: A Motivating Example**
- Frequent itemset 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.

- 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".



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

- **Example 6.1 Market basket analysis.**
- 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.
- In an alternative strategy, placing hardware and software at opposite ends of the store may attract customers who purchase such items to pick up other items along the way.
- For instance, after deciding on an expensive computer, a customer may observe security systems for sale while heading toward the software display to purchase antivirus software, and may decide to purchase a home security system as well.

- Market basket analysis can also help retailers plan which items to put on 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.

- Many Business enterprises accumulate large quantity of data from their day-to-day operations, commonly known as Market-basket transactions.
- Each row in this table corresponds to a transaction, which contains a unique identifier labelled TID and a set of items bought by a given customer, which is of much interest for learning the purchasing behavior of the customers.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

- Market basket data can be represented in a binary format where each row corresponds to a transaction and each column corresponds to an item.
- An item can be treated as a binary variable whose value is one if the item is present in a transaction and zero otherwise.
- The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently associated or purchased together.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	1
3	0	1	1	1	0	0
4	1	1	1	1	0	0

- **Item set**
- In association analysis, a collection of zero or more items is termed as item set
- E.g., {Beer, Diapers, Milk} is an example of a 3-itemset
- The null set is an itemset that does not contain any items

- **Transaction Width**
- Number of items present in a transaction

- **Association Rule**
- An Association Rule is an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets.
- The strength of an association rule can be measured in terms of its **support** and **confidence**.

- **Support**
- Support determines how often a rule is applicable to a given data set. It is the percentage of transactions in D that contain $X \cup Y$.

$$S(X \rightarrow Y) = \sigma (X \cup Y)/ N$$

$$\text{SUPPORT} = \frac{\text{number of transactions containing X and Y}}{\text{total number of transactions}}$$

- **Confidence**
- Confidence determines how frequently items in Y appears in transactions that contain X.
- It is the percentage of transactions containing X that also contain Y.

  $C(X \rightarrow Y) = \sigma (X \cup Y) / \sigma(X)$

- $$\text{CONFIDENCE} = \frac{\text{number of transactions containing X and Y}}{\text{number of transactions containing X}}$$

- Consider the rule {Milk, Diapers} → {Beer}
- Support count for {Milk, Diapers, Beer} is 2 and total number of transactions is 5,

  Support = 2/5 = 0.4 (40%)
- Confidence is obtained by dividing the support count for {Milk, Diapers , Beer} by the support count for {Milk, Diapers}
- Since there are three transactions that contains milk and diapers,

  Confidence = 2/ 3= 0.67 (67%)

- Given the definitions of transactions T, the goal of association mining is to find all rules having
  - support ≥ minsup threshold
  - Confidence ≥ minconf threshold

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

1. Frequent Itemset Generation- Find all frequent itemsets, by definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, *minsup threshold*.
2. Rule Generation- Generate strong association rules from the frequent itemsets, by definition, these rules must satisfy minimum support and minimum confidence.

- Let I = {I1, I2,..., Im} be an itemset.
- Let D, be a set of database transactions where each transaction T is a nonempty itemset such that $T \subseteq I$.
- 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.

- An itemset X is closed in a data set D if there exists no proper super-itemset Y (Y is a proper super-itemset of X if X is a proper sub-itemset of Y, that is, if $X \subset Y$. In other words, every item of X is contained in Y but there is at least one item of Y that is not in X) such that Y has the same support count as X in D.
- An itemset X is a closed frequent itemset in set D if X is both closed and frequent in D.
- 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 $X \subset Y$ and Y is frequent in D.

- **Frequent Itemset Mining Methods**
- **Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation**
- Apriori is a important algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rule.
- 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 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.**
- 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., IUA) cannot occur more frequently than I.
- Therefore, I UA is not frequent either, that is, P(I UA) < min sup.
- **"How is the Apriori property used in the algorithm?"**
- To understand this, let us look at how L_{k-1} is used to 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 is denoted C_k .
- Let l_1 and l_2 be itemsets in L_{k-1}.
- The notation $l_i[j]$ refers to the jth item in l_i (e.g., $l_1[k-2]$ refers to the second to the last item in l_1).
- For efficient implementation, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order.
- For the $(k-1)$-itemset, l_i , this means that the items are sorted such that $l_i[1] < l_i[2] < \cdots < l_i[k-1]$.
- The join, $L_{k-1} \bowtie L_{k-1}$, is performed, where members of L_{k-1} are joinable if their first $(k-2)$ items are in common.

- That is, members l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \cdots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$.
- The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $\{l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]\}$.

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 .

- A database scan to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k).
- C_k , however, can be huge, and so this could involve heavy computation.
- To reduce the size of C_k , the Apriori property is used as follows.
- Any (k − 1)-itemset that is not frequent cannot be a subset of a frequent k-itemset.
- Hence, if any (k − 1)-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k .
- This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets.

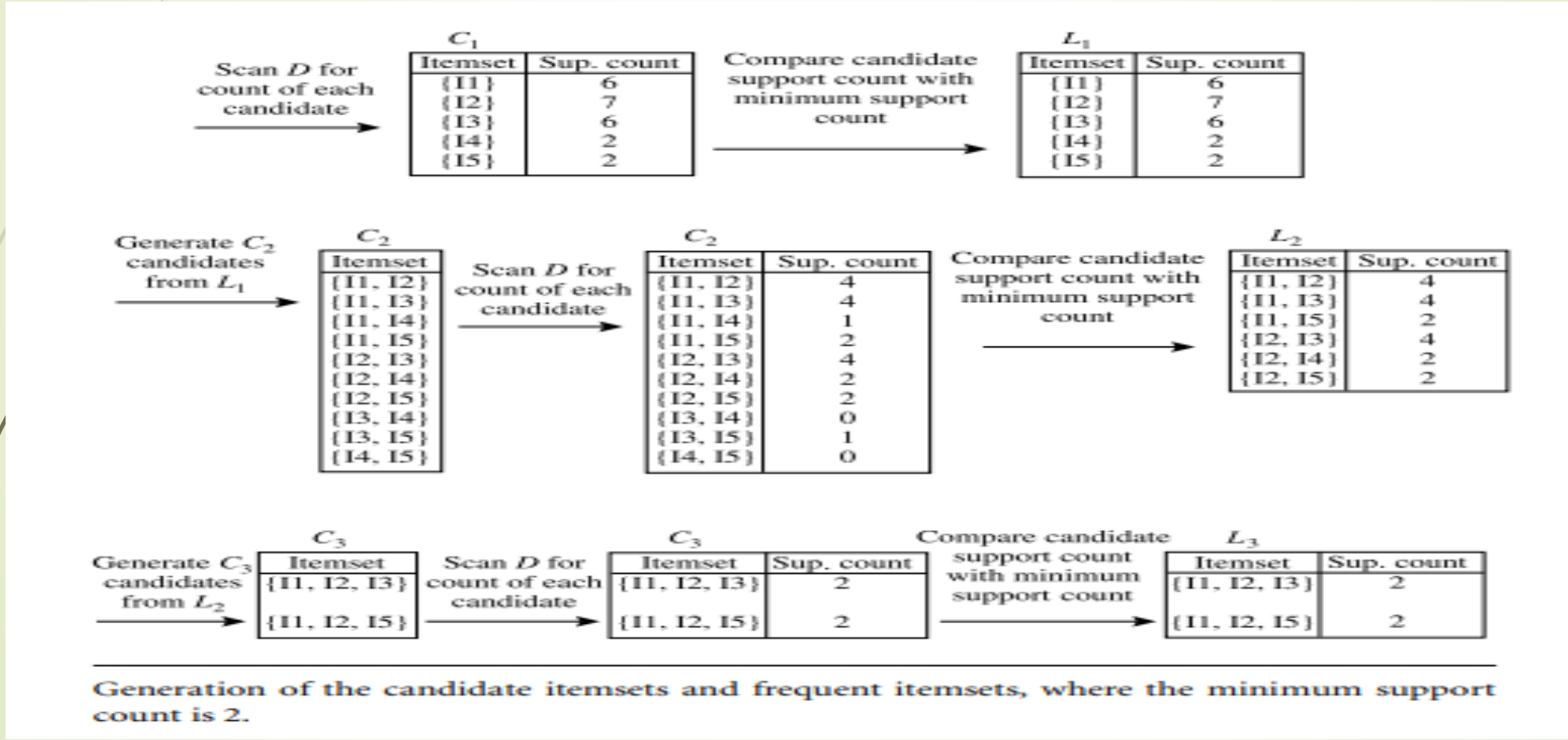
- **Apriori Example** . Let's look at a concrete example, based on the AllElectronics transaction database, D, of table below

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, $|D| = 9$.

- We use Figure below to illustrate the Apriori algorithm for finding frequent itemsets in D.



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.

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C_1. 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. (Here, we are referring to absolute support because we are using a support count. The corresponding relative support is 2/9 = 22%.) The set of frequent 1-itemsets, L_1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in C_1 satisfy minimum support.
3. To discover the set of frequent 2-itemsets, L_2, the algorithm uses the join $L_1 \bowtie L_1$ to generate a candidate set of 2-itemsets, C_2. C_2 consists of 2-itemsets. Note that no candidates are removed from C_2 during the prune step because each subset of the candidates is also frequent.
4. Next, the transactions in D are scanned and the support count of each candidate itemset in C_2 is accumulated, as shown in the middle table of the second row in Figure.

5. The set of frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.
6. The generation of the set of the candidate 3-itemsets, C_3, is detailed below.

(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 L_2. Therefore, keep {I1, I2, I3} in C_3.
- 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 L_2. Therefore, keep {I1, I2, I5} in C_3.
- The 2-item subsets of {I1, I3, I5} are {I1, I3}, {I1, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I1, I3, I5} from C_3.
- The 2-item subsets of {I2, I3, I4} are {I2, I3}, {I2, I4}, and {I3, I4}. {I3, I4} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I4} from C_3.
- The 2-item subsets of {I2, I3, I5} are {I2, I3}, {I2, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I5} from C_3.
- The 2-item subsets of {I2, I4, I5} are {I2, I4}, {I2, I5}, and {I4, I5}. {I4, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I4, I5} from C_3.

(c) Therefore, $C_3 = \{\{I1, I2, I3\}, \{I1, I2, I5\}\}$ after pruning.

Generation and pruning of candidate 3-itemsets, C_3, from L_2 using the Apriori property.

From the join step, we first get $C_3 = L_2 \bowtie L_2 = \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$. Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent. We therefore remove them from C_3, thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L_3. Note that when given a candidate k-itemset, we only need to check if its $(k - 1)$-subsets are frequent since the Apriori algorithm uses a level-wise search strategy. The resulting pruned version of C_3 is shown in the first table of the bottom row of Figure,

7. The transactions in D are scanned to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support.
8. The algorithm uses L3 $\bowtie$ L3 to generate a candidate set of 4-itemsets, C_4. 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, $C_4 = \phi$, and the algorithm terminates, having found all of the frequent itemsets.

Figure below shows pseudocode for the Apriori algorithm and its related procedures.

Algorithm: Apriori. Find frequent itemsets 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 { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(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; // join step: generate candidates
(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;
```

- Step 1 of Apriori finds the frequent 1-itemsets, L_1.
- In steps 2 through 10, L_{k-1} is used to generate candidates C_k to find L_k for $k \geq 2$.
- The apriori gen procedure generates the candidates and then uses the Apriori property to eliminate those having a subset that is not frequent (step 3).
- Once all of the candidates have been generated, the database is scanned (step 4).
- For each transaction, a subset function is used to find all subsets of the transaction that are candidates (step 5), and the count for each of these candidates is accumulated (steps 6 and 7).
- Finally, all the candidates satisfying the minimum support (step 9) form the set of frequent itemsets, L (step 11).

- A procedure can then be called to generate association rules from the frequent itemsets.
- The apriori gen procedure performs two kinds of actions, namely, join and prune.
- In the join component, L_{k-1} is joined with L_{k-1} to generate potential candidates (steps 1–4).
- The prune component (steps 5–7) employs the Apriori property to remove candidates that have a subset that is not frequent.
- The test for infrequent subsets is shown in procedure has infrequent subset.

- Example-

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

- Assume the support threshold to be 60%, which is equivalent to a minimum support count equal to 3
- Initially, every item is considered as a candidate 1-itemset

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

Items (1-itemsets)

Minimum Support = 3

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)

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

- After counting their supports, the candidate itemsets {Cola} and {Eggs} are discarded because they appear in fewer than three transactions
- In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets because the Apriori principle ensures that all supersets of the infrequent 1-itemsets must be infrequent
- Because there are only four frequent 1-itemsets, the number of candidate 2-itemsets generated by the algorithm is $\binom{4}{2} = 6$
- Two of these six candidates {Beer, Bread} and {Beer, Milk} are subsequently found to be infrequent after computing their support values
- The remaining four candidates are frequent, and thus will be used to generate candidate 3-itemsets
- With the Apriori principle, we need only to keep candidate 3-itemsets whose subsets are frequent
- The only candidate that has this property is {Bread, Diapers, Milk}

- The effectiveness of Apriori pruning strategy can be shown by counting the number of candidate itemsets generated
- With Apriori principle, the number of candidate itemset generated is computed as

$$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13 \text{ candidates}$$

- **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).
- This can be done using Eq. for 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 "$s \Rightarrow (l - s)$" if $\frac{support\_count(l)}{support\_count(s)} \geq min\_conf$, where min_conf is the minimum confidence threshold.
- Because the rules are generated from frequent itemsets, each one automatically satisfies the minimum support.
- Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

- **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:

$$
\begin{aligned}
&\{I1, I2\} \Rightarrow I5, && confidence = 2/4 = 50\% \\
&\{I1, I5\} \Rightarrow I2, && confidence = 2/2 = 100\% \\
&\{I2, I5\} \Rightarrow I1, && confidence = 2/2 = 100\% \\
&I1 \Rightarrow \{I2, I5\}, && confidence = 2/6 = 33\% \\
&I2 \Rightarrow \{I1, I5\}, && confidence = 2/7 = 29\% \\
&I5 \Rightarrow \{I1, I2\}, && confidence = 2/2 = 100\%
\end{aligned}
$$

- 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, C_k , for $k > 1$.
- Hash table is used as data structure.
- First iteration is required to count the support of each element.
- From second iteration, efforts are made to enhance execution of Apriori by utilizing hash table concept
- Hash table minimizes the number of item set generated in second iteration.
- In second iteration, i.e. 2-item set generation, for every combination of two items, we map them to diverse bucket of hash table structure and increment the bucket count.
- If count of bucket is less than min.sup_count, we remove them from candidate sets.

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

- The hash function used is

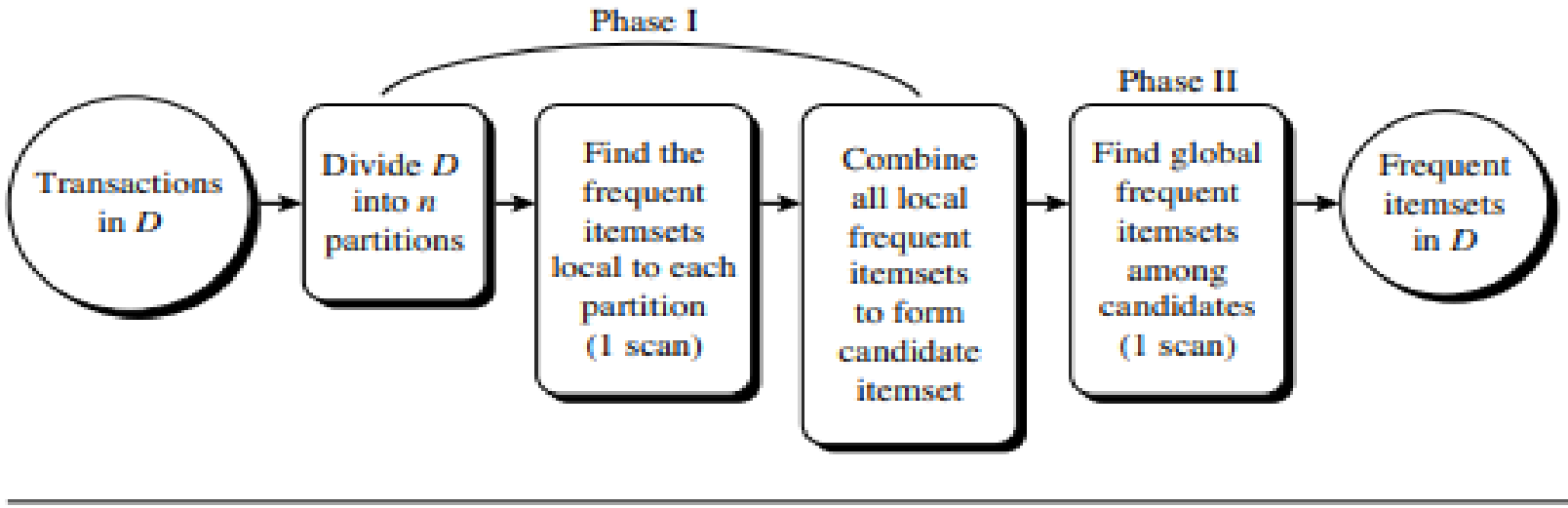
$$H(x, y)=((\text{Order of First})* 10+(\text{Order of Second}))\bmod 7$$

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

- 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).

- **Partitioning (partitioning the data to find candidate itemsets):**
- A partitioning technique can be used that requires just two database scans to mine the frequent itemsets (Figure below).



Mining by partitioning the data.

- It consists of two phases. In phase I, the algorithm divides the transactions of D into n non-overlapping partitions.
- If the minimum relative support threshold for transactions in D is min sup, then the minimum support count for a partition is *min sup × the number of transactions in that partition*.
- Example- consider the transactions in the Boolean form.

	I1	**I2**	**I3**	**I4**	**I5**
T1	1	0	0	0	1
T2	0	1	0	1	0
T3	0	0	0	1	1
T4	0	1	1	0	0
T5	0	0	0	0	1
T6	0	1	1	1	0

- Database is divided into three partitions.,each having two transactions with a support of 20%
- the support count is 1 for local i.e 20% of 2 transactions is 1.
- The support count is 3 for global ie. 20% of 6 is 2.

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	

- For each partition, all the local frequent itemsets (i.e., the itemsets frequent within the partition) are found.
- A local frequent itemset may or may not be frequent with respect to the entire database, D.
- However, any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions.
- Therefore, all local frequent itemsets are candidate itemsets with respect to D.
- The collection of frequent itemsets from all partitions forms the global candidate itemsets with respect to D.
- 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.
- Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

A Pattern-Growth Approach for Mining Frequent Itemsets

- In many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain.
- However, it can 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.*

- "Can we design a method that mines the complete set of frequent itemsets without such a costly candidate generation process?"
- An interesting method in this attempt is called frequent pattern growth, or simply FP-growth, which adopts a **divide-and-conquer strategy** as follows.
- First, it compresses the database representing frequent items into a frequent pattern tree, or 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 database separately.
- For each "pattern fragment," only its associated data sets need to be examined.
- Therefore, this approach may substantially reduce the size of the data sets to be searched, along with the "growth" of patterns being examined.

- **Example -FP-growth** (finding frequent itemsets without candidate generation).
- We reexamine the mining of transaction database, D, of Table below using the frequent pattern growth approach.

Transactional Data for an *AllElectronics* Branch

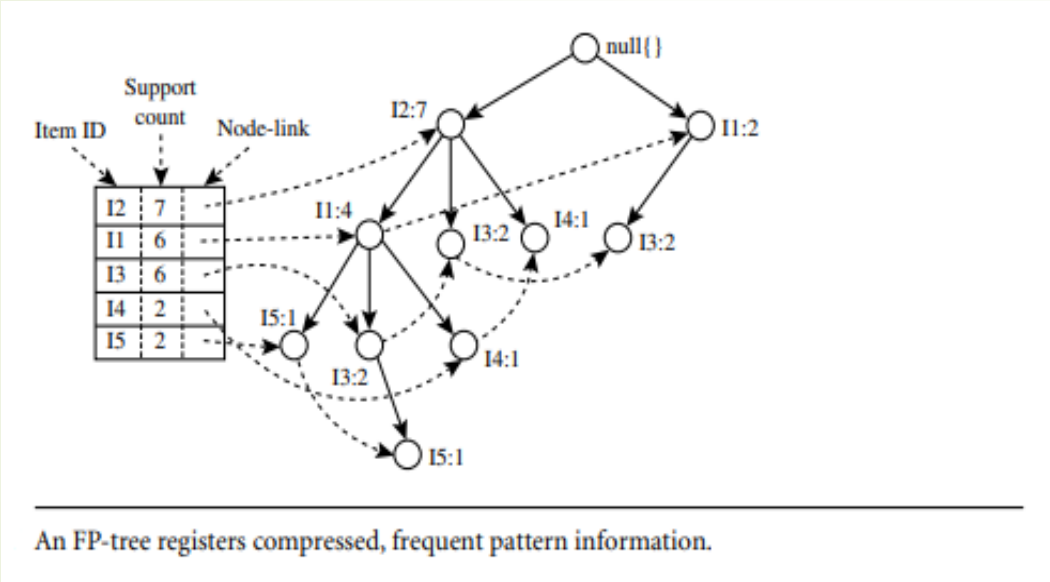
TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

- 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}}.

- An FP-tree is then constructed as follows.
- 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, <I2: 1>, <I1: 1>, and <I5: 1>, 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.
- Therefore, we instead increment the count of the I2 node by 1, and create a new node, <I4: 1>, which is linked as a child to <I2: 2>.
- In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links.

- The tree obtained after scanning all the transactions is shown in Figure below with the associated node-links.



An FP-tree registers compressed, frequent pattern information.

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

- 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 below and detailed as follows

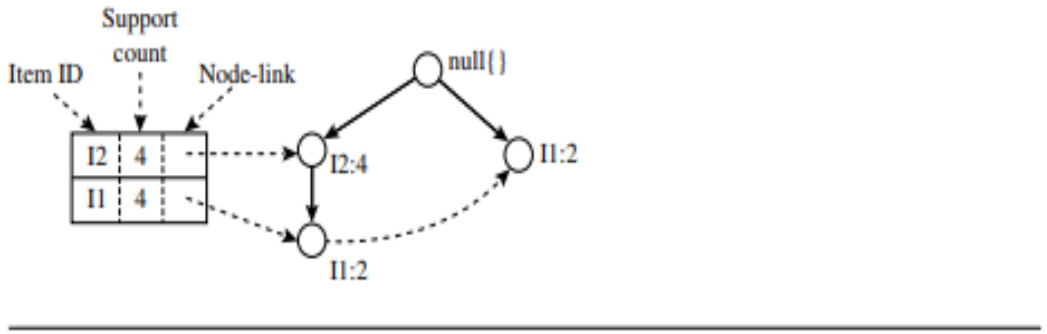
Mining the FP-Tree by Creating Conditional (Sub-)Pattern Bases

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}

- We first consider I5, which is the last item in L, rather than the first.
- The reason for starting at the end of the list will become apparent as we explain the FP-tree mining process. I5 occurs in two FP-tree branches of Figure(previous slide).
- (The occurrences of I5 can easily be found by following its chain of node-links.)
- The paths formed by these branches are <I2, I1, I5: 1> and <I2, I1, I3, I5: 1>.

- Therefore, considering I5 as a suffix, its corresponding two prefix paths are <I2, I1: 1> and <I2, I1, I3: 1>, which form its conditional pattern base.
- Using this conditional pattern base as a transaction database, we build an I5-conditional FP-tree, which contains only a single path, <I2: 2, I1: 2>; I3 is not included because its support count of 1 is less than the minimum support count.
- The single path generates all the combinations of frequent patterns: {I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}.
- For I4, its two prefix paths form the conditional pattern base, {{I2 I1: 1}, {I2: 1}}, which generates a single-node conditional FP-tree, <I2: 2>, and derives one frequent pattern, {I2, I4: 2}.
- Similar to the preceding analysis, I3's conditional pattern base is {{I2, I1: 2}, {I2: 2}, {I1: 2}}.

- Its conditional FP-tree has two branches, <I2: 4, I1: 2> and <I1: 2>, as shown in Figure below, which generates the set of patterns {{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}}.



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

- Finally, I1's conditional pattern base is {{I2: 4}}, with an FP-tree that contains only one node, <I2: 4>, which generates one frequent pattern, {I2, I1: 4}.

- This mining process is summarized in Figure below.

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, α)
(1)  if Tree contains a single path P then
(2)     for each combination (denoted as β) of the nodes in the path P
(3)        generate pattern β ∪ α with support_count = minimum support count of nodes in β;
(4)  else for each a_i in the header of Tree {
(5)     generate pattern β = a_i ∪ α with support_count = a_i.support_count;
(6)     construct β's conditional pattern base and then β's conditional FP_tree Tree_β;
(7)     if Tree_β ≠ ∅ then
(8)        call FP_growth(Tree_β, β); }
```

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

- The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively and then concatenating the suffix.
- It uses the least frequent items as a suffix, offering good selectivity.
- The method substantially reduces the search costs.
- When the database is large, it is sometimes unrealistic to construct a main memory-based FP-tree.
- A study of the FP-growth method performance shows that it is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm.

- **Mining Frequent Itemsets Using the Vertical Data Format**
- Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in TID-itemset format (i.e., {TID : itemset}), where TID is a transaction ID and itemset is the set of items bought in transaction TID.
- This is known as the horizontal data format.
- Alternatively, data can be presented in item-TID set format (i.e., {item : TID set}), where item is an item name, and TID set is the set of transaction identifiers containing the item.
- This is known as the vertical data format.

- Example -Mining frequent itemsets using the vertical data format.
- Consider the horizontal data format of the transaction database, D, of Table below

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

- This can be transformed into the vertical data format shown in Table below by scanning the data set once.

The Vertical Data Format of the Transaction Data Set D of Table 6.1

itemset	*TID_set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

- Mining can be performed on this data set by intersecting the TID sets of every pair of frequent single items.
- The minimum support count is 2.

- Because every single item is frequent in Table above, there are 10 intersections performed in total, which lead to eight nonempty 2-itemsets, as shown in Table below.

2-Itemsets in Vertical Data Format

itemset	*TID_set*
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

- Notice that because the itemsets {I1, I4} and {I3, I5} each contain only one transaction, they do not belong to the set of frequent 2-itemsets.

- Based on the Apriori property, a given 3-itemset is a candidate 3-itemset only if every one of its 2-itemset subsets is frequent.
- The candidate generation process here will generate only two 3-itemsets: {I1, I2, I3} and {I1, I2, I5}.
- By intersecting the TID sets of any two corresponding 2-itemsets of these candidate 3-itemsets, it derives Table below, where there are only two frequent 3-itemsets: {I1, I2, I3: 2} and {I1, I2, I5: 2}.

3-Itemsets in Vertical Data Format

itemset	*TID_set*
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

- This example illustrates the process of mining frequent itemsets by exploring the vertical data format.
- First, we transform the horizontally formatted data into the vertical format by scanning the data set once.
- The support count of an itemset is simply the length of the TID set of the itemset.
- Starting with k = 1, the frequent k-itemsets can be used to construct the candidate (k + 1)-itemsets based on the Apriori property.
- The computation is done by intersection of the TID sets of the frequent k-itemsets to compute the TID sets of the corresponding (k + 1)-itemsets.
- This process repeats, with k incremented by 1 each time, until no frequent itemsets or candidate itemsets can be found.

- Besides taking advantage of the Apriori property in the generation of candidate (k + 1)-itemset from frequent k-itemsets, another merit of this method is that there is no need to scan the database to find the support of (k + 1)-itemsets (for $k \geq 1$).
- This is because the TID set of each k-itemset carries the complete information required for counting such support.
- However, the TID sets can be quite long, taking substantial memory space as well as computation time for intersecting the long sets.
- To further reduce the cost of registering long TID sets, as well as the subsequent costs of intersections, we can use a technique called diffset, which keeps track of only the differences of the TID sets of a (k + 1)-itemset and a corresponding k-itemset.

- For instance, in the example we have {I1} = {T100, T400, T500, T700, T800, T900} and {I1, I2} = {T100, T400, T800, T900}.
- The diffset between the two is diffset({I1, I2}, {I1}) = {T500, T700}.
- Thus, rather than recording the four TIDs that make up the intersection of {I1} and {I2}, we can instead use diffset to record just two TIDs, indicating the difference between {I1} and {I1, I2}.
- Experiments show that in certain situations, such as when the data set contains many dense and long patterns, this technique can substantially reduce the total cost of vertical format mining of frequent itemsets.

- **Which Patterns Are Interesting?—Pattern Evaluation Methods**
- Most association rule mining algorithms employ a support–confidence framework.
- Although minimum support and confidence thresholds help weed out or exclude the exploration of a good number of uninteresting rules, many of the rules generated are still not interesting to the users.
- Unfortunately, this is especially true when mining at low support thresholds or mining for long patterns.
- This has been a major bottleneck for successful application of association rule mining.

- **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, can be used as one step toward the goal of weeding out uninteresting rules that would otherwise be presented to the user.
- "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”) **(6.6)**

  [support = 40%, confidence = 66%].

- Rule (6.6) 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 (6.6) is misleading because the probability of purchasing videos is 75%, which is even larger than 66%.
- In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other.
- Without fully understanding this phenomenon, we could easily make unwise business decisions based on Rule (6.6).
- This example also illustrates that the confidence of a rule $A \Rightarrow B$ can be deceiving.
- It does not measure the real strength (or lack of strength) of the correlation and implication between A and B.
- Hence, alternatives to the support–confidence framework can be useful in mining interesting data relationships.

- **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

  $A \Rightarrow B$ [support, confidence, correlation]. **(6.7)**

- That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets A and B.
- There are many different correlation measures available.
- Let us determine which would be good for mining large data sets.

Correlation Measures in Association Rule Mining

- Correlation measures help assess **the strength and significance** of association rules beyond simple support and confidence.
- These measures include **Lift, Kulczynski Measure, Cosine Measure, and All-Confidence** to determine whether two items co-occur more or less often than expected by chance.

1. Lift

Definition:

Lift measures how much more likely item **B** is to be purchased **when item A is purchased**, compared to when **B is bought independently**.

$$\text{Lift}(A \Rightarrow B) = \frac{\text{Support}(A \cap B)}{\text{Support}(A) \times \text{Support}(B)}$$

- **Lift > 1** → A and B are **positively correlated** (A increases the chance of buying B).
- **Lift = 1** → A and B are **independent** (no correlation).
- **Lift < 1** → A and B are **negatively correlated** (A reduces the chance of buying B).

Example

- **Transactions:**
  - 100 total transactions
  - `{Milk}` appears in **40** transactions
  - `{Bread}` appears in **50** transactions
  - `{Milk, Bread}` appears in **20** transactions

$$\text{Lift}(\text{Milk} \Rightarrow \text{Bread}) = \frac{20/100}{(40/100) \times (50/100)} = \frac{0.2}{0.4 \times 0.5} = \frac{0.2}{0.2} = 1$$

🔷 **Interpretation:** Lift = 1 → **Milk and Bread are independent** (no special relationship).

Correlation Measures in Association Rule Mining

2. Kulczynski Measure

Definition:

Kulczynski measures the **average** of two confidence values:

$$\text{Kulczynski}(A \Rightarrow B) = \frac{1}{2}\left[\frac{\text{Support}(A \cap B)}{\text{Support}(A)} + \frac{\text{Support}(A \cap B)}{\text{Support}(B)}\right]$$

- If **Kulczynski** = **1**, A and B occur together as often as expected.
- A **higher Kulczynski value** indicates a **stronger association** between A and B.

Example

- **Transactions:**
  - 100 total transactions
  - `{Milk}` appears in **40** transactions
  - `{Bread}` appears in **50** transactions
  - `{Milk, Bread}` appears in **20** transactions

$$\text{Kulczynski}(\text{Milk} \Rightarrow \text{Bread}) = \frac{1}{2}\left[\frac{20}{40} + \frac{20}{50}\right] = \frac{1}{2}\left[0.5 + 0.4\right] = 0.45$$

🔷 **Interpretation:** Kulczynski = 0.45 → **Moderate association** between Milk and Bread.

Correlation Measures in Association Rule Mining

3. Cosine Measure

Definition:

Cosine similarity measures the **degree of association** between two items based on the frequency of their co-occurrence.

$$\text{Cosine}(A, B) = \frac{\text{Support}(A \cap B)}{\sqrt{\text{Support}(A) \times \text{Support}(B)}}$$

- **Higher cosine values** (closer to 1) indicate **stronger correlation** between A and B.

Example

- **Transactions:**
  - 100 total transactions
  - `{Milk}` appears in **40** transactions
  - `{Bread}` appears in **50** transactions
  - `{Milk, Bread}` appears in **20** transactions

$$\text{Cosine}(\text{Milk}, \text{Bread}) = \frac{0.2}{\sqrt{0.4 \times 0.5}} = \frac{0.2}{\sqrt{0.2}} = \frac{0.2}{0.447} = 0.447$$

🔷 **Interpretation:** Cosine = 0.447 → **Moderate association** between Milk and Bread.

Correlation Measures in Association Rule Mining

4. All-Confidence Measure

Definition:

All-confidence measures the **minimum** confidence level across **all subsets** of the itemset.

$$\text{All-Confidence}(A, B) = \frac{\text{Support}(A \cap B)}{\max(\text{Support}(A), \text{Support}(B))}$$

- **Higher all-confidence values** indicate **stronger, reliable associations.**
- Unlike Lift, it **prevents rules dominated by a single frequent item.**

Example

- **Transactions:**
  - 100 total transactions
  - `{Milk}` appears in **40** transactions
  - `{Bread}` appears in **50** transactions
  - `{Milk, Bread}` appears in **20** transactions

$$\text{All-Confidence}(\text{Milk}, \text{Bread}) = \frac{0.2}{\max(0.4, 0.5)} = \frac{0.2}{0.5} = 0.4$$

🔷 **Interpretation:** All-Confidence = 0.4 → **Moderate association** but **not very strong.**

Comparison of Correlation Measures

Measure	Formula	Range	Meaning
Lift	$\frac{P(A \cap B)}{P(A)P(B)}$	> 1 (positive correlation), $= 1$ (independent), < 1 (negative correlation)	Shows dependency strength
Kulczynski	$\frac{1}{2}\left[\frac{P(A \cap B)}{P(A)} + \frac{P(A \cap B)}{P(B)}\right]$	$[0, 1]$	Average of rule confidence
Cosine	$\frac{P(A \cap B)}{\sqrt{P(A)P(B)}}$	$[0, 1]$	Similarity between A and B
All-Confidence	$\frac{P(A \cap B)}{\max(P(A), P(B))}$	$[0, 1]$	Ensures balanced support across items

- **Each correlation measure offers unique insights:**
  - Lift is best for dependency strength.
  - Kulczynski balances confidence levels.
  - Cosine is useful for similarity analysis.
  - All-confidence ensures fair support across all items.

- 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.

Constraint-Based Association Mining

- A data mining process may uncover thousands of rules from a given set of data, most of which end up being unrelated or uninteresting to the users.
- Often, users have a good sense of which "direction" of mining may lead to interesting patterns and the "form" of the patterns or rules they would like to find.
- Thus, a good heuristic is to have the users specify such intuition or expectations as constraints to confine the search space.
- This strategy is known as constraint-based mining.
- The constraints can include the following:

Constraint-Based Association Mining

- **Types of the constraints :**
- **Knowledge-Based Constraints**
  - specify the type of patterns to be extracted, such as association, Correlation
  - focusing only on relevant relationships such as **positive rules, negative rules, causal rules, hybrid rules, or correlation-based rules**.

1. **Positive Association Rules**
   - Identify items that frequently appear together.
   - **Example:** `{Bread} → {Butter}` (Customers who buy bread often buy butter).
2. **Negative Association Rules**
   - Identify items that rarely appear together.
   - **Example:** `{Tea} → ¬{Coffee}` (Customers who buy tea rarely buy coffee).
3. **Causal Rules**
   - Identify relationships where one item causes another.
   - **Example:** `{Gym Membership} → {Protein Supplements}` (People who join a gym are likely to buy protein supplements).

Constraint-Based Association Mining

- **Types of the constraints :**
- **Knowledge-Based Constraints**
  - specify the type of patterns to be extracted, such as association, Correlation
  - focusing only on relevant relationships such as **positive rules, negative rules, causal rules, hybrid rules, or correlation-based rules**.

4. **Hybrid Rules**
   - Combine both positive and negative associations.
   - **Example:** `{Shoes} → {Socks}, ¬{Sandals}` (Customers who buy shoes often buy socks but rarely buy sandals).
5. **Correlation Rules**
   - Identify strong statistical relationships between items.
   - Correlation measures (like **Lift** and **Chi-Square**) determine if the relationship is meaningful beyond just co-occurrence.
   - **Example:** `{Laptop} ↔ {Mouse}` (Customers who buy a laptop are **strongly correlated** with buying a mouse, with a Lift value > 1).

Example of Knowledge Type Constraints in Association Rule Mining

Scenario:

A **supermarket** wants to analyze customer purchases but is interested in **both negative and correlation-based rules** to understand buying behavior better.

Dataset (Transactions)

Transaction ID	Items Purchased
T1	{Bread, Butter, Milk}
T2	{Bread, Butter, Eggs}
T3	{Milk, Butter, Cheese}
T4	{Bread, Milk, Eggs}
T5	{Milk, Butter, Eggs}

Applying Knowledge Type Constraints (Negative & Correlation Rules)

✅ **Negative Rule Extracted:** `{Bread} → ¬{Cheese}`

- Customers who buy bread rarely buy cheese.

✅ **Correlation Rule Extracted:** `{Milk} ↔ {Butter} (Lift = 2.5)`

- Milk and butter have a strong correlation, meaning they are **more likely to be bought together than expected by chance.**

✅ **Correlation Rule Extracted:** `{Eggs} ↔ {Bread} (Lift = 1.8)`

- Customers who buy eggs are **moderately likely** to buy bread.

Constraint-Based Association Mining

- **Types of the constraints :**
- **Data Constraints**
  - **Define restrictions on the dataset itself** before applying mining techniques.
  - These constraints help in refining the dataset by limiting the scope based on **specific attributes, values, or conditions**, ensuring that only relevant and meaningful patterns are extracted.

Constraint-Based Association Mining

- **Types of Data Constraints**
  - **Attribute Constraints**
    - Restrict mining to specific attributes (columns).
    - **Example:** Only consider transactions containing "Food Items" and ignore others.
  - **Value Constraints**
    - Focus on specific values in attributes.
    - **Example:** Analyze only transactions where the total amount is **greater than $50**.
  - **Range Constraints**
    - Filter transactions based on numerical limits.
    - **Example:** Consider only transactions made in the **last 6 months**.
  - **Category Constraints**
    - Limit the dataset to a specific category.
    - **Example:** Analyze purchases only for **dairy products** and exclude other categories.

Constraint-Based Association Mining

Example of Data Constraints in Association Rule Mining

Scenario:

A **supermarket** wants to analyze customer purchases but is only interested in:

- Transactions with a total **bill amount greater than $20** (**Value Constraint**).
- Purchases made **in the last 3 months** (**Range Constraint**).
- Only transactions involving **grocery items** (**Category Constraint**).

Dataset (Before Applying Constraints)

Transaction ID	Items Purchased	Bill Amount	Date
T1	{Bread, Butter, Milk}	$15	2024-09-10
T2	{Bread, Butter, Eggs}	$25	2024-11-05
T3	{Milk, Butter, Cheese}	$30	2024-12-15
T4	{Laptop, Mouse}	$800	2024-12-18
T5	{Milk, Butter, Eggs}	$22	2024-12-25

Applying Data Constraints

Filtered Transactions (After Applying Constraints):

✅ Only transactions with **Bill Amount > $20** → **T2, T3, T5**

✅ Only transactions from **last 3 months (Oct–Dec 2024)** → **T3, T5**

✅ Only transactions with **Grocery Items** (No electronics) → **T3, T5**

Transaction ID	Items Purchased	Bill Amount	Date
T3	{Milk, Butter, Cheese}	$30	2024-12-15
T5	{Milk, Butter, Eggs}	$22	2024-12-25

Final Association Rule Mining (After Applying Constraints)

Rules Extracted:

- **Correlation Rule:** `{Milk} ⇔ {Butter} (Lift = 2.5)`
- **Negative Rule:** `{Cheese} → ¬{Eggs}` (Cus ers buying cheese rarely buy eggs).

Constraint-Based Association Mining

- **Types of constraints :**
- **Dimension/level constraints**:
  - Dimension/level constraints specify the desired **dimensions (attributes)** of the data or the **levels of concept hierarchies** to be considered during data mining.
  - These constraints help in filtering out irrelevant data and focusing only on specific dimensions or hierarchical levels, improving efficiency and relevance.
  - **Concept Hierarchies & Levels**
  - In data mining, many attributes have **hierarchical levels**. For example:
  - **Location Hierarchy:**
    - **Country → State → City → Store**
  - **Time Hierarchy:**
    - **Year → Quarter → Month → Week → Day**
  - **Product Hierarchy:**
    - **Category → Subcategory → Brand → Product**

Constraint-Based Association Mining

Example of Dimension/Level Constraints

Scenario:

A **retail company** wants to analyze product sales but is only interested in trends at the **State level** rather than individual stores.

Dataset (Full Hierarchy Available)

Transaction ID	Product	Store	City	State	Country	Sales Amount
T1	Laptop	Store A	NYC	New York	USA	$1200
T2	Phone	Store B	LA	California	USA	$800
T3	Laptop	Store C	Dallas	Texas	USA	$1100
T4	Tablet	Store D	Houston	Texas	USA	$600

Applying Dimension/Level Constraint:

- Ignore **Store** and **City** levels.
- Aggregate data at the **State level** instead.

Filtered Data (After Applying Constraint)

State	Total Sales
New York	$1200
California	$800
Texas	$1700

Constraint-Based Association Mining

- **Interestingness constraints**:
  - Interestingness constraints are used in **association rule mining** to filter and prioritize the most valuable and meaningful patterns.
  - Since association mining can generate a huge number of rules, these constraints **eliminate unimportant rules** and focus on those that provide useful insights.
  - These specify thresholds on statistical measures of rule interestingness, such as support, confidence, and correlation.

Constraint-Based Association Mining

- **Interestingness constraints**:
- **Types of Interestingness Constraints:**
  - **Support Constraint**
    - Minimum occurrence of a rule in the dataset.
    - **Example:** Ignore rules that appear in less than **5% of transactions**.
  - **Confidence Constraint**
    - Measures how often a rule is correct.
    - **Example:** Only keep rules with confidence **above 70%**.
  - **Lift Constraint**
    - Measures the strength of an association.
    - **Example:** Focus only on rules with **Lift > 1.5** to find strong correlations.
  - **Unexpectedness Constraint**
    - Prioritize rules that are surprising or counterintuitive.
    - Example: {Expensive Wine} → {Young Customers} (Surprising insight that young people buy costly wine).

Constraint-Based Association Mining

- Interestingness constraints:

Example of Data Constraints in Association Rule Mining

Scenario:

A **supermarket** wants to analyze customer purchases but is only interested in:

- Transactions with a total **bill amount greater than $20** (**Value Constraint**).
- Purchases made **in the last 3 months** (**Range Constraint**).
- Only transactions involving **grocery items** (**Category Constraint**).

Dataset (Before Applying Constraints)

Transaction ID	Items Purchased	Bill Amount	Date
T1	{Bread, Butter, Milk}	$15	2024-09-10
T2	{Bread, Butter, Eggs}	$25	2024-11-05
T3	{Milk, Butter, Cheese}	$30	2024-12-15
T4	{Laptop, Mouse}	$800	2024-12-18
T5	{Milk, Butter, Eggs}	$22	2024-12-25

Dataset (After Applying Data Constraints)

Transaction ID	Items Purchased	Bill Amount	Date
T3	{Milk, Butter, Cheese}	$30	2024-12-15
T5	{Milk, Butter, Eggs}	$22	2024-12-25

Applying Interestingness Constraints

✅ **Support Constraint:** Ignore rules occurring in **less than 5% of transactions**.

✅ **Confidence Constraint:** Only keep rules with confidence **above 70%**.

✅ **Lift Constraint:** Select rules where **Lift > 1.5** to ensure strong correlation.

Final Extracted Rules (After Interestingness Constraints):

1. **High Confidence Rule:** `{Milk} → {Butter} (Confidence = 80%)`
2. **Strong Correlation:** `{Milk} ↔ {Butter} (Lift = 2.5)`
3. **Unexpected Rule:** `{Cheese} → ¬{Eggs} (Confidence = 85%)`

Constraint-Based Association Mining

- **Rule constraints**:
- Rule constraints define **the structure and format** of the rules that should be mined.
- Instead of mining all possible association rules, these constraints focus on **specific patterns, conditions, or rule structures** based on predefined templates or logical relationships.

Constraint-Based Association Mining

- **Types of Rule Constraints**
- **Metarules (Rule Templates)**
  - Define the pattern of the rule that should be mined.
  - **Example:** If the supermarket is only interested in rules **where a dairy product appears in the antecedent**, the metarule could be:
  - **Template**: {Dairy_Item} → {X}
  - **Result:** {Milk} → {Butter}
- **Predicate Constraints**
  - Limit the number of items in the **antecedent (LHS)** or **consequent (RHS)**.
  - **Example:**
  - **Max 2 items in antecedent**: {Milk, Butter} → {X}
  - **Max 1 item in consequent: {X} → {Cheese}**

Constraint-Based Association Mining

- **Types of Rule Constraints**
- **Attribute-Value Constraints**
  - Restrict the rule based on attribute relationships.
- **Example**:
  - Only extract rules where the total bill is above $20 → {Milk, Butter} → {Eggs} (applies only if the transaction total exceeds $20).
- **Aggregate Constraints**
  - Enforce conditions based on aggregate values like count, sum, or average.
  - **Example:**
    - If at least 10 customers purchased an item together → Keep the rule {Milk} → {Butter}.Ignore rules involving rare items.

Constraint-Based Association Mining

Example of Rule Constraints in Our Case

Dataset (Filtered Transactions with Constraints Applied)

Transaction ID	Items Purchased	Bill Amount	Date
T3	{Milk, Butter, Cheese}	$30	2024-12-15
T5	{Milk, Butter, Eggs}	$22	2024-12-25

Applying Rule Constraints

✅ **Metarule Constraint:** `{Dairy_Item} → {X}`

- **Extracted Rule:** `{Milk} → {Butter}` ✅
- **Filtered Out Rule:** `{Eggs} → {Milk}` ❌ (Doesn't match the template).

✅ **Predicate Constraint:** Limit 2 items in antecedent.

- **Extracted Rule:** `{Milk, Butter} → {X}` ✅
- **Filtered Out Rule:** `{Milk, Butter, Cheese} → {Eggs}` ❌ (Too many items).

✅ **Aggregate Constraint:** Only consider rules occurring in **at least 5 transactions**.

- **Extracted Rule:** `{Milk} → {Butter}` (Occurs in 10+ transactions).
- **Filtered Out Rule:** `{Cheese} → {Eggs}` (Occurs in only 2 transactions).

Final Rules After Applying Rule Constraints

1. `{Milk} → {Butter}` (Follows metarule & predicate constraints)
2. `{Milk, Butter} → {Cheese}` (Valid within constraints)

"How are metarules useful?"

Metarule-guided mining. Suppose that as a market analyst for *AllElectronics*, you have access to the data describing customers (such as customer age, address, and credit rating) as well as the list of customer transactions. You are interested in finding associations between customer traits and the items that customers buy. However, rather than finding *all* of the association rules reflecting these relationships, you are particularly interested only in determining which pairs of customer traits promote the sale of office software. A metarule can be used to specify this information describing the form of rules you are interested in finding. An example of such a metarule is

$$P_1(X, Y) \wedge P_2(X, W) \Rightarrow buys(X, \text{“}office\ software\text{”}), \tag{5.26}$$

where P_1 and P_2 are **predicate variables** that are instantiated to attributes from the given database during the mining process, X is a variable representing a customer, and Y and W take on values of the attributes assigned to P_1 and P_2, respectively. Typically, a user will specify a list of attributes to be considered for instantiation with P_1 and P_2. Otherwise, a default set may be used.

In general, a metarule forms a hypothesis regarding the relationships that the user is interested in probing or confirming. The data mining system can then search for rules that match the given metarule. For instance, Rule (5.27) matches or **complies with** Metarule (5.26).

$$age(X, \text{“}30\ldots39\text{”}) \wedge income(X, \text{“}41K\ldots60K\text{”}) \Rightarrow buys(X, \text{“}office\ software\text{”}) \tag{5.27}$$

"How are metarules useful?"

Example 5.14 **A closer look at mining guided by rule constraints.** Suppose that *AllElectronics* has a sales multidimensional database with the following interrelated relations:

- *sales(customer_name, item_name, TID)*
- *lives_in(customer_name, region, city)*
- *item(item_name, group, price)*
- *transaction(TID, day, month, year)*

where *lives_in*, *item*, and *transaction* are three dimension tables, linked to the fact table *sales* via three keys, *customer_name*, *item_name*, and *TID* (*transaction_id*), respectively.

Our association mining query is to "*Find the sales of which cheap items (where the sum of the prices is less than $100) may promote the sales of which expensive items (where the minimum price is $500) of the same group for Chicago customers in 2004.*" This can be expressed in the DMQL data mining query language as follows, where each line of the query has been enumerated to aid in our discussion:

(1) mine associations as
(2) $lives\_in(C, \_, \text{"}Chicago\text{"}) \wedge sales^{+}(C, ?\{I\}, \{S\}) \Rightarrow sales^{+}(C, ?\{J\}, \{T\})$
(3) from sales
(4) where S.year = 2004 and T.year = 2004 and I.group = J.group
(5) group by C, I.group
(6) having sum(I.price) < 100 and min(J.price) ≥ 500
(7) with support threshold = 1%
(8) with confidence threshold = 50%

"How are metarules useful?"

(1) mine associations as
(2) $lives\_in(C, \_, \text{“}Chicago\text{”}) \wedge sales^{+}(C, ?\{I\}, \{S\}) \Rightarrow sales^{+}(C, ?\{J\}, \{T\})$
(3) from sales
(4) where S.year = 2004 and T.year = 2004 and I.group = J.group
(5) group by C, I.group
(6) having sum(I.price) $<$ 100 and min(J.price) $\geq$ 500
(7) with support threshold = 1%
(8) with confidence threshold = 50%

Before we discuss the rule constraints, let's look at the above query. Line 1 is a knowledge type constraint, where association patterns are to be discovered. Line 2 specifies a metarule. This is an abbreviated form for the following metarule for hybrid-dimensional association rules (multidimensional association rules where the repeated predicate here is *sales*):

$$lives\_in(C, \_, \text{“}Chicago\text{”}) \wedge sales(C, ?I_1, S_1) \wedge \ldots \wedge sales(C, ?I_k, S_k) \wedge I = \{I_1, \ldots, I_k\} \wedge S = \{S_1, \ldots, S_k\} \Rightarrow sales(C, ?J_1, T_1) \wedge \ldots \wedge sales(C, ?J_m, T_m) \wedge J = \{J_1, \ldots, J_m\} \wedge T = \{T_1, \ldots, T_m\}$$

which means that one or more *sales* records in the form of "$sales(C, ?I_1, S_1) \wedge \ldots sales(C, ?I_k, S_k)$" will reside at the rule antecedent (left-hand side), and the question mark "?" means that only $item\_name, I_1, \ldots, I_k$ need be printed out. "$I = \{I_1, \ldots, I_k\}$" means that all the Is at the antecedent are taken from a set I, obtained from the SQL-like **where** clause of line 4. Similar notational conventions are used at the consequent (right-hand side).

The metarule may allow the generation of association rules like the following:

$$lives\_in(C, \_, \text{“}Chicago\text{”}) \wedge sales(C, \text{“}Census\_CD\text{”}, \_) \wedge sales(C, \text{“}MS/Office\text{”}, \_) \Rightarrow sales(C, \text{“}MS/SQLServer\text{”}, \_) \quad [1.5\%, 68\%], \quad (5.29)$$

Similarity and Dissimilarity

- **Similarity**
  - Numerical measure of how alike two data objects are
  - Value is higher when objects are more alike
  - Often falls in the range [0,1]
- **Dissimilarity** (e.g., distance)
  - Numerical measure of how different two data objects are
  - Lower when objects are more alike
  - Minimum dissimilarity is often 0
  - Upper limit varies
- **Proximity** refers to a similarity or dissimilarity

Data Matrix and Dissimilarity Matrix

- Data matrix
  - n data points with p dimensions
  - Two modes

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

- Dissimilarity matrix
  - n data points, but registers only the distance
  - A triangular matrix
  - Single mode

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix}$$

Proximity Measure for Nominal Attributes

- Can take 2 or more states, e.g., red, yellow, blue, green (generalization of a binary attribute)

Method 1: Simple Matching for Nominal Attributes

$$SMC = \frac{m}{p}$$

Where:

- m: Number of matching attributes between two objects
- p: Total number of attributes

Solved Example (Using 4 Nominal States)

Let's consider the following two objects with nominal attributes having more than two states:

Attribute	Object A	Object B
Color	Red	Green
Shape	Circle	Circle
Material	Plastic	Wood
Size	Large	Large

Step 1: Identify Matches

- **Color:** Different (Red vs. Green) → **No Match**
- **Shape:** Same (Circle) → **Match**
- **Material:** Different (Plastic vs. Wood) → **No Match**
- **Size:** Same (Large) → **Match**

So, $m = 2$ (Shape and Size match), and $p = 4$ (total number of attributes).

Step 2: Compute SMC

$$SMC = \frac{2}{4} = 0.5$$

Proximity Measure for Binary Attributes

- A contingency table for binary data

Object *j*

Object *i*	1	0	sum
1	q	r	$q+r$
0	s	t	$s+t$
sum	$q+s$	$r+t$	p

- Distance measure for symmetric binary variables:

$$d(i,j) = \frac{r+s}{q+r+s+t}$$

Real-life Examples

1.Sport Event Outcome:
1. 0: Team A wins, 1: Team B wins (Neither outcome is preferred)

2.Light Switch State:
1. 0: Off, 1: On

3.Gender for Voting Preferences (assuming no bias):
1. 0: Male, 1: Female (Both genders are given equal consideration)

4.Network Connection:
1. 0: Disconnected, 1: Connected (Both states are analyzed equally in certain network analyses)

5.Weather Condition:
1. 0: Sunny, 1: Rainy (Both conditions are relevant for analysis)

6.Coin Toss Outcome:
1. 0: Heads, 1: Tails (No preference between outcomes)

Proximity Measure for Binary Attributes

- A contingency table for binary data

Object *j*

Object *i*	1	0	sum
1	q	r	$q+r$
0	s	t	$s+t$
sum	$q+s$	$r+t$	p

Contingency Table for Binary Data (Real-Life Example)

Scenario: Analysis of Gym Attendance and Diet Plan Adoption

Two groups of individuals are surveyed about their gym attendance and adherence to a diet plan.

Each individual responds as follows:

- 1: Yes (attends gym or follows diet)
- 0: No (does not attend gym or follow diet)

Survey Data for 10 Individuals

Person	Gym Attendance	Diet Plan
1	1	1
2	0	1
3	1	0
4	1	1
5	0	0
6	1	1
7	0	0
8	1	0
9	0	1
10	1	1

Step 1: Count the Frequencies Using Variables q, r, s, and t

- q: Number of individuals who attend the gym and follow the diet plan (Gym = 1, Diet = 1)
- r: Number of individuals who attend the gym but do not follow the diet plan (Gym = 1, Diet = 0)
- s: Number of individuals who do not attend the gym but follow the diet plan (Gym = 0, Diet = 1)
- t: Number of individuals who neither attend the gym nor follow the diet plan (Gym = 0, Diet = 0)

Count the values:

- $q = 4$ (Persons 1, 4, 6, and 10)
- $r = 2$ (Persons 3 and 8)
- $s = 2$ (Persons 2 and 9)
- $t = 2$ (Persons 5 and 7)

Proximity Measure for Binary Attributes

- A contingency table for binary data

Object *j*

Object *i*	1	0	sum
1	q	r	$q+r$
0	s	t	$s+t$
sum	$q+s$	$r+t$	p

Step 1: Count the Frequencies Using Variables q, r, s, and t

- q: Number of individuals who attend the gym and follow the diet plan (Gym = 1, Diet = 1)
- r: Number of individuals who attend the gym but do not follow the diet plan (Gym = 1, Diet = 0)
- s: Number of individuals who do not attend the gym but follow the diet plan (Gym = 0, Diet = 1)
- t: Number of individuals who neither attend the gym nor follow the diet plan (Gym = 0, Diet = 0)

Count the values:

- $q = 4$ (Persons 1, 4, 6, and 10)
- $r = 2$ (Persons 3 and 8)
- $s = 2$ (Persons 2 and 9)
- $t = 2$ (Persons 5 and 7)

Step 2: Create the Contingency Table

Gym Attendance \ \ Diet Plan	Yes (1)	No (0)	Total
Yes (1)	$q = 4$	$r = 2$	6
No (0)	$s = 2$	$t = 2$	4
Total	6	4	10

Step 3: Conclusion

The contingency table has been filled using q, r, s, and t, where:

- $q = 4$: 4 individuals attend both the gym and follow the diet plan
- $r = 2$: 2 individuals attend the gym but don't follow the diet plan
- $s = 2$: 2 individuals don't attend the gym but follow the diet plan
- $t = 2$: 2 individuals neither attend the gym nor follow the diet plan

Proximity Measure for Binary Attributes

- A contingency table for binary data

Object *j*

Object *i*	1	0	sum
1	q	r	$q+r$
0	s	t	$s+t$
sum	$q+s$	$r+t$	p

Step 2: Create the Contingency Table

Gym Attendance \ \ Diet Plan	Yes (1)	No (0)	Total
Yes (1)	$q = 4$	$r = 2$	6
No (0)	$s = 2$	$t = 2$	4
Total	6	4	10

Step 3: Conclusion

The contingency table has been filled using q, r, s, and t, where:

- $q = 4$: 4 individuals attend both the gym and follow the diet plan
- $r = 2$: 2 individuals attend the gym but don't follow the diet plan
- $s = 2$: 2 individuals don't attend the gym but follow the diet plan
- $t = 2$: 2 individuals neither attend the gym nor follow the diet plan

Symmetric binary dissimilarity

Distance measure for symmetric binary variables:

$$d(i, j) = \frac{r + s}{q + r + s + t}$$

Where:

- $q = 4$ (both Gym and Diet = 1)
- $t = 2$ (both Gym and Diet = 0)
- $r = 2$ (Gym = 1, Diet = 0)
- $s = 2$ (Gym = 0, Diet = 1)

Proximity Measure for Binary Attributes

- A contingency table for binary data
- Distance measure for asymmetric binary variables:
- The dissimilarity based on such variables is called asymmetric binary dissimilarity, where the number of negative matches, **t**, is considered unimportant and thus is ignored in the computation, as shown in Equation

Object *i* \ Object *j*	1	0	sum
1	q	r	$q+r$
0	s	t	$s+t$
sum	$q+s$	$r+t$	p

$$d(i, j) = \frac{r + s}{q + r + s}$$

Proximity Measure for Binary Attributes

- A contingency table for binary data

Object *i* \ Object *j*	1	0	sum
1	q	r	$q+r$
0	s	t	$s+t$
sum	$q+s$	$r+t$	p

- Jaccard coefficient
- (*similarity* measure for *asymmetric* binary variables):

$$sim_{Jaccard}(i, j) = \frac{q}{q + r + s}$$

$$= 1 - d(i, j).$$

- Note: Jaccard coefficient is the same as "coherence":

$$coherence(i, j) = \frac{sup(i, j)}{sup(i) + sup(j) - sup(i, j)} = \frac{q}{(q + r) + (q + s) - q}$$

Proximity Measure for Binary Attributes

- A contingency table for binary data

Object *i* \ Object *j*	1	0	sum
1	q	r	$q+r$
0	s	t	$s+t$
sum	$q+s$	$r+t$	p

- Distance measure for symmetric binary variables:

$$d(i, j) = \frac{r + s}{q + r + s + t}$$

- Distance measure for asymmetric binary variables:

$$d(i, j) = \frac{r + s}{q + r + s}$$

- Jaccard coefficient (*similarity* measure for *asymmetric* binary variables):

$$sim_{Jaccard}(i, j) = \frac{q}{q + r + s}$$

  - Note: Jaccard coefficient is the same as "coherence":

$$coherence(i, j) = \frac{sup(i, j)}{sup(i) + sup(j) - sup(i, j)} = \frac{q}{(q + r) + (q + s) - q}$$

Dissimilarity between Binary Variables

- Example

Name	Gender	Fever	Cough	Test-1	Test-2	Test-3	Test-4
Jack	M	Y	N	P	N	N	N
Mary	F	Y	N	P	N	P	N
Jim	M	Y	P	N	N	N	N

- Gender is a symmetric attribute
- The remaining attributes are asymmetric binary
- Let the values Y and P be 1, and the value N 0

$$d(jack, mary) = \frac{0+1}{2+0+1} = 0.33$$

$$d(jack, jim) = \frac{1+1}{1+1+1} = 0.67$$

$$d(jim, mary) = \frac{1+2}{1+1+2} = 0.75$$

Categorical, Ordinal, and Ratio-Scaled Variables

- "How can we compute the dissimilarity between objects described by categorical, ordinal, and ratio-scaled variables?"
- Use data Mining page number 383

Standardizing Numeric Data

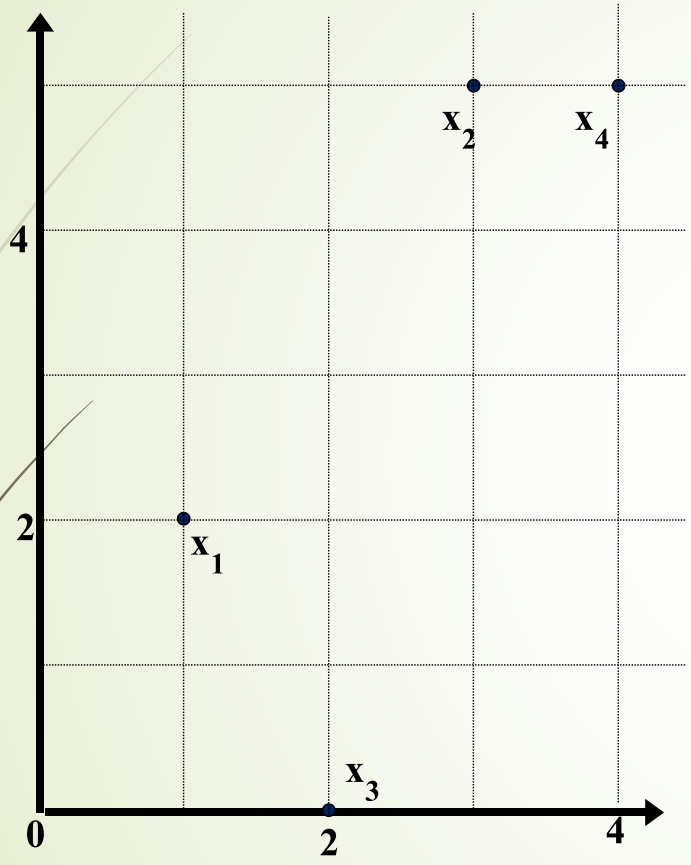
- Z-score: $z = \frac{x - \mu}{\sigma}$
  - X: raw score to be standardized, μ: mean of the population, σ: standard deviation
  - the distance between the raw score and the population mean in units of the standard deviation
  - negative when the raw score is below the mean, "+" when above
- An alternative way: Calculate the mean absolute deviation

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + \ldots + |x_{nf} - m_f|)$$

where $m_f = \frac{1}{n}(x_{1f} + x_{2f} + \ldots + x_{nf})$.

  - standardized measure (z-score): $z_{if} = \frac{x_{if} - m_f}{s_f}$
- Using mean absolute deviation is more robust than using standard deviation

Example: Data Matrix and Dissimilarity Matrix



Data Matrix

point	attribute1	attribute2
x1	1	2
x2	3	5
x3	2	0
x4	4	5

Dissimilarity Matrix

(with Euclidean Distance)

	x1	*x2*	*x3*	*x4*
x1	0			
x2	3.61	0		
x3	5.1	5.1	0	
x4	4.24	1	5.39	0

Distance on Numeric Data: Minkowski Distance

- *Minkowski distance*: A popular distance measure

$$d(i, j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{ip} - x_{jp}|^h}$$

where $i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \ldots, x_{jp})$ are two p-dimensional data objects, and h is the order (the distance so defined is also called L-h norm)

- Properties
  - $d(i, j) > 0$ if $i \neq j$, and $d(i, i) = 0$ (Positive definiteness)
  - $d(i, j) = d(j, i)$ (Symmetry)
  - $d(i, j) \leq d(i, k) + d(k, j)$ (Triangle Inequality)
- A distance that satisfies these properties is a metric

Special Cases of Minkowski Distance

- *h* = 1: Manhattan (city block, L_1 norm) distance
  - E.g., the Hamming distance: the number of bits that are different between two binary vectors

$$d(i,j)=|x_{i_1}-x_{j_1}|+|x_{i_2}-x_{j_2}|+...+|x_{i_p}-x_{j_p}|$$

- *h* = 2: (L_2 norm) Euclidean distance

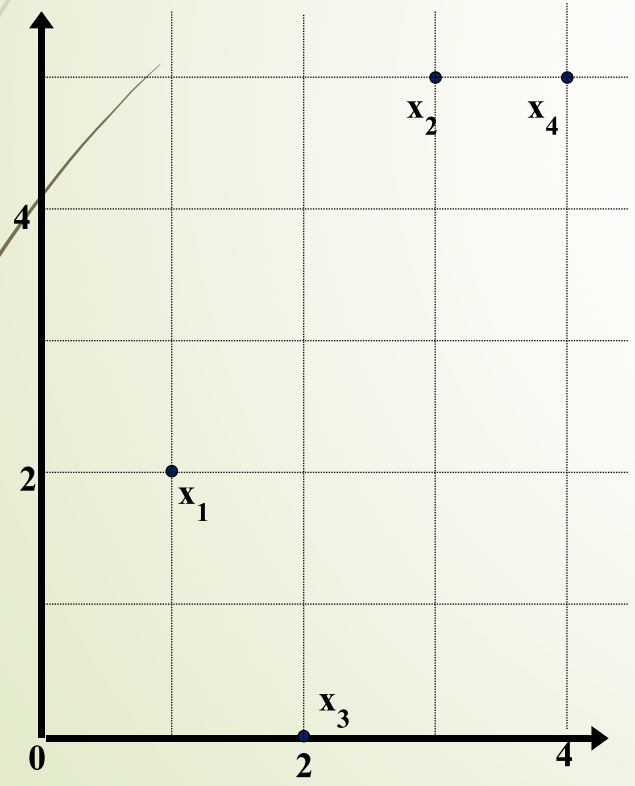
$$d(i,j)=\sqrt{(|x_{i_1}-x_{j_1}|^2+|x_{i_2}-x_{j_2}|^2+...+|x_{i_p}-x_{j_p}|^2)}$$

- $h \to \infty$. "supremum" (L_{max} norm, L_∞ norm) distance.
  - This is the maximum difference between any component (attribute) of the vectors

$$d(i,j)=\lim_{h\to\infty}\left(\sum_{f=1}^{p}|x_{if}-x_{jf}|^h\right)^{\frac{1}{h}}=\max_{f}^{p}|x_{if}-x_{jf}|$$

Example: Minkowski Distance

point	attribute 1	attribute 2
x1	1	2
x2	3	5
x3	2	0
x4	4	5



Dissimilarity Matrices

Manhattan (L_1)

L	x1	x2	x3	x4
x1	0			
x2	5	0		
x3	3	6	0	
x4	6	1	7	0

Euclidean (L_2)

L2	x1	x2	x3	x4
x1	0			
x2	3.61	0		
x3	2.24	5.1	0	
x4	4.24	1	5.39	0

Supremum

L_∞	x1	x2	x3	x4
x1	0			
x2	3	0		
x3	2	5	0	
x4	3	1	5	0

Ordinal Variables

- An ordinal variable can be discrete or continuous
- Order is important, e.g., rank
- Can be treated like interval-scaled
  - replace x_{if} by their rank $r_{if} \in \{1, \ldots, M_f\}$
  - map the range of each variable onto [0, 1] by replacing *i*-th object in the *f*-th variable by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}$$

  - compute the dissimilarity using methods for interval-scaled variables

Attributes of Mixed Type

- A database may contain all attribute types
  - Nominal, symmetric binary, asymmetric binary, numeric, ordinal
- One may use a weighted formula to combine their effects

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}$$

  - *f* is binary or nominal:

    $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$, or $d_{ij}^{(f)} = 1$ otherwise
  - *f* is numeric: use the normalized distance
  - *f* is ordinal
    - Compute ranks r_{if} and
    - Treat z_{if} as interval-scaled

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}$$

Cosine Similarity

- A **document** can be represented by thousands of attributes, each recording the *frequency* of a particular word (such as keywords) or phrase in the document.

Document	*team*	*coach*	*hockey*	*baseball*	*soccer*	*penalty*	*score*	*win*	*loss*	*season*
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

- Other vector objects: gene features in micro-arrays, …
- Applications: information retrieval, biologic taxonomy, gene feature mapping, ...
- Cosine measure: If d_1 and d_2 are two vectors (e.g., term-frequency vectors), then

$$\cos(d_1, d_2) = (d_1 \bullet d_2) / ||d_1|| \, ||d_2|| ,$$

where $\bullet$ indicates vector dot product, $||d||$: the length of vector d

Example: Cosine Similarity

- $\cos(d_1, d_2) = (d_1 \bullet d_2) / ||d_1|| \ ||d_2||$,
  where $\bullet$ indicates vector dot product, $||d||$: the length of vector d

- Ex: Find the **similarity** between documents 1 and 2.

d_1 **=** (5, 0, 3, 0, 2, 0, 0, 2, 0, 0)

d_2 **=** (3, 0, 2, 0, 1, 1, 0, 1, 0, 1)

$d_1 \bullet d_2 = 5*3+0*0+3*2+0*0+2*1+0*1+0*1+2*1+0*0+0*1 = 25$

$||d_1|| = (5*5+0*0+3*3+0*0+2*2+0*0+0*0+2*2+0*0+0*0)^{0.5} = (42)^{0.5} = 6.481$

$||d_2|| = (3*3+0*0+2*2+0*0+1*1+1*1+0*0+1*1+0*0+1*1)^{0.5} = (17)^{0.5} = 4.12$

$\cos(d_1, d_2) = 0.94$