Module 5: Syllabus

Association Rules Mining: Concepts, Apriori and FP-Growth Algorithm. **Cluster Analysis**: Introduction, Concepts, Types of data in cluster analysis, Categorization of clustering methods. **Partitioning method:** K-Means and K-Medoid Clustering

Association Rules Mining: Concepts

Association rule mining finds interesting association or correlation relationships among a large set of data items. With massive amounts of data continuously being collected and stored in databases, many industries are becoming interested in mining association rules from their databases. For example, the discovery of interesting association relationships among huge amounts of business transaction records can help catalog design, cross-marketing, lossleader analysis, and other business decision making processes.

A typical example of association rule 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" (Figure 6.1). The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers to do selective marketing and plan their shelf space. For instance, placing milk and bread within close proximity may further encourage the sale of these items together within single visits to the store.

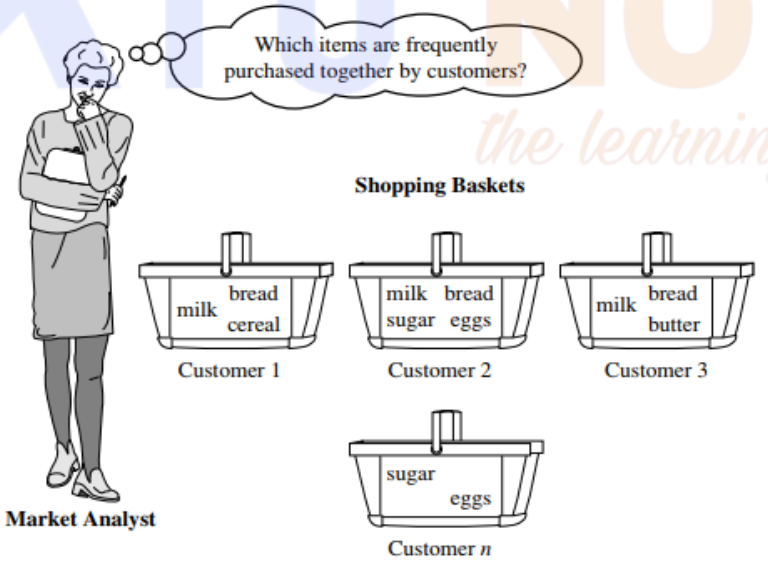


Figure 6.1 Market basket analysis.

Basic concepts

1. Transaction database

- Let I = {I1, I2,..., Im} be an itemset {SET OF ITEMS} .
- Let D, the task-relevant data, be a set of database transactions T where each transaction t_i is a nonempty item set such that $t_i \subseteq I$.
- Transaction database, T={t_1, t_2,……..,t_n}
- A transaction database T contains "n" transaction.
- Each transaction is associated with an identifier, called a TID.

Table 6.1 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

Fig: Sample Transaction database

2. Association rule

An association rule is an implication of the form X ⇒ Y, where X⊆ I, Y ⊆ I, and X∩Y = φ.

- Meaning that whenever X appears in the Transaction Y also tends to appear.
- X and Y are single item or set of Items.
- Example: {Milk} ⇒ {Bread},{Computer, CD Drive}⇒ {CD}
- Usually we prefer association rules that satisfies required support and confidence.

3. Support, Confidence and Lift

- Support and confidence are used to measure ***interestingness of the rule***
- **Support(X)** is the number of times X appears in the transaction divided by total number of transaction.

Support(X) =(No. of times X appears)/(total number of transaction)=**P(X)**

Support(X,Y) =(No. of times X and Y appears together)/(total number of transaction)=**P(X∩*Y*)**

- **Confidence** of the association rule ***X ⇒ Y*** *is defined as the ratio of support of X and Y together to the support of X*

Confidence (***X ⇒ Y)=*** *(support of X and Y together)/(support of X)=* **P(X∩*Y*)/ P(X)= P(*Y*|*X*)**

- **Lift** is used to measure the power of association between items that are purchased together.

Lift(*X ⇒ Y)=* P(X∩*Y*)/ P(X)= P(*Y*|*X*)/ P(Y)

4. Frequent Items

Items that frequently appeared in the transactions are called Frequent Items. Usually we select frequent items based on minimum support count.

Association Rules Mining: Algorithms

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

Step 1. **Find all frequent itemsets**: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count.

2. **Generate strong association rules from the frequent itemsets**: By definition, these rules must satisfy minimum support and minimum confidence.

1. Apriori Algorithm

Apriori algorithm consist of two parts

Part 1: Find all frequent item sets: Item sets that exceeds minimum support

Part 2: Generating strong association rules from frequent itemsets (Association rule that meet minimum confidence)

PART 1: Finding frequent itemsets

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.

The Apriori property. All non empty subsets of a frequent itemset must also be frequent.

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

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. This set is denoted L1. L1 is used to fnd L2, the frequent 2-itemsets, which is used to fnd 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.

A two-step process is followed to find frequent items 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:** Ck is a superset of Lk , that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in Ck . A database scan to determine the count of each candidate in Ck would result in the determination of Lk (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to Lk). Ck , however, can be huge, and so this could involve heavy computation. To reduce the size of Ck , 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 Lk−1, then the candidate cannot be frequent either and so can be removed from Ck .

PART 2: 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 strongassociation rules satisfy both minimum support and minimum confidence). This can be done using Equation for confidence.

Confidence ($X \Rightarrow Y$)= *(support of X and Y together)/(support of X)*= $P(X \cap Y) / P(X) = P(Y/X)$

.

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;
```

Figure 6.4 Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

Example Problem 1:

Based on the AllElectronics transaction database, D, of Table 6.1. Find association rule present in the database by considering Minimum support of 20% and confidence of 75%?.

Based on the AllElectronics transaction database, D, of Table 6.1. Find association rule present in the database by considering Minimum support of 20% and confidence of 75%?.

Table 6.1 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

Solution:

There are nine transactions in this database, that is, n = 9.

Set of Items, I={I1,I2,I3,I4,I5}

Part 1: Frequent Item Set generation

Step 1- E 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.

Scan *D* for count of each candidate →

C_1

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

Step 2- The set of frequent 1-itemsets, L1, can then be determined Minimum support count required is 2, that is, min sup = 2. (Here, support is 2/9 = 22%.). All Items in C1 are qualified

Compare candidate support count with minimum support count →

L_1

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

Step 3- 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.

Generate C_2 candidates from L_1 →

C_2

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

Step 4. Next, the transactions in D are scanned and the support count of each candidate itemset in C2 is calculated

Scan D for count of each candidate →

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

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

Compare candidate support count with minimum support count →

L_2

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

Step 6. 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}

C3
Items
{I1, I2, I3},
{I1, I2, I5},
{I1, I3, I5},
{I2, I3, I4},
{I2, I3, I5}
{I2, I4, I5}

Step 7. The transactions in D are scanned to determine support count of C3

C3	
Items	Sup count
{I1, I2, I3},	2
{I1, I2, I5},	2
{I1, I3, I5},	1
{I2, I3, I4},	0
{I2, I3, I5}	1
{I2, I4, I5}	0

Step 8: Find 3 frequent item set, L3. (candidate 3-itemsets in C3 having minimum support.)

Compare candidate support count with minimum support count →

L_3

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

Step 9: Generation of the set of the candidate 4-itemsets, C4. From the join step, we first get C4 = L3 ⋈ L3

C4	
Items	Sup count
{I1, I2, I3,I5 },	1

Step 10: Find 4 frequent item set, L4. (C3 having minimum support)

NULL list. Stop Frequent Item set generation.

PART 2: Generating Association Rules from Frequent Item sets

- Minimum confidence required 75 %
- **Confidence** (***X*** ⇒ ***Y)=*** *(support of X and Y together)/(support of X)=* **P(X∩*Y*)/ P(X)= P(*Y*|*X*)**

Consider L3={**{I1, I2, I3},{I1, I2, I5}**}

Association rules formed from frequent Item set **{I1, I2, I3},**

I1^I2⇒ I3, confidence = 2/4 = 50%
I1^I3⇒ I2, confidence = 2/4 = 50%
I2^I3⇒ I1, confidence = 2/4 = 50%
I1⇒ I2^I3, confidence = 2/6 = 33%
I2⇒ I1^I3, confidence = 2/7 = 29%
I3⇒ I1^I2, confidence = 2/6 = 33%

Association rules formed from frequent Item set **{I1, I2, I5},**

$I1 \wedge I2 \Rightarrow I5$, $confidence = 2/4 = 50\%$
$I1 \wedge I5 \Rightarrow I2$, $confidence = 2/2 = 100\%$
$I2 \wedge I5 \Rightarrow I1$, $confidence = 2/2 = 100\%$
$I1 \Rightarrow I2 \wedge I5$, $confidence = 2/6 = 33\%$
$I2 \Rightarrow I1 \wedge I5$, $confidence = 2/7 = 29\%$
$I5 \Rightarrow I1 \wedge I2$, $confidence = 2/2 = 100\%$

Association rules that satisfy minimum confidence are {I1^I5 I2 ,I2^I5 I1, I5 I1^I2}

Major drawbacks of Apriori Algorithm

1. The number of candidate itemset grows quickly and result in large candidate set. For example, if there are 10^4 frequent 1-itemsets (L1), the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets(C2).
2. The Apriori algorithm requires many scans of the database.

FP-growth Association Rule Mining Algorithm

"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 (a special kind of projected database), 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.

Note: FPgrowth algorithm find set of frequent itemsets without candidate generation process

Advantages of FP growth algorithm:-

1. Faster than apriori algorithm

2. No candidate generation

3. Only two passes over dataset

Disadvantages of FP growth algorithm:-

1. FP tree may not fit in memory

2. FP tree is expensive to build

Fp growth algorithm example

FP-growth Association Rule Mining Algorithm

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); }
```

Figure 6.9 FP-growth algorithm for discovering frequent itemsets without candidate generation.

Example Problem:

Based on the AllElectronics transaction database, D, of Table 6.1. Find frequent pattern present in the database using FP Growth Algorithm by considering minimum support of 20% and confidence of 75%?.

Table 6.1 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

Solution:

FP-Tree corresponding to given Dataset is shown in below figure

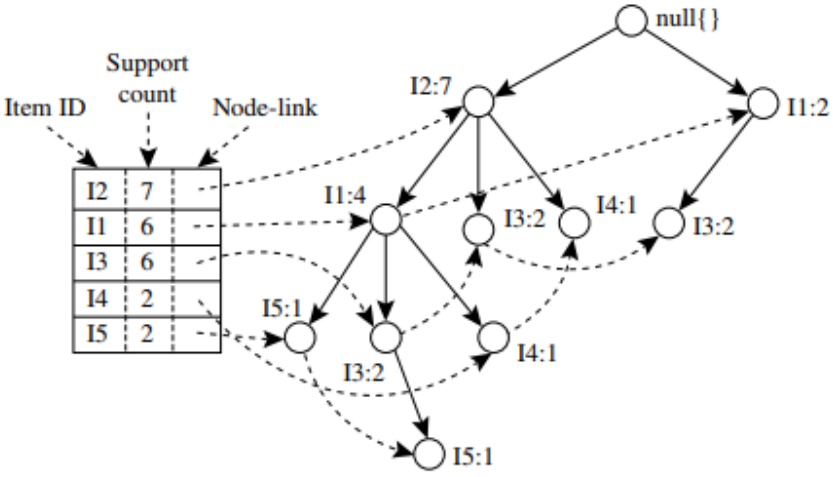


Figure 6.7 An FP-tree registers compressed, frequent pattern information.

Frequent patterns generated from the given Dataset is shown in below figure

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}

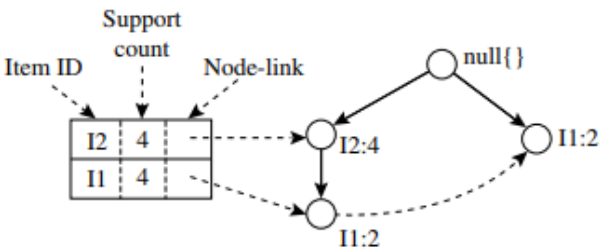


Figure 6.8 The conditional FP-tree associated with the conditional node I3.

What is Cluster Analysis?

- **Cluster:** A collection of data objects
  - **similar** (or related) to one another within the same group
  - **dissimilar** (or unrelated) to the objects in other groups
  -
- **Cluster analysis** (or *clustering*, *data segmentation, …*)
  - Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters
- **Unsupervised learning:** no predefined classes (i.e., *learning by observations* vs. learning by examples: supervised)

- Typical applications of Cluster analysis
  - As a **stand-alone tool** to get insight into data distribution
  - As a **preprocessing step** for other algorithms

Clustering for Data Understanding and Applications

- Biology: taxonomy of living things: kingdom, phylum, class, order, family, genus and species
- Information retrieval: document clustering
- Land use: Identification of areas of similar land use in an earth observation database
- Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- City-planning: Identifying groups of houses according to their house type, value, and geographical location
- Earth-quake studies: Observed earth quake epicenters should be clustered along continent faults
- Climate: understanding earth climate, find patterns of atmospheric and ocean
- Economic Science: market research

Clustering as a Preprocessing Tool (Utility)

- Summarization:
  - Preprocessing for regression, PCA, classification, and association analysis
- Compression:
  - Image processing: vector quantization
- Finding K-nearest Neighbors
  - Localizing search to one or a small number of clusters
- Outlier detection
  - Outliers are often viewed as those “far away” from any cluster

Quality: What Is Good Clustering?

- A good clustering method will produce high quality clusters
  - **high intra-class similarity**: cohesive within clusters
  - **low inter-class similarity**: distinctive between clusters

- The quality of a clustering method depends on
  - the similarity measure used by the method
  - its implementation, and

- Its ability to discover some or all of the hidden patterns

Major Clustering Approaches

In general, the major clustering methods can be classified into the following categories.

- Partitioning Methods
- Hierarchical Methods
- Density-Based Methods
- Grid-Based Methods

- **Partitioning approach:**
  - Construct various partitions and then evaluate them by some criterion, e.g., minimizing the sum of square errors
  - Given a database of n objects, a partitioning method constructs k partitions of the data object, where each partition represents a cluster .
  - That is, it clusters the data into k groups, which together satisfy the following requirements:
    (1) each group must contain at least one object, and
    (2) Each object must belong to exactly one group
  - Typical methods: **k-means, k-medoids, CLARANS**

- **Hierarchical approach:**
  - A hierarchical method creates a hierarchical decomposition of the given set of data objects.
  - A hierarchical method can be classified as being either agglomerative or divisive
  - **The agglomerative approach**, also called the bottom-up approach, starts with each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds.
  - **The divisive approach**, also called the top-down approach, starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds.
  - Typical methods: **Diana, Agnes, BIRCH, CAMELEON**

- **Density-based approach:**
  - Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes.
  - Density-based clustering methods have been developed based on the notion of **density.**
  - Based on connectivity and density functions
  - Typical methods: **DBSACN, OPTICS, DenClue**

- **Grid-based approach:**

- Grid-based methods quantize the ***object space into a finite number of cells that form a grid structure.***
- All of the clustering operations are performed on the grid structure (i.e., on the quantized space).
- The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space.
- Typical methods: STING, WaveCluster, CLIQUE

- **Model-based:**
  - Model-based methods hypothesize a ***model for each of the clusters*** and find the best fit of the data to the given model.
  - Typical methods: **Expectation-Maximization (EM), SOM, COBWEB**
- Frequent pattern-based:
  - Based on the analysis of ***frequent patterns***
  - Typical methods**: p-Cluster**
- User-guided or constraint-based:
  - Clustering by considering user-specified or application-specific constraints
  - Typical methods: **COD (obstacles), constrained clustering**

Module 5

Association Rules Mining :- Concepts, Apriori and FP-Growth algorithm.

Cluster Analysis :- Introduction, Concepts, Types of data in cluster analysis, Categorization of clustering methods, Partitioning method :- K-means and K-Medoid clustering

Cluster Analysis

→ A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters

→ The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering

→ clustering is also called data segmentation

→ clustering can also be used for outlier detection.

→ clustering is an example of unsupervised learning

→ Requirements of clustering in data mining

1. Scalability : Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions of objects. Clustering on a sample

of a given large data set may lead to biased results. Highly scalable clustering algorithms are needed.

2. Ability to deal with different types of attributes

Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal) and ordinal data or mixtures of these data types.

3. Discovery of clusters with arbitrary shape.

Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms that can detect clusters

of arbitrary shape.

4. Minimal requirements of domain knowledge to determine input parameters.

Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive to input parameters. Parameters are often difficult to determine, especially for datasets containing high dimensional objects.

5. Ability to deal with noisy data:-

Most real-world database contain outliers or missing, unknown or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

6. Incremental clustering and insensitivity to the order of input records

Some clustering algorithms can not

Incorporate newly inserted data to existing clustering structure and instead, must determine a new clustering from scratch. Some clustering algorithms are sensitive to the order of input data. That is, given a set of data objects, such an algorithm may return dramatically different clustering depending on the order of presentation of input objects. It is important to develop incremental clustering algorithms and algorithms that are insensitive to the order of input.

7. High dimensionality:

A database or a data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at handling low dimensional data, involving only two or three dimensions. Finding clusters of data objects in high dimensional space is challenging

8. Constraint-based clustering:

Real world applications may need to perform clustering under various kinds of constraints. A challenging task is to find groups of data with good clustering behaviour that satisfy specified constraints.

9. Interpretability & Usability:

Users ~~except~~ expect clustering results should be interpretable, comprehensible and usable.

Types of data in Cluster Analysis

→ Main memory based clustering algorithms typically operate on either of the following two data structures.

1. Data Matrix (Object-by-Variable Structure)

This represents n objects, such as persons with p variables (also called measurements or attributes), such as age, height, weight, gender and so on.

The structure is in the form of a relational table or n by p matrix (n objects x p variables)

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2f} & \cdots & x_{2p} \\ \vdots & & & & \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

~~2 Similarity~~

2. Dissimilarity Matrix [object by object structure]

This stores a collection of proximities that are available for all pairs of n object. It is often represented by an n by n table.

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

d(i,j) is the measured difference or dissimilarity between object i and j. In general d(i,j) is a nonnegative no that close to 0 when objects i and j are highly similar or near each other.

→ Rows and columns of the data matrix represent different entities while those of the dissimilarity matrix represent the same entity.

So data matrix is called two-mode matrix dissimilarity matrix is called one-mode matrix

→ Many clustering algorithms operate on dissimilarity matrix. If data are represented in the form of a data matrix, it can first be transformed into a dissimilarity matrix before applying such clustering algorithms.

Different Types of data in cluster analysis

1. Interval-Scaled Variables.

→ Interval scaled variables are continuous measurements of a roughly linear scale.

eg: height, weight, temperature.

→ The measurement unit used can affect the clustering analysis.

→ To avoid the dependence ~~of~~ on the choice of measurement unit, the data should be standardized.

→ To standardize measurements, one choice is to convert the original measurements to unitless variables. Given measurements for a variable f, this can be performed as follows.

1. Calculate the mean absolute deviation s_f

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

where $x_{1f}, x_{2f} \ldots x_{nf}$ are n measurements of f and m_f is the mean value of f.

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

2. Calculate the standardized measurement or z-score

$$z_{if} = \frac{x_{if} - m_f}{s_f}$$

→ After standardization or without standardization in certain applications, the dissimilarity (or similarity) between the objects described by interval scaled variables is typically computed based on the distance between each pair of object. The most popular distance measure is Euclidean distance

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots (x_{in} - x_{jn})^2}$$

where $i = (x_{i1}, x_{i2} \ldots x_{in})$ & $j = (x_{j1}, x_{j2} \ldots x_{jn})$ are two n-dimensional data objects.

Another well-known metric is Manhattan (city block) distance

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{in} - x_{jn}|$$

Both the Euclidean and Manhattan distance satisfy the following mathematic requirements of a distance function:

1. $d(i,j) \geq 0$, Distance is a nonnegative no.
2. $d(i,i) = 0$, The distance of an object to itself is 0.
3. $d(i,j) = d(j,i)$; Distance is a symmetric function.
4. $d(i,j) \leq d(i,h) + d(h,j)$

Going directly from object i to object j in space is no more than making a detour over any other object h.

→ Minkowski distance is a generalization of both Euclidean distance and Manhalten distance

$$d(i,j) = \left(|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \dots |x_{in} - x_{jn}|^p \right)^{1/p}$$

P is positive integer

P = 1 = Mahalten distance

P = 2 = Euclidean distance

→ If a variable is assigned a weight according to its perceived importance, the weighted Euclidean distance can be computed as

$$d(i,j) = \sqrt{w_1 |x_{i1} - x_{j1}|^2 + w_2 |x_{i2} - x_{j2}|^2 + \dots w_m |x_{in} - x_{jn}|^2}$$

2. Binary Variables.

A binary variable has only two states: 0 or 1, where 0 means that variable is absent and 1 means that it is present.

→ How to compute dissimilarity between two binary variables?

* One approach involves computing a dissimilarity matrix from the given binary data.

* If all binary variables are thought of as having the same weight, we can create a contingency table

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

q = no of variables that equal 1 for both object i and j.

r = no of variables that equal 1 for object i but that are for object j

s = no of variables that are equal 0 for object i and equal 1 for object j

t = no of variables that are equal 0 for both object i and j.

p = q+r+s+t → total no of variables.

→ A binary variable is symmetric, if both of its states are equally valuable and carry the same weight:

eg: Gender < Male 0
female 1.

→ Dissimilarity that is based on symmetric binary variable is called symmetric binary dissimilarity

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

→ A binary variable is asymmetric if the outcome of the states are not equally important.

eg: Cancerous — non cancerous.

→ The dissimilarity based on such variables is called asymmetric binary dissimilarity.

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

→ asymmetric binary similarity blw object i & j

$$Sim(i,j) = \frac{q}{q+r+s} = 1 - d(i,j)$$

* Sim(i,j) - Jaccard coefficient.

eg:- Suppose that a patient record table contains the attribute name, gender, fever, cough, test-1, test-2, test-3 and test-4 Gender is symmetric attribute & remaining are asymmetric

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	Y	N	N	N	N

→ If distance is compiled based on asymmetric variables.

$$d(\text{Jack}, \text{Mary}) = \frac{0+1}{2+0+1} = 0.33$$

$$d(\text{Jack}, \text{Jim}) = \frac{1+1}{1+1+1} = 0.67$$

$$d(\text{Mary}, \text{Jim}) = \frac{1+2}{1+1+2} = 0.75$$

3. Categorical Variables.

A categorical variable is a generalization of the binary variable in that it can take on more than two states.

eg: color.

→ The state of the categorical variables can be denoted by letters, symbols or a set of integers.

→ Dissimilarity computed between objects described by categorical variables is

$$d(i,j) = \frac{p-m}{p}$$

m = no of matches (no of variables for which i and j are in same state)

P = total no of variables.

eg:

Object identifier	test-1 (categorical)
Cases 1	Code A
2	code B
3	code C
4	Code A

P = 1

d(1,1) = 0

d(2,1) = 1 $\left(\frac{P-0}{P} = \frac{1-0}{1} = 1\right)$

Dissimilarity matrix is

$$\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 1 & 1 & 0 & \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

4. Ordinal Variables.

→ A discrete ordinal variable resembles categorical variable, except that the M states of the ordinal value are ordered in a meaningful sequence.

eg: Professional rank - Assistant Professor
Associate Professor
Professor

→ A continuous ordinal variable looks like a set of continuous data of an unknown scale; ie the relative ordering of the values is essential but their actual magnitude is not.

eg: relative ranking in a sports (gold silver bronze)

→ Suppose that f is a variable from a set of ordinal variables describing n objects. The dissimilarity computation with respect to f involves the following steps:

1. The value of f for the i^{th} object is x_{if} and f has M_f ordered states, representing $1, \ldots, M_f$. Replace each x_{if} by its corresponding rank

$$r_{if} \in \{1, \ldots, M_f\}.$$

2. Since each ordinal variable can have a different number of states, it is often necessary to map the range of each variable onto $[0.0, 1.0]$, so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the i^{th} object in the f^{th} variable by

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

3. Dissimilarity can then be compiled using any of the distance measures.

eg:

Object Identifier	test-2 (Ordinal)
1	excellent-③
2	fair ①
3	Good ②
4	Excellent ③

$M_f = 3$.

object	test2
1	3
2	2
3	1
4	3

Normalize:

object	test2
1	1
2	0
3	0.5
4	1

Dissimilarity matrix

$$\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1.0 & 0.5 & 0 \end{bmatrix}$$

5. Ratio - Scaled Variables.

A ratio scaled variable makes a positive measurement on a non linear scale, such as an exponential scale, approximately following the formula

Ae^{Bt} or Ae^{-Bt}

A, B - +ve constants

t - time

eg: growth of bacteria populations

How to compute ratio - scaled variable?

Three approaches are used.

1. Treat ratio-scaled variables like interval scaled variables.

2. Treat x_{if} as continuous ordinal data and treat their ranks as interval valued.

3. Apply logarithmic transformation to a ratio-scaled variable f having value x_{if} for object i by using formula $y_{if} = \log(x_{if})$. The y_{if} values can be treated as interval valued.

eg:

Object identifier	test 3 (ratio scaled)
1	445
2	22
3	164
4	1210

Take logarithmic transformation

1	2.65
2	1.34
3	2.21
4	3.08

Using Euclidean distance, dissimilarity ~~measure~~ matrix

$$\begin{bmatrix} 0 & & & \\ 1.31 & 0 & & \\ 0.44 & 0.87 & 0 & \\ 0.43 & 1.74 & 0.87 & 0 \end{bmatrix}$$

6 Variables of Mixed Types

→ One approach to calculate dissimilarity between objects of mixed variable types is to group each kind of variable together, performing a separate cluster analysis for each variable type.

→ A more preferable approach is to process all types of variables together, performing a single cluster analysis. One such technique combine the different variables into a single dissimilarity matrix, bringing all of the variables onto a common scale of the interval [0.0, 1.0]

→ Q(4) Suppose that the dataset contains p variables of mixed type. The dissimilarity d(i,j) between objects i & j is defined as

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

f represents a variable where the indicator $\delta_{ij}^{(f)} = 0$ if either

(1) x_{if} or x_{jf} is missing

(2) $x_{if} = x_{jf} = 0$ and variable f is asymmetric binary.

otherwise $\delta_{ij}^{(f)} = 1$.

* The contribution of variable f to the dissimilarity between i and j, that is $d_{ij}^{(f)}$ is computed dependent on its type.

1. If f is interval based

$$d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{\max_h x_{hf} - \min_h x_{hf}}$$

where h runs over all non missing objects for variable f.

2. If f is binary or categorical

$d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$

otherwise $d_{ij}^{(f)} = 1$

3. If f is ordinal - Compute the ranks r_{if} and $z_{if} = \frac{r_{if} - 1}{M_f - 1}$ and treat z_{if} as interval-scaled.

4. If f is ratio scaled :- either perform logarithmic transformation and treat the transformed data as interval based or treat f as continuous ordinal data, Compute r_{if} and z_{if} and then treat z_{if} as interval based.

eg:

Object identifier	test 1 (categorical)	test 2 (ordinal)	test 3 (ratio-scaled)
1	code A	excellent	445
2	code B	fair	22
3	code C	good	164
4	code A	Excellent	1210

Dissimilarity Matrix for test 1

$$\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 1 & 1 & 0 & \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Dissimilarity matrix for test 2

$$\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1.0 & 0.5 & 0 \end{bmatrix}$$

For test 3, logarithmic transformation is applied and we got 2.65, 1.34, 2.21 and 3.08

$$\max_h x_h = 3.08$$

$$\min_h x_h = 1.34.$$

We then normalize the dissimilarity matrix

$$\begin{bmatrix} 0 & & & \\ (2.65-1.34) & 0 & & \\ 1.31 & & & \\ |2.21-2.65| & (2.21-1.34) & 0 & \\ 0.44 & 0.87 & & 0 \\ 0.43 & 1.74 & 0.87 & 0 \end{bmatrix}$$

by dividing by $3.08 - 1.34 = 1.74$

Finally dissimilarity matrix for test 3 is

$$\begin{bmatrix} 0 & & & \\ .75 & 0 & & \\ .25 & .50 & 0 & \\ .25 & 1.00 & 0.50 & 0 \end{bmatrix}$$

Now we can use these three dissimilarity matrices in

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

$$d(2,1) = \frac{\delta_{21}^{(test-1)} \times d_{(2,1)}^{(test-1)} + \delta_{21}^{(test-2)} d_{(2,1)}^{(test-2)} + \delta_{21}^{(test-3)} d_{(2,1)}^{(test-3)}}{\delta_{21}^{(test-1)} + \delta_{21}^{(test-2)} + \delta_{21}^{(test-3)}}$$

$$d(2,1) = \frac{1 \times 1 + 1 \times 1 + 1 \times .75}{3}$$

$$= .92$$

$$d(3,1) = \frac{1 \times 1 + 1 \times .5 + 1 \times .25}{3}$$

$$= \frac{1 + .5 + .25}{3}$$

$$= .58$$

Final Matrix will be

$$\begin{bmatrix} 0 & & & \\ .92 & 0 & & \\ .58 & .67 & 0 & \\ .08 & 1.00 & .67 & 0 \end{bmatrix}$$

7. Vector objects

→ Popular similarity function to compare two vectors x and y is cosine measure

$$S(x,y) = \frac{x^t \cdot y}{\|x\| \|y\|}$$

x^t = transposition of vector x

$\|x\|$ $\|y\|$ Euclidean norms

if $x = (x_1, x_2 \ldots x_n)$

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \ldots x_n^2}$$

S is the cosine of angle between x & y.

eg: $x = (1,1,0,0)$ $\quad x = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ $\quad \|x\| = \sqrt{1^2+1^2}$

$y = (0,1,1,0)$ $\quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$ $\quad \|y\| = \sqrt{0^2+1^2+1^2+0^2} = \sqrt{2}$

$$S(x,y) = \frac{x^t y}{\|x\| \|y\|}$$

$$= \frac{0+1+0+0}{\sqrt{2}\sqrt{2}}$$

$$= .5$$

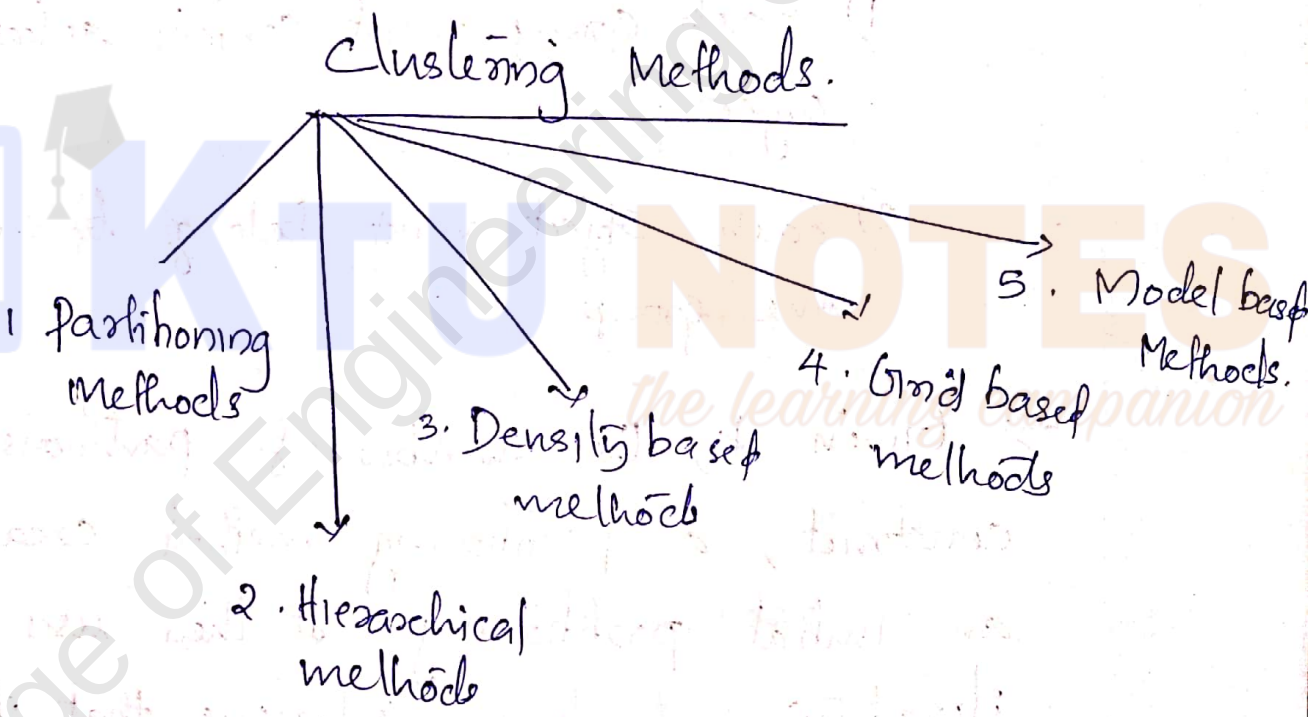
$[1,1,0,0]_{1\times 4} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}_{4\times 1}$ $\quad 1+1$

A simple variable to above measure is

$$s(x,y) = \frac{x^t y}{x^t x + y^t \cdot y - x^t x}$$

It is called tanimoto coefficient or tanimoto distance.

Categorization of clustering Methods



1. Partitioning Methods.

→ Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster and $k \leq n$.

→ That is, it classifies the data into k groups, which together satisfy the following requirements.

1) Each group must contain atleast one object.

2) Each object must belong to exactly one group.

→ Given k, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses the iterative relocation technique that attempts to improve the partitioning by moving objects from one group to another.

→ The general criterion of a good partitioning is that objects in the same cluster are close or related to each other whereas objects of different clusters are far apart or very different.

eg: k-means algorithm

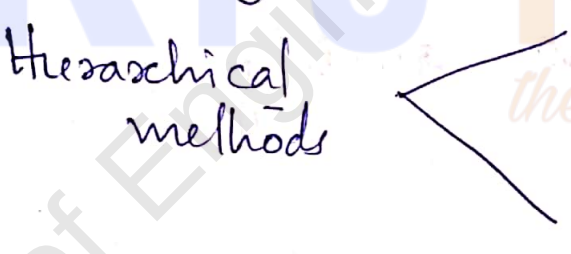
cluster is represented by the mean value of objects in the cluster

k-medoids algorithm

Each cluster is represented by one of the objects located near the center of the cluster.

2. Hierarchical Methods-

A hierarchical method creates a hierarchical decomposition of the given set of data objects



→ Agglomerative approach also known as bottom up approach, starts with each objects forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one or a termination condition hold.

→ The divisive approach is also called a top down approach, starts with all of the objects in the same cluster. In each iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster or until a termination condition holds.

→ Hierarchical methods suffer from the fact that once a step is done, it can never be undone. The such techniques cannot correct erroneous decisions.

→ There are two approaches for improving the quality of hierarchical clustering.

1. Perform careful analysis of object linkages at each hierarchical partitioning

2. Integrate hierarchical agglomeration and other approaches by first using a hierarchical agglomerative algm to group objects into microclusters and then performing microclustering on the microclusters using another methods.

3. Density-based Methods

→ Most partitioning methods clusters objects based on their distance between the objects. Such methods can only find spherical shaped clusters.

→ General approach in density based methods is to continue growing the given cluster as long as the density (no of objects or data points) in the neighborhood exceeds a some threshold; that is for each data point within a given cluster, the neighborhood of a given radius has to contain atleast a minimum no of points.

eg: DBSCAN, OPTICS, DENCLUE

4. Grid-based Methods

Grid based methods quantize the object space into finite no of cells that form a grid structure. All the clustering operations are performed on the grid structure (on quantized space).

The main advantage of this approach is fast processing time, which is independent of the no of data objects and depends only on the no of cells in each dimension in the quantized space

eg: STING

WaveCluster - both grid based & densitybased

5. Model based Methods

→ Model based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model.

→ A model based algm may locate clusters by constructing a density function that reflects the spatial distribution of the data points.

eg: EM algm - peforom expectation - maximization analysis

COBWEB - performs probability analysis

SOM - Self organizing feature map

Clustering tasks :- that requires special attention are

1. Clustering high-dimensional data

→ clustering high dimensional data is challenging due to the curse of dimensionality

→ Many dimensions may not be relevant. As the no of dimensions increases the data become increasingly sparse so that distance measurement between pairs of points become meaningless and the average density of points anywhere in data is likely to be low.

→ CLIQUE and PROCLUS are two subspace clustering methods, which search for clusters in subspaces of data rather than entire data space.

→ Frequent pattern based clustering, extracts distinct frequent patterns among the subset of dimensions that occurs frequently.

eg: pCluster

2. Constraint based clustering:

→ performs clustering by incorporation of user specified or application oriented constraints.

→ A constraint expresses a user's expectation or describes properties of the desired clustering results.

→ Various kinds of constraints can be specified, either by a user or as per application requirements.

Partitioning Methods

K-means Method :- Centroid based Technique

→ The K-means algorithm takes the input parameter k, and partitions a set of n objects into k clusters so that the resulting intracluster similarity is high but the intercluster similarity is low. cluster similarity is measured in regard to the mean value of

the objects is a cluster, which can be viewed as the cluster's centroid or centre of gravity.

Algorithm:

Input

k: the no of ~~map~~ clusters

D: Data set containing n objects

Output:

A set of k clusters

Method

(1) arbitrarily choose k objects from D as the initial cluster centers.

(2) repeat

(3) (re) assign each object to the cluster to which the object is similar, based on the mean value of the objects in the cluster

(4) update the cluster means, calculate the mean value of the objects for each cluster

(5) Until no change.

First, it randomly selects k of the objects, each of which initially represents a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and cluster mean. It then computes the new mean for each cluster. This process iterates until ~~convergence~~ the criterion function converges.

Typically square error criterion is used

$$E = \sum_{i=1}^{k} \sum_{P \in C_i} |P - m_i|^2$$

P = represents an element in cluster C_i

m_i = mean of cluster C_i.

→ Computational complexity

$$O(nkt)$$

n = no of objects

k = no of clusters

t = no of iterations.

Disadvantages

→ Can not be applied for data containing categorical attributes

→ Users need to specify the no of clusters in advance

→ Not suitable for discovering clusters with nonconvex shapes or clusters of very different size

→ It is sensitive to noise and outlier datapoints

Variants of K-means algorithm

1. K-modes Method :- which extends the K-means paradigm to cluster categorical data by replacing means of the ~~data~~ clusters with modes, using new dissimilarity measures to deal with categorical objects. and a frequency based to update modes of clusters.

2. EM [Expectation - Maximization] algorithms extends K-means paradigm by assigning each object to clusters according to a weight representing its probability of membership.

→ How can we make k-means algorithm more scalable ?.

One approach is

→ 1. Identify 3 kinds of regions in data

1. Regions that are discardable

– An object is discardable if its membership in a cluster is ~~asset~~ ascertained.

2. Regions that are compressible

– An object that is not discardable but it belongs to a subcluster

3. Regions that must be retained in main memory

Objects that are not discardable & compressible.

* A data structure known as a clustering feature is used to summarize objects that have been discarded or ~~summarized~~ Compressed

* To achieve scalability, the iterative algm only uses clustering feature of compressed objects & objects that have to be retained in main mly

→ The another approach is to first groups nearby objects into microclusters and then performs k-means clustering on the microclusters.

Problems

1. $A_1(2,10)$ $A_2(2,5)$ $A_3(8,4)$
$B_1(5,8)$ $B_2(7,5)$ $B_3(6,4)$
$C_1(1,2)$ $C_2(4,9)$

Cluster the points into 3 clusters.
Initial centers are A_1, B_1 & C_1.
The distance function is Euclidean distance.

$A_1(2,10)$ $B_1(5,8)$ $C_1(1,2)$

$d(A_1, A_2) = \sqrt{0+5^2} = \sqrt{25}$

$d(A_1, B_2) = \sqrt{5^2+5^2} = \sqrt{50}$

$d(A_1, A_3) = \sqrt{6^2+6^2} = \sqrt{72}$

$d(A_1, B_3) = \sqrt{4^2+6^2} = \sqrt{60}$

$d(A_1, C_2) = \sqrt{2^2+1^2} = \sqrt{5}$

$d(B_1, A_2) = \sqrt{3^2+3^2} = \sqrt{18}$

$d(B_1, A_3) = \sqrt{3^2+4^2} = \sqrt{25}$

$d(B_1, B_2) = \sqrt{2^2+3^2} = \sqrt{13}$

$d(B_1, B_3) = \sqrt{1^2+4^2} = \sqrt{17}$

$d(B_1, C_2) = \sqrt{2}$

$d(C_1, C_2) = \sqrt{3^2 + 7^2} = \sqrt{58}$

$d(C_1, A_2) = \sqrt{1^2 + 3^2} = \sqrt{10}$

$d(C_1, A_3) = \sqrt{7^2 + 2^2} = \sqrt{53}$

$d(C_1, B_2) = \sqrt{6^2 + 3^2} = \sqrt{45}$

$d(C_1, B_3) = \sqrt{5^2 + 2^2} = \sqrt{29}$

clusters

$\{A_1\}$, Mean :- (2, 10)

$\{B_1, B_2, A_3, B_3, C_2\}$ = Mean = (6, 6)

$\{C_1, A_2\}$ Mean :- (1.5, 3.5)

Final clusters

$\{A_1, C_2, B_1\}$

$\{A_3, B_2, B_3\}$

$\{C_1, A_2\}$

K-medoids Method:- Representative Object Based Technique

K-means algorithm is sensitive to outliers. To diminish such sensitivity, instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is clustered with the representative object to which it is most similar. The partitioning method is then performed based on the principle of minimizing the sum of dissimilarities between each object and its corresponding reference point. That is, an absolute error criterion is used

$$E = \sum_{j=1}^{k} \sum_{p \in C_j} |p - o_j|$$

Medoid :- representative object in a set, it has smallest average dissimilarity to other objects in the set.

p = point in the space representing a given object in cluster C_j

o_j = representative object of C_j

The algorithm iterates until, each representative element is actually medoid, or most centrally located object of its cluster. This is the basis of the k-medoids method

Working of k-medoid algorithm

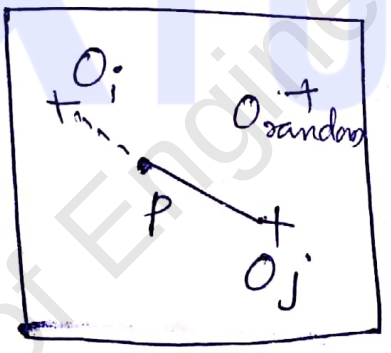
→ The initial representative objects are chosen arbitrarily.

→ The iterative process of replacing representative objects by non representative objects continues as long as quality of resulting cluster is improved.

→ The quality is estimated using a cost function that measures the average dissimilarity between an object and the representative object of its cluster.

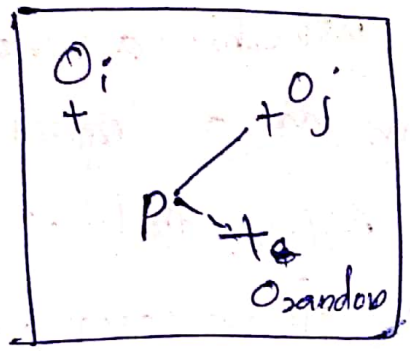
→ To determine whether a nonrepresentative object O_{random} is a good replacement for a current representative object O_j, the following four cases are examined

Case 1: P is currently belongs to representative object O_j. If O_j is replaced by O_{random} as a representative object and P is closest to one of the other representative object O_i, $i \neq j$, then P is reassigned to O_i.

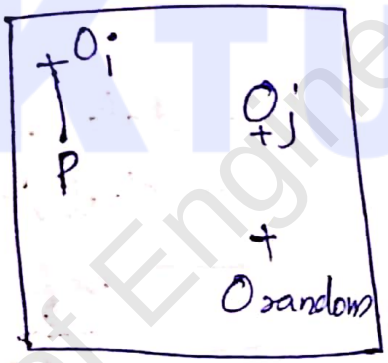


• data object
+ cluster center
—— before swapping
---- after swapping.

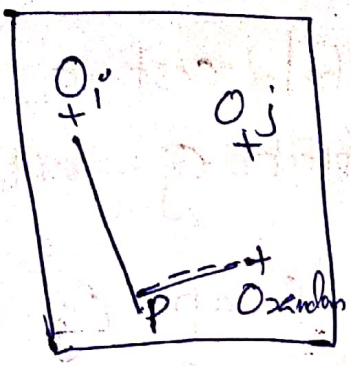
Case 2:- P is currently belongs to representative object O_j. If O_j is replaced by O_{random} as a representative object and P is closest to O_{random} then P is reassigned to O_{random}.



Case 3 : P is currently belong to representative object O_i. $i \neq j$. If O_j is replaced by O_{random} as a representative object and P is still closest to O_i, then the assignment does not change.



Case 4 : P is currently belongs to representative object O_i, $i \neq j$. If O_j is replaced by O_{random} as a representative object and P is closest to O_{random} then P is reassigned to O_{random}



PAM (Partitioning about Around Medoids)

→ It is the first k-medoid algm introduced

Algorithm

Input

k : the no of clusters

D : a data set containing n objects

Output : A set of k clusters.

Method

(1) arbitrarily choose k objects in D as the initial representative objects or seeds.

(2) repeat

(3) assign each remaining object to the cluster with the nearest representative object.

(4) randomly select a nonrepresentative object O_{random}.

(5) compute the total cost S, of swapping representative object O_j with O_{random}.

(6) if $S < 0$ then swap O_j with O_{random} to form the new set of k representative ~~element~~ objects

(7) Until no change.

It attempts to determine k partitions for n objects. After an initial random selection of k representative objects, the algm repeatedly tries to make a better choice of cluster representatives. All of the possible pairs of objects are analyzed, where one object in each pair is considered a representative object and the other is not. The quality of the resulting clustering is calculated for each such combination. An object O_j is replaced with object causing the greatest reduction in error.

The set of best objects for each cluster in one iteration forms the representative objects for the next iteration. The final set of representative objects are the representative medoids of the clusters.

Complexity of each iteration - $O(k(n-k)^2)$

Advantages of k-medoid algm

→ More robust than k-means in the presence of noise and outliers.

Drawback

→ Processing is costly.

In both k-means & k-medoids, user need to specify no of clusters k.

Association Rules Mining

Frequent itemsets, closed itemsets and Association Rules

Let $I = \{I_1, I_2, \ldots I_m$ be a set of items. Let D, be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with an identifier TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$.

→ An association Rule is an implication of the form

$A \Rightarrow B$ where $A \subset I$

$B \subset I$

$A \cap B = \phi$

→ The support s of the association rule $A \Rightarrow B$ is the percentage of transactions in D that contains $A \cup B$, [both A & B)

This is taken to be the probability $A \cup B$

→ The confidence c of the association rule $A \Rightarrow B$ is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability $P(B|A)$.

→ Rules that satisfy both minimum support and minimum confidence is called strong.

→ A set of items is referred to as an itemset. An itemset that contains k-items is a k-itemset.

→ The occurance frequency of an itemset is the no of transactions that contains the itemset.

It is also known as frequency, support count or count of the itemset.

→ Support $(A \Rightarrow B) = P(A \cup B)$ ← This is called relative support.

Occurance frequency is called absolute support.

→ If the relative support of an itemset I satisfies a prespecified minimum support threshold (ie, the absolute support of I satisfies the corresponding minimum support count threshold), then I is a frequent itemset. The set of frequent k-itemset is denoted by L_k.

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

$$= \frac{\text{Support}(A \cup B)}{\text{Support}(A)}$$

$$= \frac{\text{Support\_count}(A \cup B)}{\text{Support\_count}(A)}$$

→ A typical example of frequent itemset mining is market basket Analysis

This process analyzes the customer buying habits by finding associations between the different items that customers place in their shopping basket

" or which items are frequently purchased together"

→ Association rule mining can be viewed as a two step process

1. Find all frequent itemsets:
2. Generate strong association rules from the frequent itemsets.

→ An itemset X is closed in a dataset S if there exists no proper super-itemset Y such that Y has the same support count as X in S.

→ An itemset X is a closed frequent itemset in S if X is both closed and frequent in S.

→ An itemset X is maximal frequent itemset in set S if X is frequent and there exists no super itemset Y such that $X \subset Y$ and Y is frequent in S.

The Apriori Algorithm :- Finding Frequent itemsets using candidate Generation

→ First the set of frequent 1-itemsets is found by scanning the database to accumulate the count of each item and collecting those items that satisfy minimum threshold. The resulting set is denoted by L_1.

→ L_1 is used to find L_2 and so on until no more frequent k itemsets can be found.

→ Finding of each L_k requires one full scan of the database.

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

~~Apriori~~

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

ie. If I is not frequent, and if an item A is added to the itemset I, then $I \cup A$ is also not frequent.

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

→ How L_{k-1} is used to find L_k, $k \geq 2$.

A two step process is followed.

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 j^{th} item in l_i. Apriori assumes that items in a transaction or itemset are sorted in lexicographic order. For (k-1) itemset, $l_i[1] < l_i[2] \ldots < l_i[k-1]$.

The join of $L_{k-1} \bowtie L_{k-1}$ is performed where members of L_{k-1} are joinable if their first (k-2) itemsets are in common.

ie members l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1] \wedge l_1[2] \wedge l_2[2] \ldots \wedge l_1[k-2] = l_2[k-2] \wedge l_1[k-1] < l_2[k-1])$

The resulting itemsets formed by joining l_1 & l_2 is $l_1[1], l_1[2], l_1[3] \ldots l_1[k-2], l_1[k-1], l_2[k-1]$

2. The prune step: C_k is superset of L_k, ie its members may or may not be frequent but all the frequent k-itemsets are included in C_k. A scan of the database to

determine the count of each candidate C_k would result in determination of L_k. [all candidates having count not less than minimum support count are frequent, therefore belong to L_k].

C_k, can be huge. To reduce size of C_k, the apriori property is used. Any k-1 itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence if (k-1) subset of a candidate k-itemset is not in L_{k-1}, then it cannot be frequent and can be removed from C_k.

Eg

TID	List of Item-IDs
T_1	I_1, I_2, I_5
T_2	I_2, I_4
T_3	I_2, I_3
T_4	I_1, I_2, I_4
T_5	I_1, I_3
T_6	I_2, I_3
T_7	I_1, I_3
T_8	I_1, I_2, I_3, I_5
T_9	I_1, I_2, I_3

1. In the first iteration, each item is a member of the set of candidate 1-itemsets C_1. The algm simply scans all of the transactions in order to count no of occurances of each item.

C_1

Itemset	Sup.Count
$\{I_1\}$	6
$\{I_2\}$	7
$\{I_3\}$	6
$\{I_4\}$	2
$\{I_5\}$	2

2. If min Then L_1 is formed by selecting candidate 1-itemsets satisfying minimum support (min-sup)

Eg: Here minimum support = 2

Then L_1

Itemset	Sup. count
$\{I_1\}$	6
$\{I_2\}$	7
$\{I_3\}$	6
$\{I_4\}$	2
$\{I_5\}$	2

3. To find frequent 2-itemset L_2.
$L_1 \bowtie L_1$ is performed to generate a candidate set of 2-itemsets.

C_2

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

No candidates are removed from C_2 during prune step because each subset of the candidates is also frequent.

4: Transactions in D are scanned and support count of each candidate itemset in C_2 is accumulated.

C_2 Itemset	Sup. count
$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_4\}$	1
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2
$\{I_3, I_4\}$	0
$\{I_3, I_5\}$	1
$\{I_4, I_5\}$	0

5. The set of frequent 2-itemsets L_2 is then determined, consisting of those candidate itemsets in C_2 having minimum support.

L_2

itemset	sup-count
$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2

6. Generation of Candidate 3-itemsets.

(a) To join $L_2 \bowtie L_2$.

[first $l_1[1] = l_2[1]$

[first L_{k-1} item should be common]

Here first 1 item should be common

$\{I_1, I_2, I_3\}$

$\{I_1, I_2, I_5\}$

$\{I_1, I_3, I_5\}$

$\{I_2, I_3, I_4\}$

$\{I_2, I_3, I_5\}$

$\{I_2, I_4, I_5\}$

b) Prune using Apriori property.

1. The 2-itemset subset of $\{I_1, I_2, I_3\}$ are $\{I_1, I_2\}$, $\{I_1, I_3\}$, $\{I_2, I_3\}$. These subsets are in L_2.
   So $\{I_1, I_2, I_3\}$ is in C_3

2. The 2-item subset of $\{I_1, I_2, I_5\}$ are also in L_2
   So $\{I_1, I_2, I_5\}$ is in C_3

3. For the remaining 3-itemsets, this property is not kept.
   So they are not included in C_3.

Finally C_3 is

itemsets
$\{I_1, I_2, I_3\}$
$\{I_1, I_2, I_5\}$

itemset	sup_count
$\{I_1, I_2, I_3\}$	2
$\{I_1, I_2, I_5\}$	2

7. L_3 is formed by selecting candidate 3-itemset having minimum support

L_3

itemset	sup_count
$\{I_1, I_2, I_3\}$	2
$\{I_1, I_2, I_5\}$	2

8 (ii) $L_3 \bowtie L_3$ is performed to generate C_4. This result is $\{I_1, I_2, I_3, I_5\}$

The subset $\{I_2, I_3, I_5\}$ is not in L_3.
So it is removed.

So $C_4 = \phi$, the algorithm terminates.

Algm: Apriori

Input:

D, a database of transactions.
min-sup: - the minimum support count threshold

Method

L_1 = find frequent-1-itemsets(D)
for ($k = 2$; $L_{k-1} \neq \phi$; $k++$) {
C_k = apriori-gen(L_{k-1});
for each transaction $t \in D$ { // scan D for count
C_t = subset(C_k, t) // get the subset of t that are candidates
for each candidate $c \in C_t$
c.count ++;
}
$L_k = \{ c \in C_k \mid c.count \geq min\text{-}sup \}$
}

return $L = \cup_k L_k$.

Procedure apriori_gen (L_{k-1}: frequent $(k-1)$ itemset)

---

for each itemset $l_1 \in L_{k-1}$

for each itemset $l_2 \in L_{k-1}$

if ($l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \ldots$
$l_1[k-2] = l_2[k-2] \wedge l_1[k-1] < l_2[k-1]$)

{

$c = l_1 \bowtie l_2$

if has_infrequent_subset (c, L_{k-1}) then

delete c // prune step

else add c to C_k;

}

return C_k;

Procedure has_infrequent_subset (c: candidate k-itemset)
L_{k-1}: frequent $(k-1)$ itemset); // use prev knowledge

---

for each $(k-1)$ subset s of c

if $s \notin L_{k-1}$ then

return TRUE

else

return FALSE

→ Generating Association Rules from frequent itemsets

Association rules are generated as follows.

1. For each frequent itemset l, generate all non-empty subset of l

2. For every non empty subset s of l, output rule $s \Rightarrow (l-s)$, if $\frac{\text{support\_count}(l)}{\text{support\_count}(s)} \geq$ min-conf; where min-conf is the minimum confidence threshold.

eg $\{I_1, I_2, I_3\}$, $\{I_1, I_2, I_5\}$ are two frequent itemsets.

Take $\underbrace{\{I_1, I_2, I_5\}}_{l}$

Subset are

$\{I_1, I_2\}, \{I_1, I_5\}, \{I_2, I_5\}$

$\{I_1\}, \{I_2\}, \{I_5\}$

s $I_1 \wedge I_2 \Rightarrow I_5$ confidence = 2/4 = 50

$I_1 \wedge I_5 \Rightarrow I_2$ = 2/2 = 100%.

$I_2 \wedge I_5 \Rightarrow I_1$ = 2/2 = 100%.

$I_1 \Rightarrow I_2 \wedge I_5$ = 2/6 = 33%.

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

$I_5 \Rightarrow I_1 \wedge I_2$ = 2/2 = 100%.

if minimum confidence threshold is 70%.
then only three rules are selected

$I_1 \wedge I_5 \Rightarrow I_2$
$I_2 \wedge I_5 \Rightarrow I_1$
$I_5 \Rightarrow I_1 \wedge I_2$

Take $\{I_1, I_2, I_3\}$

Subsets are $\{I_1\}, \{I_2\}, \{I_3\}, \{I_1, I_2\}$
$\{I_2, I_3\}, \{I_1, I_3\}$.

$I_1 \Rightarrow I_2 \wedge I_3$ = 2/6 = 33%
$I_2 \Rightarrow I_1 \wedge I_3$ = 2/7 = 29%
$I_3 \Rightarrow I_1 \wedge I_2$ = 2/6 = 33%.

no rules are selected

$I_1 I_2 \Rightarrow I_3$ = 2/4 = 50%
$I_1 \wedge I_3 \Rightarrow I_2$ = 2/4 = 50%
$I_2 \wedge I_3 \Rightarrow I_1$ = 2/4 = 50%.

Mining Frequent Patterns | Itemsets without Candidate Generation

The main disadvantages of Apriori algm are

1. It may need to generate a huge no of candidate sets.
2. It may need to repeatedly scan the database and check a large set of candidates by pattern matching.

FP growth [Frequent Pattern Growth] algm mines the complete set of frequent itemsets without candidate generation.

FP Growth Algorithm

→ It adopts a divide and conquer strategy.

→ First it compresses the database representing frequent itemsets 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 such database separately.

Eg:

TID	List of item-IDs
T_1	I_1, I_2, I_5
T_2	I_2, I_4
T_3	I_2, I_3
T_4	I_1, I_2, I_4
T_5	I_1, I_3
T_6	I_2, I_3
T_7	I_1, I_3
T_8	I_1, I_2, I_3, I_5
T_9	I_1, I_2, I_3

Find frequent itemsets?

Minimum Support count is 2

→ Scan the database and find set of frequent 1-itemsets and their support count

→ The set of frequent itemsets are sorted in the order of descending support count

→ The resulting set or list is denoted by L

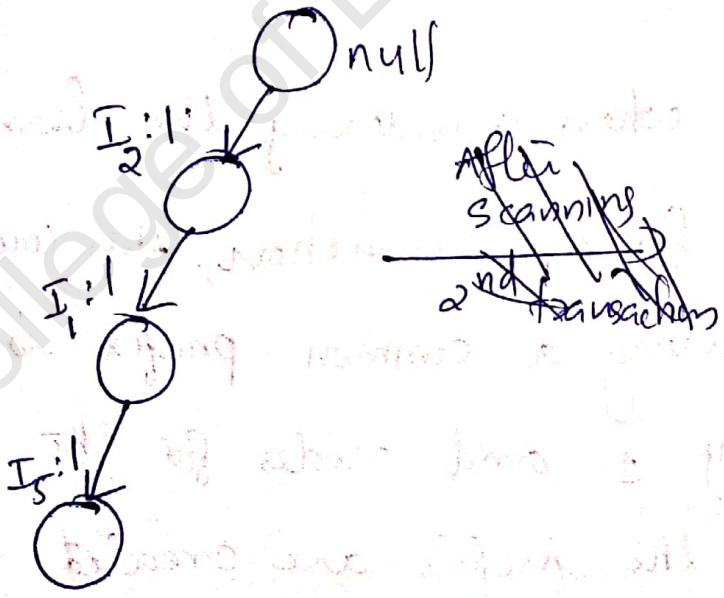
$$L = \{ \{I2:7\}, \{I1:6\}, \{I3:6\}, \{I4:2\}, \{I5:2\} \}$$

→ Then FP tree is constructed.

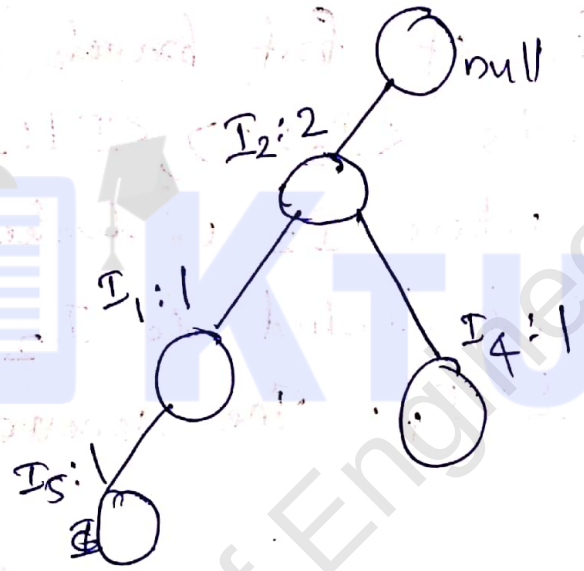
- First create the root of the tree labeled with null

- Scan the database D a second time.
- The items in each transaction are processed in L order [sorted according to descending support count] and a branch is created for each transaction.

eg: $T1 : I_1, I_2, I_5$ which contains I_2, I_1 & I_5 in L order leads to the construction of first branch of tree with nodes $\langle I_2 : 1\rangle$, $\langle I1 : 1\rangle$ and $\langle I_5 : 1\rangle$ where I_2 is linked to the root, I_1 is linked to I_2 and I_5 is linked to I_1. The second transaction

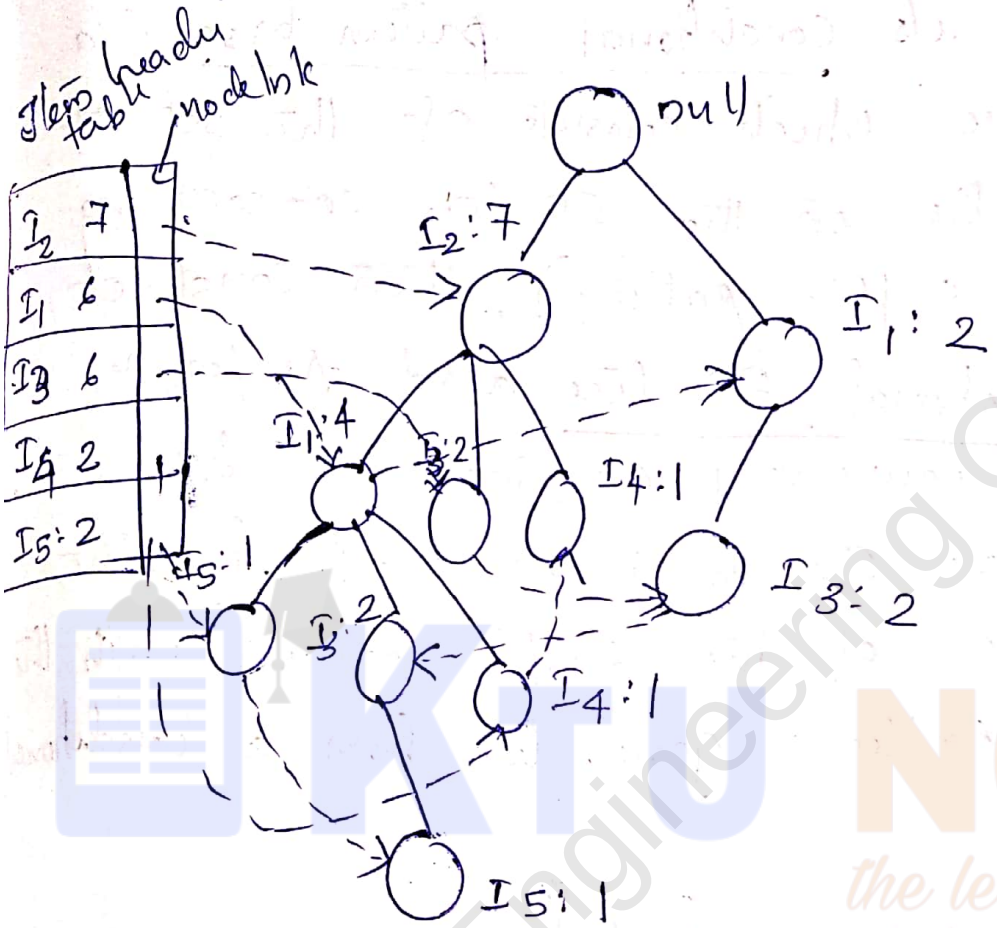


The second transaction T_2, contains I2 and I_4 in L order which would result in a branch where I_2 is linked to the root and I_4 is linked to I2. This branch share a common prefix I_2, with the path for T_1. Therefore instead increment count of I_2 node and create new node $\langle I_4 : 1 \rangle$ which is linked to I_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.



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

The FP tree is mined as follows.

Start from each frequent length-1 pattern [item with minimum support count]
Construct its conditional pattern base (a subdatabase which consists of the set of prefix paths in the FP tree co-occuring with the suffix pattern), then construct its conditional FP tree and perform mining recursively on such a tree.
The pattern Growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional tree.

Eg: From the previous FP tree.

We first consider I5. I5 occurs in two branches of FP tree.

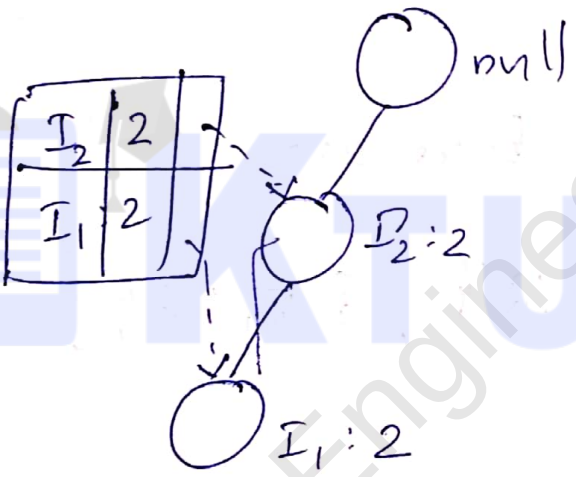
The path ~~followed~~ formed by these branches are <I2, I1, I5 : 1> and <I2 : I1 : I3 : I5 : 1>

Considering I5 as suffix, its corresponding prefix paths are <I2, I1 : 1> and <I2, I1, I3 : 1> which forms its conditional pattern base.

Then construct conditional FP tree.

	Support count
I_2	2
I_1	2
~~I_3~~	1

→ It is eliminated because support count is less than minimum support count.



Conditional FP tree >

< I2 : 2, I1 : 2 >

The single path generates all the combination of frequent patterns:

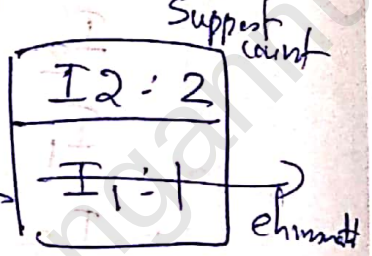
{I2, I5 : 2}, {I1, I5 : 2}, ~~{I4, I2}~~

{I2, I1, I5 : 2}

For I4,

The prefix path from the conditional pattern base

{ {I2, I1 : 1}, {I2 : 1} }



which a single node conditional FP tree <I2 : 2> and derives one frequent pattern, {I2, I4 : 2}

For I3

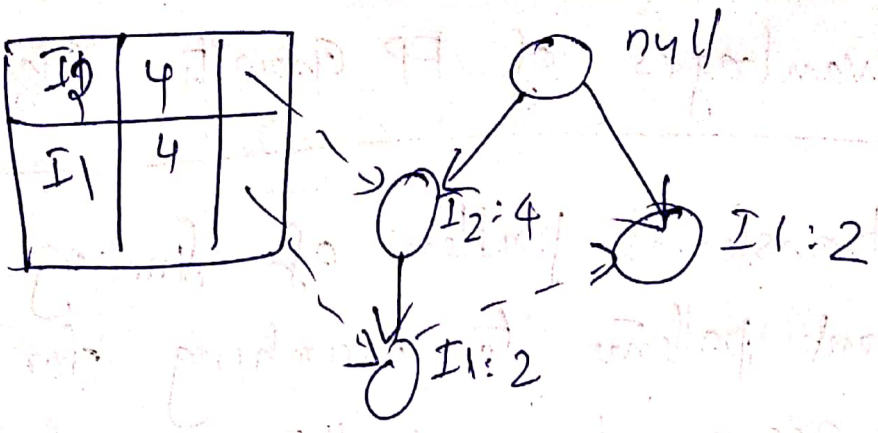
Conditional pattern bases are

{ {I2 : 2}, {I2, I1 : 2}, {I1 : 2} }

From same branch of FP tree

Here the conditional FP tree contains two branches.

<I2 : 4, I1 : 2> and <I1 : 2>



which Generates a set of patterns

$\{\{I2, I3:4\}, \{I1, I3:4\}, \{I2, I1, I3:2\}\}$

For I1,

Condition base is $\{I_2:4\}$ whose fp tree contains only one node $\langle I2:4\rangle$, which Generates frequent pattern $\{I_2, I1:4\}$

Summary

Item	Conditional Pattern base	Conditional FP tree	Frequent patterns Generated
I5	$\{\{I_2, I1:1\}, \{I_2, I1, I3:1\}\}$	$\langle I_2:2, I1:2\rangle$	$\{I_2, I5:2\}\{I1, I5:2\}$ $\{I2, I1, I5:2\}$
I4	$\{\{I2, I1:1\}, \{I_2, 1\}\}$	$\langle I2:2\rangle$	$\{I_2, I_4:2\}$
I3	$\{\{I_2, I_1:2\}, \{I_2:2\}\{I1:2\}\}$	$\langle I2:4, I1:2\rangle\langle I1:2\rangle$	$\{I_2, I_3:4\}$ $\{I1, I3:4\}$ $\{I_2, I1, I3:2\}$
I1	$\{\{I_2:4\}\}$	$\{I_2:4\}$	$\{I_2, I_1:4\}$

Advantages of FP Growth Algm

→ It transforms problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating suffix

→ It is efficient and scalable for mining both long and short frequent patterns

→ It is faster than Apriori algorithm.

→ when database is large, and if not possible to construct main memory based FP tree, first partition the database into set of projected database and construct FP tree and mine patterns in each projected database.