Scheme of Evaluation

Internal Assessment Test 2 – September 2024

Sub:	Data Mining & Business Intelligence							Sub Code:	22MCA252
Date:	14-09-24	Duration:	90 mins	Max Marks:	50	Sem:	2	Branch:	MCA

Q.NO	Description	Marks Distribution	Max Marks
1	**Explain Data cube operations with example of each operations** • List out the types of data cube operations and also explain it in brief with example.	10	10
2	**Difference between OLAP and OLTP** • Give at least minimum 7-10 points	10	10
3	**Explain the concept description in detail** • List out the procedure for concept of description • Explain it in brief	2 8	10
4	**What is Association rule mining? Explain it in detail.** • Definition of Association rule mining • Explain it with example	2 8	10
5	**Define Apriori principle. Briefly discuss about Apriori algorithm for frequent Item set generation** • Definition of Apriori Principle • Explanation of Apriori Algorithm with example	3 7	10
6	**For a given transaction data ,generate frequent itemset and identify valid association rules with minimum support as 60% and minium confidence as 75%** **Tid** / **Items** 1 – Bread,Cheese,eggs,juice 2 – Bread,Cheese, ,juice 3 – Bread,Milk,Yogurt 4 – Bread,Juice,Milk 5 – Cheese,juice,Milk • Solutions given by stepwise	10	10
7	**Write FP Growth algorithm for discovering frequent itemsets without candidate generation.** • Explanation of FP Growth algorithm	10	10

<table>
<tr>
<td>8</td>
<td>Construct FP tree for the transaction data set shown in the table and explain steps of construction using FP Growth Algorithm
<table>
<tr><th>Tid</th><th>Items</th></tr>
<tr><td>T1</td><td>{E,K,M,N,O,Y}</td></tr>
<tr><td>T2</td><td>{D,E,K,N,O,Y}</td></tr>
<tr><td>T3</td><td>{A,E,K,M}</td></tr>
<tr><td>T4</td><td>{C,K,M,U,Y}</td></tr>
<tr><td>T5</td><td>{C,E,I,K,O,O}</td></tr>
</table>
Solutions given by stepwise</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>Explain classification process. Write an algorithm for decision tree induction technique
Definition of classification process
Explanation of the algorithm for decision tree</td>
<td>2
8</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>Write a note on Naïve Baye’s Classifier.
Give short notes on Naïve Bayes Classifier with example.</td>
<td>10</td>
<td>10</td>
</tr>
</table>

Internal Assessment Test 2 – September 2024

Data Mining and Business Intelligence							Sub Code:	22MCA252
14/09/2024	Duration:	90 min's	Max Marks:	50	Sem:	I	Branch:	MCA

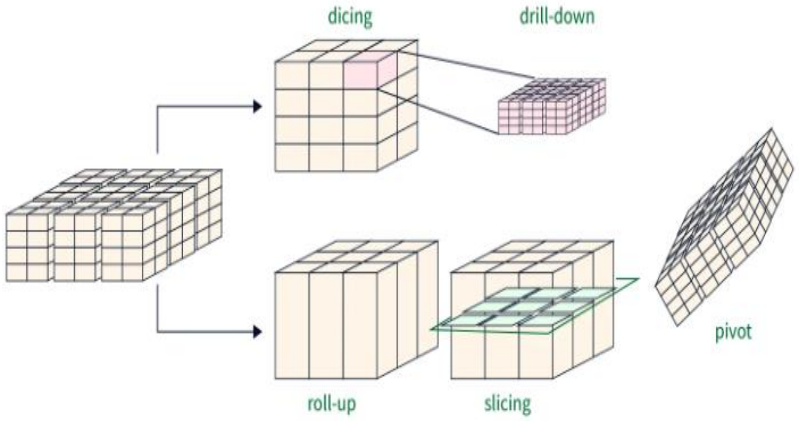
PART I

1. Explain Data cube operations with example of each operations

Operations on Data Cube

Operations on a data cube in data mining are used to analyze data from different perspectives and gain insights into data patterns and trends. The five common operations on a data cube are -

- **Roll-up** - This operation involves summarizing data along one or more dimensions of a data cube. It results in a data cube with a lower level of granularity. For example, we can roll up a sales data cube from monthly sales to quarterly sales, resulting in a data cube with fewer dimensions and a higher level of aggregation.
- **Drill-down** - This operation involves increasing the level of detail in a data cube by adding more dimensions or attributes to the existing dimensions. It results in a data cube with a higher level of granularity. For example, we can drill down a sales data cube from quarterly to monthly sales by adding the month dimension to the existing time dimension.
- **Slice** - This operation involves selecting a subset of a data cube by fixing the values of one or more dimensions. It results in a smaller data cube with the same dimensions but fewer data points. For example, we can slice a sales data cube to analyze sales data for a particular region and time period.
- **Dice** - This operation involves selecting a subset of a data cube by fixing the values of one or more dimensions and selecting a range of values for another dimension. It results in a smaller data cube with fewer dimensions and data points. For example, we can dice a sales data cube to analyze sales data for a particular region, time period, and product category.
- **Pivot** - This operation involves changing the orientation of a data cube by rotating the dimensions and measures. It results in a data cube with a different perspective on the data. For example, we can pivot a sales data cube to analyze sales data by product category and time period instead of the time period and product category.



Advantages of Data Cube

Data cube in data mining provides several advantages -

- **Multidimensional analysis** - Data cube technology in data mining enables users to analyze data from multiple perspectives and dimensions, such as time, product, location, and customer, allowing for a more comprehensive data view.
- **Fast query performance** - Data cubes pre-aggregate data at multiple levels of granularity, making it easier and faster to query large datasets and retrieve results.
- **Reduced data redundancy** - Data cubes store pre-aggregated data at various levels of granularity, reducing the need to store redundant data in a database.
- **Data visualization** - Data cube in data mining can be visualized using charts, graphs, and other graphical representations, making it easier for users to understand and analyze complex data.
- **Improved decision-making** - Data cube technology in data mining allows users to drill down, roll up, slice, and dice data, enabling them to make informed decisions based on insights gained from the data.
- **Scalability** - Data cubes can handle large datasets and be stored in a database, making them scalable for enterprise-level data mining.

Disadvantages of Data Cube

While data cube in data mining provides several advantages, they also have some disadvantages -

- **Data cube creation** - Creating a data cube in data mining can be a time-consuming and complex process that requires careful consideration of the dimensions, measures, and aggregation levels.

- **Data storage requirements** - Data cubes can require significant storage space, especially when dealing with large datasets with many dimensions and measures.
- **Limited flexibility** - Data cubes are optimized for multidimensional analysis and may need to be more flexible to accommodate changes to the underlying data or analysis requirements.
- **Data quality issues** - Data cube technology in data mining relies on the accuracy and consistency of the underlying data, which can be challenging to achieve when dealing with complex datasets.
- **Complexity** - While data cubes simplify the analysis of complex data, the analysis itself can be complex, requiring knowledge of the dimensions, measures, and aggregation levels used in the data cube.

2. Difference between OLAP and OLTP

OLAP	OLTP
Involves historical processing of information.	Involves day-to-day processing.
OLAP systems are used by knowledge workers such as executives, managers and analysts.	OLTP systems are used by clerks, DBAs, or database professionals.
Useful in analyzing the business.	Useful in running the business.
Contains historical data.	Contains current data.
Provides summarized and multidimensional consolidated data.	Provides primitive and highly detailed data.
Number or users is in hundreds.	Number of users is in thousands.
Number of records accessed is in millions.	Number of records accessed is in tens.
Database size is from 100 GB to 1 TB	Database size is from 100 MB to 1 GB.
Highly flexible.	Provides high performance.
Based on Star Schema, Snowflake, Schema and Fact Constellation Schema.	Based on Entity Relationship Model.

PART II

3. Explain the concept description in detail

The simplest kind of descriptive data mining is concept description. A concept usually refers to a collection of data such as frequent_buyers, graduate_students, and so on. As a data mining task, concept description is not a simple enumeration of the data. Instead, concept description generates descriptions for characterization and comparison of the data. It is some times called class description, when the concept to be described refers to a class of objects. Characterization provides a concise and succinct summarization of the given collection of the data, while concept or class comparison (also known as discrimination) provides discriminations comparing two or more collections of data. Since concept description involves both characterization and comparison, techniques for accomplishing each of these tasks will study. Concept description has close ties with the data generalization. Given the large amount of data stored in database, it is useful to be describe concepts in concise and succinct terms at generalized at multiple levels of abstraction facilities users in examining the general behavior of the data. Given the ABCompany database, for example, instead of examining individual customer transactions, sales managers may prefer to view the data generalized to higher levels, such as summarized to higher levels, such as summarized by customer groups according to geographic regions, frequency of purchases per group, and customer income. Such multiple dimensional, multilevel data generalization is similar to multidimensional data analysis in data warehouses. The fundamental differences between concept description in large databases and online analytical processing involve the following.

Complex data types and aggregation:

Data warehouses and OLAP tools are based on a multidimensional data model that views data in the form of a data cube , consisting of dimensions (or attributes) and measures(aggregate functions). However, the possible data types of the dimensions and measures for most commercial versions of these systems are restricted. Many current OLAP systems confine dimensions to non-numeric data, similarly, measures (such as count (), sum (), average ()) in current OLAP systems apply only to numeric data. In contrast, for concept formation, the database attributes can be of various data types, including numeric, nonnumeric, spatial, text, or image. Furthermore, the aggregation of attributes in a database may include sophisticated data types, such as the collection of nonnumeric data, the merging of spatial region, the composition of images, the integration of texts, and the grouping of object pointers. Therefore, OLAP, with its restrictions on the possible dimension and measure types, represents a simplified model for data analyses. Concept description in databases can handle complex data types of the attributes and their aggregations, as necessary.

User-control versus automation:

On-line analytical processing in data warehouses is a purely user-controlled process. the selection of dimensions and the application of OLAP operations, such as drill-down, roll-up, slicing, and dicing, are directed and controlled by the users, although the control in most OLAP systems is quite user-friendly, users do require a good understanding of the role of each dimension. Furthermore, in order to find a satisfactory description of the data, users may need to specify a long sequence of OLAP operations. In contrast, concept description in data mining strives for a more automated process that helps determine which dimensions (or attributes) should be included in the analyses, and the degree to which the giver data set should be generalized in order to produce an interesting summarization of the data. Recently, data warehousing and OLAP technology has been evolving towards handling more complex types of data and embedding more knowledge discovery mechanisms. As this technology continues to develop , it is expected that additional descriptive data mining features will be integrated into future OLAP systems. Methods for concept description, including multilevel generalization, summarization, characterization, and comparison are outlined below. Such methods set the foundation for implementation of two major functional modules in data mining: multiple-level characterization and comparison. In addition, you will also examine techniques for the presentation of concept a description in multiple forms, including tables, charts, graphs, and rules.

Data Generalization and Summarization-Based Characterization

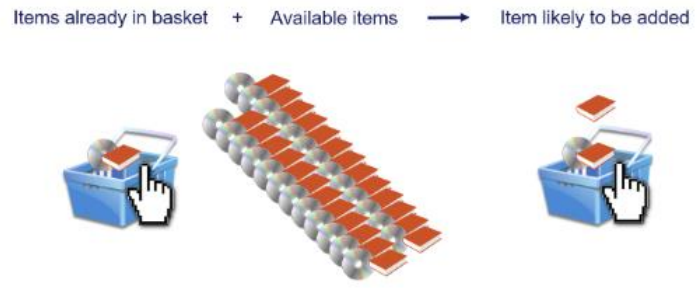
Data and objects in databases often contain detailed information at primitive concept levels. .For example, the item relation in sales database may contain attributes describing low-level item information such s item _ID , name , brand, category, supplier, place_made, and price. It is useful to be able to summarize a large set or data and present it at a high conceptual level.. For example, summarizing a large set of items relating to Christmas season sales provides a general description of such data , which can be very helpful for sales and marketing managers. This requires an important functionality in data mining: data generalization. Data generalization is a process that abstracts a large set of task-relevant data in a database from a relatively low conceptual level to higher conceptual levels. Methods for the efficient and flexible generalization of large data sets can be categorized according to two approaches :(1) the data cube (or OLAP) approach and (2) the attribute –oriented induction approach .In this section, we describe the attribute-oriented induction approach.

Attribute-Oriented Induction

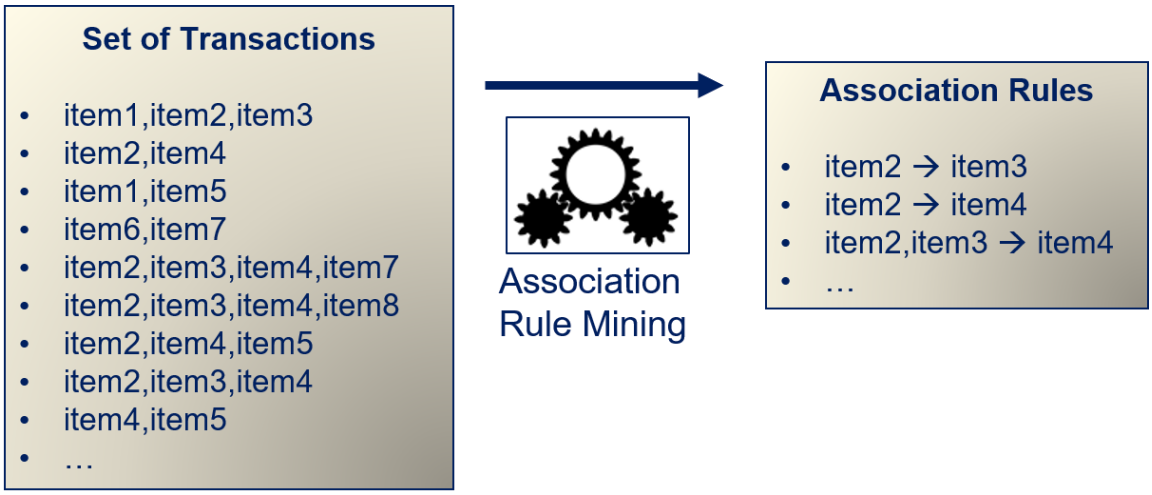
The attribute-oriented induction (AOI)) approach to data generalization and summarization-based characterization was first proposed in 1989,a few years prior to the introduction of the data cube approach. The data cube approach can be considered as a data warehouse-based, pre-computationoriented, materialized-view approach. It performs off-line aggregation before an OLAP or data mining query is submitted for processing. On the other hand, the attribute-oriented induction approach, at least in its initial proposal, is a relational database query –oriented, generalization –based, on-line data analysis technique. However, there is no inherent barrier distinguishing the two approaches based on on-line aggregation versus off-line pre computation. Some aggregations in the data cube can be computed on-line, while off-line while off-line pre -computation of multidimensional space can speed up attribute –oriented induction as well.

4. What is Association rule mining? Explain it in detail.

Associations are relationships between objects. The idea behind association rule mining is to determine rules, that allow us to identify which objects may be related to a set of objects we already know. In the association rule mining terminology, we refer to the objects as items. A common example for association rule mining is basket analysis. A shopper puts items from a store into a basket. Once there are some items in the basket, it is possible to recommend associated items that are available in the store to the shopper.



n this example, the association between items is defined as "shoppers bought items together". More generally speaking, we have *transactions*, and in each transaction we observe a set of related objects. We apply association rule mining to a set of transactions to infer *association rules* that describe the associations between items.



The relationship that the rules describe should be "interesting". The meaning of interesting is defined by the use case. In the example above, interesting is defined as "shoppers bought items together". If the association rules should, e.g., find groups of collaborators, interesting would be defined as "worked together in the past".

> We can also formally define association rule mining. We have a finite set of items $I = \{i_1, \ldots, i_m\}$ and transactions that are a subset of the items, i.e., transactions $T = \{t_1, \ldots, t_n\}$ with $t_j \subseteq I, j = 1, \ldots, n$. Association rules are logical rules of the form $X \Rightarrow Y$, where X and Y are disjoint subsets of items, i.e., $X, Y \subseteq I$ and $X \cap Y = \emptyset$. We refer to X as the *antecedent* or left-hand-side of the rule and to Y as the *consequent* or right-hand-side of the rule.

The goal of association rule mining is to identify good rules based on a set of transactions. A generic way to define "interesting relationships" is that items occur often together in transactions. Consider the following example with ten transactions.

```
[['item1', 'item2', 'item3'],

 ['item2', 'item4'],

 ['item1', 'item5'],

 ['item6', 'item7'],

 ['item2', 'item3', 'item4', 'item7'],

 ['item2', 'item3', 'item4', 'item8'],

 ['item2', 'item4', 'item5'],

 ['item2', 'item3', 'item4'],

 ['item4', 'item5'],

 ['item6', 'item7']]
```

We can see that the items item2, item3, and item4 occur often together. Thus, there seems to be an interesting relationship between the items. The question is, how can we find such interesting combinations of items automatically and how can we create good rules from interesting combinations of items.

Key Components of Association Rule Mining

1. **Itemset**: A collection of one or more items (e.g., products).
    - Example: {Bread, Butter}, {Milk, Eggs, Bread}
2. **Support**: This indicates how frequently an itemset appears in the dataset.

    Support (X)=Number of transactions containing XTotal number of transactions\text{Support (X)} = \frac{\text{Number of transactions containing X}}{\text{Total number of transactions}}Support (X)=Total number of transactionsNumber of transactions containing X

    Example: If {Milk, Bread} appears in 20 transactions out of 100 total transactions, the support is 20%.

3. **Confidence**: This measures how often items in Y appear in transactions that contain X.

Confidence (X→Y)=Support (X ∪ Y)Support (X)\text{Confidence (X→Y)} = \frac{\text{Support (X ∪ Y)}}{\text{Support (X)}}Confidence (X→Y)=Support (X)Support (X ∪ Y)

Example: If 10 transactions contain both Milk and Butter, and Milk appears in 20 transactions, the confidence of the rule Milk → Butter is 50%.

4. **Lift**: This measures the strength of the rule, indicating whether the occurrence of X and Y together is independent of each other.

   Lift (X→Y)=Support (X ∪ Y)Support (X)×Support (Y)\text{Lift (X→Y)} = \frac{\text{Support (X ∪ Y)}}{\text{Support (X)} \times \text{Support (Y)}}Lift (X→Y)=Support (X)×Support (Y)Support (X ∪ Y)

   A lift greater than 1 implies that X and Y occur together more frequently than would be expected if they were independent.

5. **Leverage**: This measures the difference between the observed frequency of X and Y appearing together and the expected frequency if X and Y were independent.

   Leverage (X→Y)=Support (X ∪ Y)−(Support (X)×Support (Y))\text{Leverage (X→Y)} = \text{Support (X ∪ Y)} - (\text{Support (X)} \times \text{Support (Y)})Leverage (X→Y)=Support (X ∪ Y)−(Support (X)×Support (Y))

Example of Association Rule

In a supermarket, suppose we have the following transaction data:

Transaction ID	Items Bought
T1	Bread, Butter, Milk
T2	Bread, Butter
T3	Milk, Eggs
T4	Bread, Butter, Milk, Eggs
T5	Butter, Eggs

Association Rule: {Bread} → {Butter}

- **Support**: The support of {Bread, Butter} is 3/5 = 60% (3 transactions contain both).
- **Confidence**: The confidence of {Bread} → {Butter} is 3/3 = 100% (All transactions containing bread also contain butter).
- **Lift**: If the probability of buying butter independently is 80%, the lift of {Bread} → {Butter} is 1.00.8=1.25\frac{1.0}{0.8} = 1.250.81.0=1.25.

Apriori Algorithm

The **Apriori algorithm** is a common method for association rule mining. It works by:

1. **Generating frequent itemsets** using support.
2. **Generating association rules** from those frequent itemsets.
3. Pruning the itemsets that do not meet the minimum support and confidence thresholds.

Applications of Association Rule Mining

1. **Market Basket Analysis**: Discovering which items are frequently purchased together.
2. **Recommendation Systems**: Recommending products based on customer behavior.
3. **Fraud Detection**: Identifying unusual patterns that may indicate fraudulent activity.
4. **Healthcare**: Finding correlations between symptoms and diseases, or treatments and outcomes.

PART III

5. Define Apriori principle. Briefly discuss about Apriori algorithm for frequent Item set generation

The **Apriori principle** is a foundational concept in association rule mining that states:

If an itemset is frequent, then all its subsets must also be frequent.

Conversely:

If an itemset is infrequent, then all its supersets will also be infrequent.

This principle allows for efficient pruning of candidate itemsets when searching for frequent itemsets in large datasets. By eliminating infrequent itemsets early, the Apriori algorithm reduces the computational complexity of finding frequent itemsets.

Apriori Algorithm for Frequent Itemset Generation

The **Apriori algorithm** is a widely used technique for finding frequent itemsets in large databases. It leverages the Apriori principle to generate and prune candidate itemsets efficiently. The algorithm works iteratively, increasing the size of itemsets (from 1-itemset, 2-itemset, etc.) until no more frequent itemsets can be found.

Steps in Apriori Algorithm:

1. **Set Minimum Support Threshold**:

- Before starting, a minimum support threshold is set to filter out infrequent itemsets.

2. **Generate 1-Itemsets (C1)**:
   - The algorithm first scans the dataset and counts the frequency (support) of each item. This generates a list of 1-itemsets.
3. **Prune Infrequent 1-Itemsets**:
   - Itemsets whose support is below the minimum support threshold are discarded. The remaining itemsets are called frequent 1-itemsets (L1).
4. **Generate Candidate 2-Itemsets (C2)**:
   - From the frequent 1-itemsets (L1), pairs of items (2-itemsets) are generated. These are called candidate 2-itemsets.
5. **Prune Infrequent 2-Itemsets**:
   - Again, the support of each 2-itemset is calculated, and those with support below the minimum threshold are removed, leaving frequent 2-itemsets (L2).
6. **Repeat Process for Larger Itemsets**:
   - The algorithm continues iteratively, generating candidate 3-itemsets (C3) from frequent 2-itemsets (L2), then candidate 4-itemsets (C4), and so on.
   - At each step, the infrequent itemsets are pruned based on the support threshold.
7. **Terminate**:
   - The algorithm stops when no further frequent itemsets can be generated (i.e., no more candidates pass the support threshold).
8. **Generate Association Rules**:
   - After generating all frequent itemsets, the algorithm can derive **association rules** (like $X \rightarrow Y$) from these itemsets, which must satisfy the minimum **confidence** threshold.

Example of Apriori Algorithm:

Consider the following transactions:

Transaction ID	Items Bought
T1	Bread, Butter, Milk
T2	Bread, Butter
T3	Milk, Eggs
T4	Bread, Butter, Eggs
T5	Butter, Eggs

Step 1: Set minimum support = 2 transactions (40%)

Step 2: Generate 1-itemsets and prune:

Item	Support Count	Pruned?
Bread	3	No
Butter	4	No

Item	Support Count	Pruned?
Milk	2	No
Eggs	3	No

All 1-itemsets are frequent.

Step 3: Generate 2-itemsets from frequent 1-itemsets:

2-Itemset	Support Count	Pruned?
{Bread, Butter}	3	No
{Bread, Milk}	1	Yes
{Bread, Eggs}	1	Yes
{Butter, Milk}	1	Yes
{Butter, Eggs}	2	No
{Milk, Eggs}	1	Yes

After pruning, only {Bread, Butter} and {Butter, Eggs} remain as frequent 2-itemsets.

Step 4: Generate 3-itemsets:

3-Itemset	Support Count	Pruned?
{Bread, Butter, Eggs}	1	Yes

There are no frequent 3-itemsets, so the algorithm terminates.

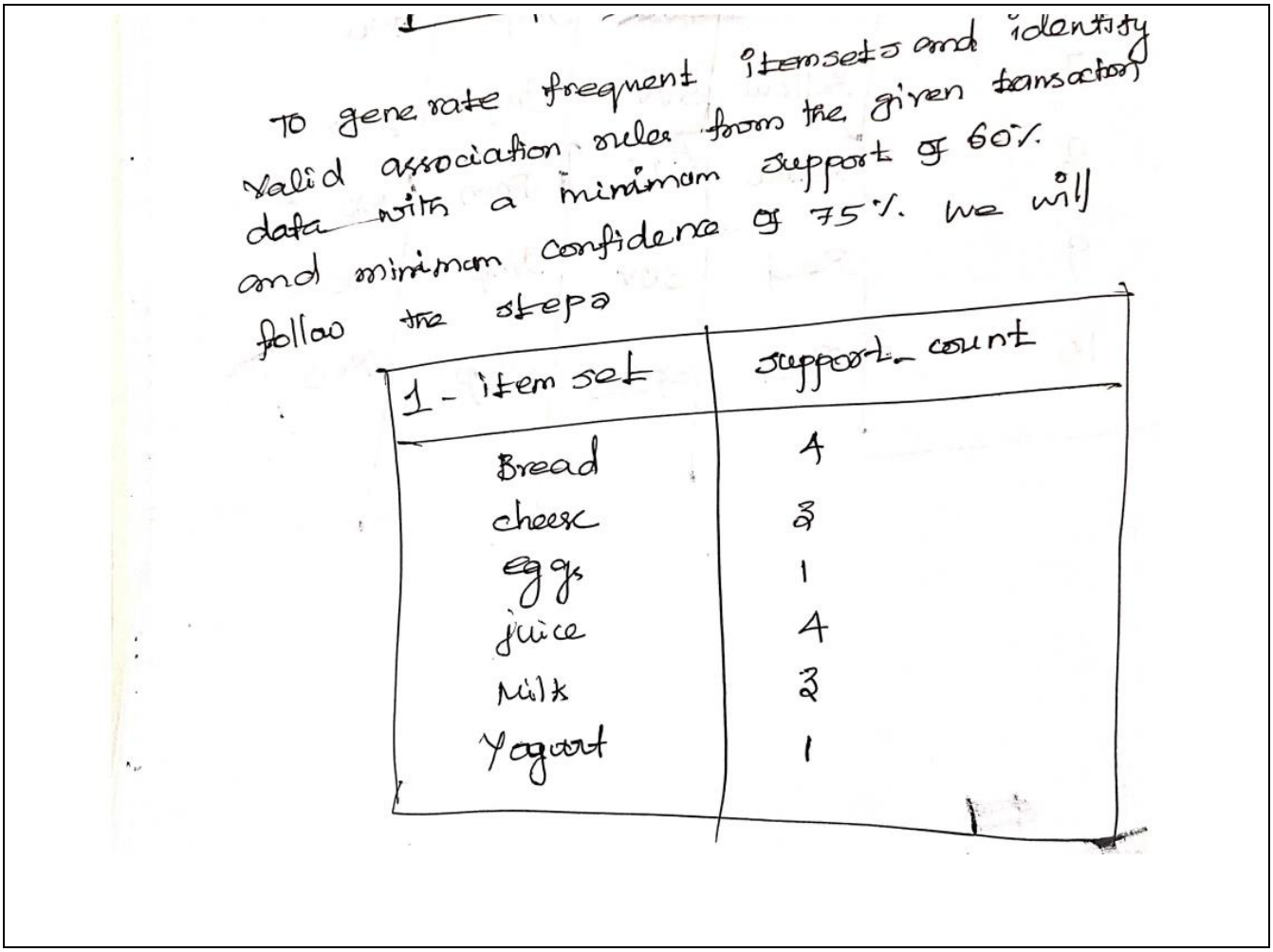
Efficiency of the Apriori Algorithm

The Apriori algorithm is efficient due to its ability to **prune the search space**. Instead of generating all possible itemsets, it focuses only on the frequent itemsets, significantly reducing the number of calculations.

However, the algorithm may still have limitations with very large datasets because of multiple passes over the data, which can be time-consuming.

6. For a given transaction data ,generate frequent itemset and identify valid association rules with minimum support as 60% and minium confidence as 75%

Tid	Items
1	Bread,Cheese,eggs,juice
2	Bread,Cheese, ,juice
3	Bread,Milk,Yogurt
4	Bread,Juice,Milk
5	Cheese,juice,Milk



To generate frequent itemsets and identify valid association rules from the given transaction data with a minimum support of 60% and minimum confidence of 75%. we will follow the steps

1 - item set	support count
Bread	4
cheese	3
eggs	1
juice	4
Milk	3
Yogurt	1

min_support = 60%.

min_support_count = min_support × itemset count

= 60% × 6

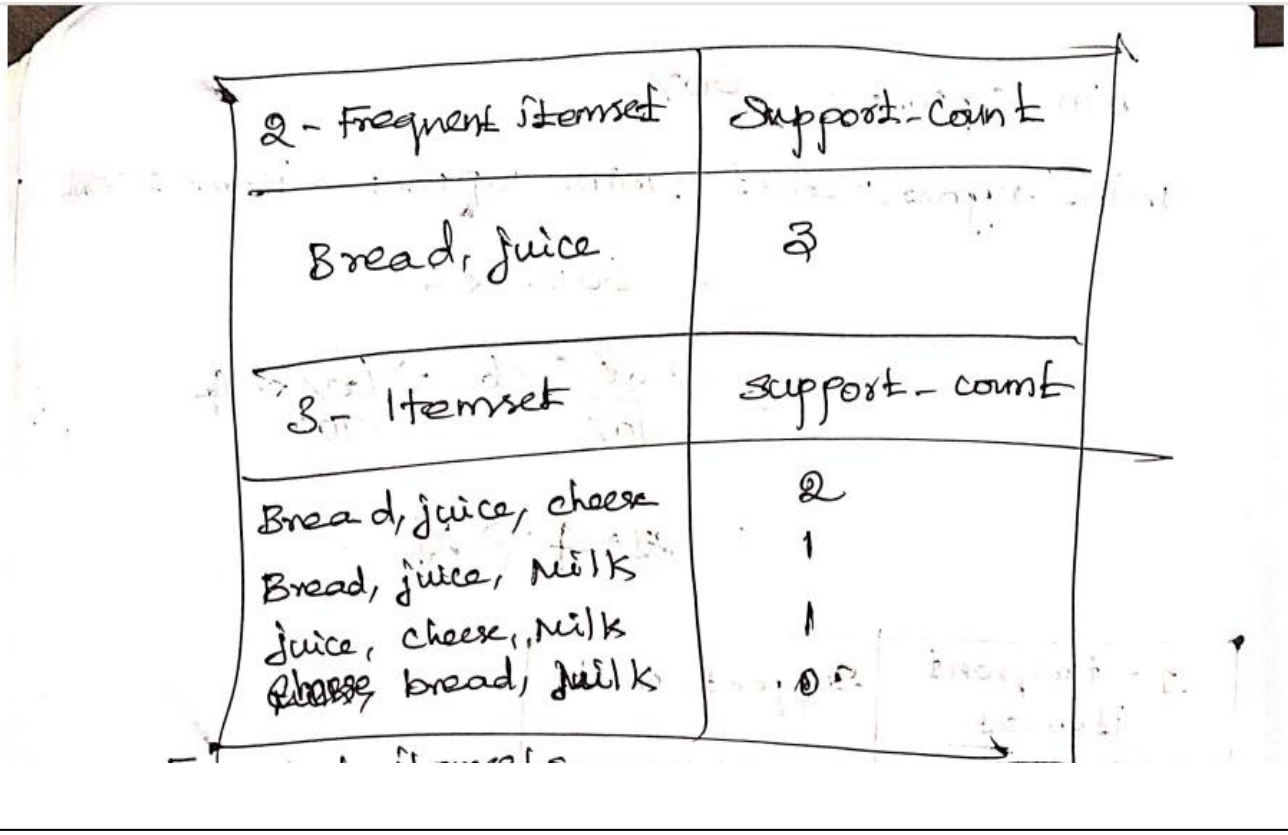
$\Rightarrow \frac{60}{100} \times 6 \Rightarrow \frac{60}{100} \times 4$

$= 2.4 \Rightarrow 2$

1 - Frequent itemset	Support_count
Bread	4
juice	4
cheese	3
Milk	3

2 - item set	Support count
Bread, juice	3
Bread, Milk	2
Bread, cheese	2
juice, cheese	2
juice, Milk	2
cheese, Milk	2

Bread, juice, cheese, Milk.



2 - Frequent Itemset	Support-Count
Bread, juice	3

3 - Itemset	support - count
Bread, juice, cheese	2
Bread, juice, Milk	1
juice, cheese, Milk	1
cheese, bread, Milk	0

min. confidence = 75%.

confidence $(X \rightarrow Y) = P(Y|X) = P(X \cup Y) / P(X)$

{Bread, juice}

Therefore candidate rules are:

For {Bread, juice}

$\Rightarrow$ Bread > juice $= \frac{P(\text{Bread, juice})}{P(\text{Bread})}$

$= \frac{3}{3} = 100\%$ (strong)

$\Rightarrow$ juice > Bread $= \frac{P(\text{juice, Bread})}{P(\text{juice})}$

$= \frac{3}{4}$

$= 75\%$ (strong)

Valid Association Rule (with minimum confidence of 75%):

{Bread} $\Rightarrow$ {juice} with 100% confidence.

This rule means that everytime "Bread" appears in a transaction, "juice" also appears with 100% confidence based on the data provided.

PART IV

7. Write FP Growth algorithm for discovering frequent itemsets without candidate generation.

FP-Growth Algorithm

The **FP-Growth (Frequent Pattern Growth)** algorithm is an efficient method for discovering frequent itemsets in a dataset without generating candidate itemsets. It overcomes the limitations of the Apriori algorithm, which involves the generation of a large number of candidate itemsets. Instead of candidate generation, FP-Growth uses a compact data structure called the **FP-Tree (Frequent Pattern Tree)** to store the dataset and extract frequent itemsets directly.

Steps of the FP-Growth Algorithm

1. **Build the FP-Tree**:
   - **Step 1.1: Scan the dataset**: The algorithm first scans the dataset to compute the frequency (support) of each individual item.
   - **Step 1.2: Sort items in descending order of support**: Items are sorted by their frequency in each transaction. This ensures that the most frequent items appear closer to the root of the tree.
   - **Step 1.3: Build the FP-Tree**: The algorithm builds a compressed representation of the dataset called an FP-Tree by adding transactions one by one. Transactions with shared items share common prefixes in the tree.
2. **Mining the FP-Tree**:
   - **Step 2.1: Conditional Pattern Base**: For each item in the FP-Tree, the algorithm constructs its conditional pattern base, which is the set of prefix paths ending with the item.
   - **Step 2.2: Conditional FP-Tree**: From the conditional pattern base, a new FP-Tree is created, called a conditional FP-Tree, which is recursively mined to extract frequent itemsets.
3. **Extract Frequent Itemsets**:
   - The frequent itemsets are extracted by traversing the FP-Tree and combining the items in the prefix paths with the suffix item (the item whose conditional pattern base is being analyzed).

Detailed Steps of FP-Growth Algorithm

1. Build the FP-Tree

Given a dataset of transactions:

Transaction ID	Items Bought
T1	{Bread, Milk, Butter}
T2	{Bread, Butter}

Transaction ID	Items Bought
T3	{Milk, Eggs}
T4	{Bread, Butter, Eggs}
T5	{Butter, Eggs}

Step 1: Scan the dataset to find item frequencies:

Item	Support
Bread	3
Milk	2
Butter	4
Eggs	3

Step 2: Sort items by support:

- For each transaction, sort the items in descending order based on their frequency.

Transaction ID	Sorted Items
T1	Butter, Bread, Milk
T2	Butter, Bread
T3	Eggs, Milk
T4	Butter, Bread, Eggs
T5	Butter, Eggs

Step 3: Build the FP-Tree:

- Start with an empty root node. For each transaction, insert items into the tree.

```
scss
Copy code
Root
 ├─ Butter (4)
 │   ├─ Bread (3)
 │   │   ├─ Milk (1)
 │   │   └─ Eggs (1)
 │   └─ Eggs (1)
 └─ Eggs (2)
     └─ Milk (1)
```

This FP-Tree represents the compressed set of transactions.

2. Mining the FP-Tree

Once the FP-Tree is built, the algorithm mines it to discover frequent itemsets. This is done recursively by:

- **Step 2.1: Extracting conditional pattern bases** for each item.
- **Step 2.2: Creating conditional FP-Trees** for those bases.

For example, to find frequent itemsets involving **Eggs**, the conditional pattern base for Eggs would be:

- From path Butter → Bread → Eggs (support = 1)
- From path Butter → Eggs (support = 1)
- From path Eggs → Milk (support = 1)

The frequent itemsets involving Eggs can now be mined recursively.

3. Extract Frequent Itemsets

The final frequent itemsets are obtained by combining the items from the prefix paths with the item whose conditional pattern base is being mined. Some of the frequent itemsets extracted could be:

- {Eggs}
- {Butter, Eggs}
- {Butter, Bread, Eggs}
- {Milk, Eggs}
- {Bread, Butter}

FP-Growth Algorithm in Pseudocode

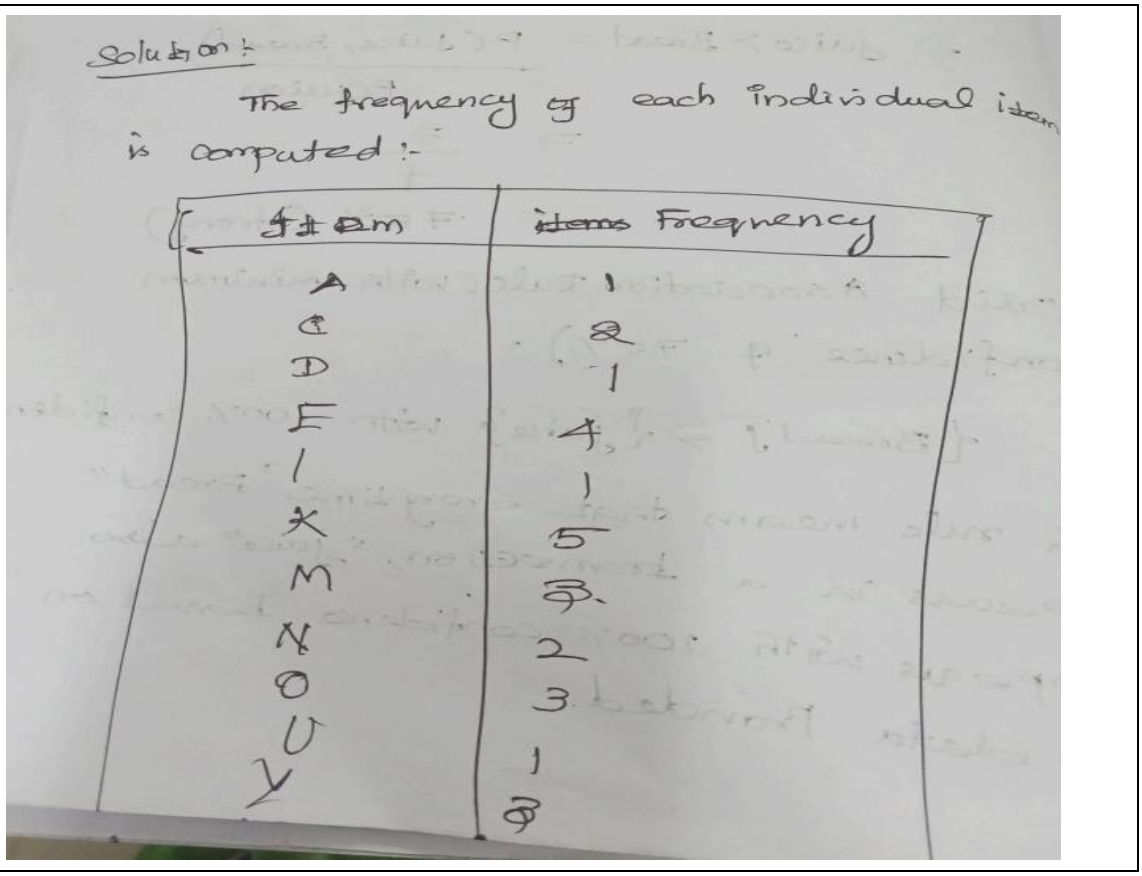
```
plaintext
Copy code
Algorithm: FP-Growth
Input: A transactional database, and a minimum support threshold.
Output: The complete set of frequent itemsets.

1. Build the FP-Tree:
    a. Scan the dataset to find the support of each item.
    b. Remove infrequent items (those that do not meet the minimum support).
    c. Sort frequent items in descending order of support.
    d. For each transaction, insert items into the FP-Tree, sharing common
prefixes.

2. Mine the FP-Tree recursively:
    a. For each item in the FP-Tree, construct its conditional pattern base.
    b. Build the conditional FP-Tree from the conditional pattern base.
    c. Recursively mine the conditional FP-Tree.
    d. Combine frequent itemsets from the conditional FP-Tree with the suffix
item.

3. Return all frequent itemsets.
```

Advantages of FP-Growth

1. **No Candidate Generation**: Unlike the Apriori algorithm, FP-Growth does not generate candidate itemsets, which reduces computational overhead.
2. **Compact Data Representation**: The FP-Tree efficiently compresses the dataset, allowing frequent patterns to be mined directly from the tree.
3. **Faster Execution**: FP-Growth is generally faster than Apriori, especially for large datasets, because it avoids multiple scans over the database.

8. Construct FP tree for the transaction data set shown in the table and explain steps of construction using FP Growth Algorithm

Tid	Items
T1	{E,K,M,N,O,Y}
T2	{D,E,K,N,O,Y}
T3	{A,E,K,M}
T4	{C,K,M,U,Y}
T5	{C,E,I,K,O,O}



* Frequent Pattern set (L) is built which will contain all the elements whose frequency is greater than or equal to the minimum support.

As minimum support be 3.

These elements are stored in descending order of their respective frequencies

After insertion of the relevant items, the set L looks like this:-

$$L = \{K:5, E:4, M:3, O:3, Y:3\}$$

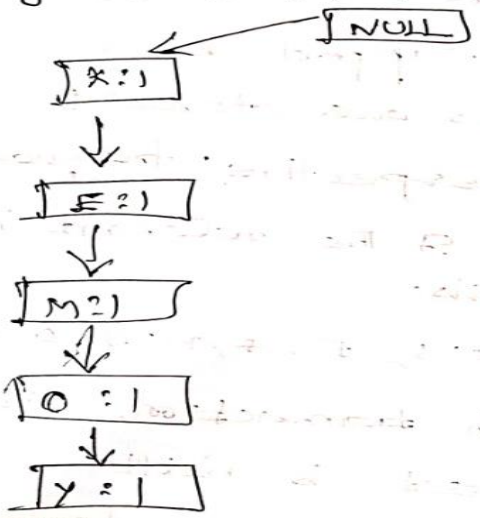
Now, for each transaction, the respective Ordered_Item set is built.

Frequent Pattern set $L = \{K:5, E:4, M:3, O:3, Y:3\}$

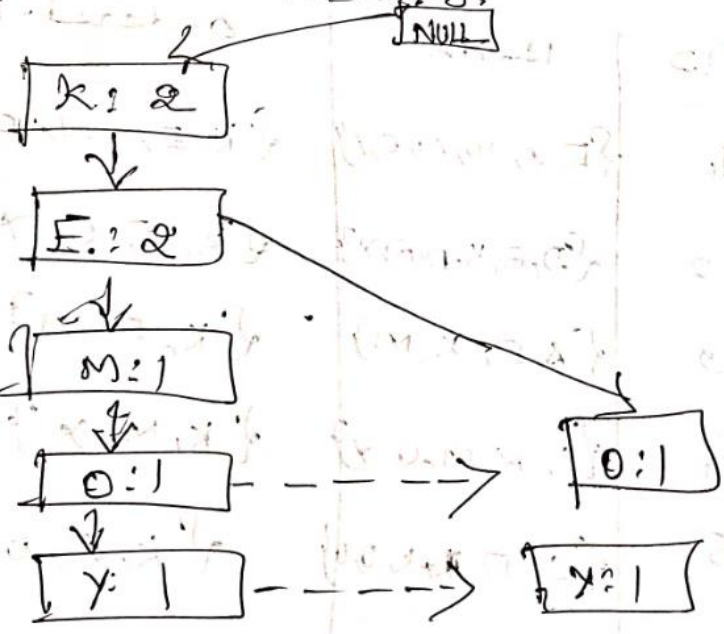
TID	Items	Ordered-Item set
T_1	{E, K, M, N, O, Y}	{K, E, M, O, Y}
T_2	{D, E, K, N, O, Y}	{K, E, O, Y}
T_3	{A, E, K, M}	{K, E, M}
T_4	{C, K, M, U, Y}	{K, M, Y}
T_5	{C, E, I, K, O}	{K, E, O}

Now, all the ordered item sets are inserted in to a Trie data structure.

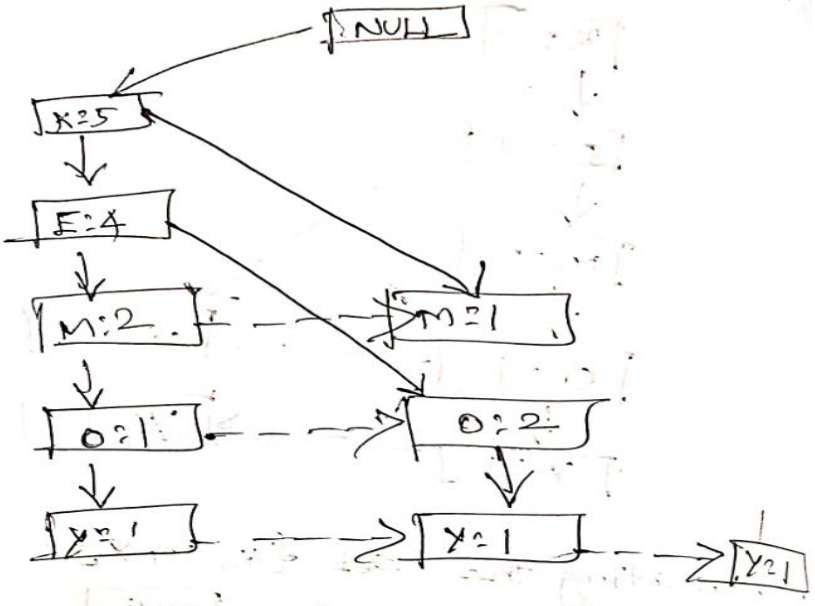
a) Inserting the set {K, E, M, O, Y}:



b) Inserting the set {K, E, O, Y}:



e) Inserting the set {K, E, O}:



Now for each item, the conditional Pattern Base is computed which is Path labels of all the paths which lead to any node of the given item in the frequent - Pattern tree -

Items	conditional Pattern Base
Y	{{K, E, M, O:1}, {K, E, O:1}, {K, M:1}}
O	{{K, E, M:1}, {K, E:2}}
M	{{K, E:2}, {K:1}}
E	{K:4}
K	—

Now for each item the conditional frequent Pattern Tree is built. It is done by taking the set of elements which is common in all the paths in the conditional Pattern Base of that item and calculating it's support count by summing the support counts of all the paths in the conditional Pattern Base.

Items	Conditional Pattern Base	Conditional Frequent Pattern Tree
Y	{K, E, M, O : 1} {K, E, O : 1} {K, M : 1}	{K : 3}
O	{K, E, M : 1} {K, E : 2}	{K, E : 3}
M	{K, E : 2}, {K : 1}	{K : 3}
E	{K : 4}	{K : 4}
K	—	—

From the conditional Frequent Pattern tree, the Frequent Pattern rules are generated by pairing the items of the conditional Frequent Pattern Tree set to the corresponding to the item as given in the below table:

Items	Frequent Pattern Generated
Y	{⟨K, Y : 3⟩}
O	{⟨K, O : 3⟩, ⟨E, O : 3⟩, {E, K, O : 3}}
M	{⟨K, M : 3⟩}
E	{⟨E, K : 3⟩}
K	—

PART V

9. Write an algorithm for decision tree induction technique

Algorithm for Decision Tree Induction

Input:

- Training dataset `D` with `n` attributes and `m` instances.
- Target attribute (class label).

Output:

- A decision tree.

Steps:

1. **Start**
   - If all instances in `D` belong to the same class, return a single-node tree with that class label.
2. **Attribute Selection**
   - For each attribute `A`, compute the **information gain** (or **Gini index**, **Gain ratio**, etc., depending on the criterion) to determine how well `A` classifies the instances.
   - Select the attribute `A` that has the highest information gain as the **splitting attribute**.
3. **Create Node**
   - Create a decision node in the tree corresponding to the selected attribute `A`.
4. **Partition the Dataset**
   - Partition the dataset `D` into subsets `D1, D2, ..., Dk`, based on the values of the selected attribute `A`.
   - For each subset `Di` corresponding to a value `vi` of attribute `A`:
     - If `Di` is empty, create a leaf node with the most common class label from the parent set.
     - If `Di` contains instances with more than one class, repeat the process recursively using `Di` as the new dataset.
5. **Stop Condition**
   - The recursion stops when:
     - All instances in a subset belong to the same class.
     - There are no more attributes to split on.
     - The dataset is empty.
6. **Return Tree**
   - Once all subsets are processed, return the complete decision tree.

Algorithm: DecisionTreeInduction(D, attributes)

- Input:
- D - Dataset
- attributes - List of attributes
-
- Output:
- Decision tree
-
- if all instances in D have the same class label then
- return a leaf node with that class label
- else if attributes is empty then
- return a leaf node with the most common class label in D
- else
- A = Attribute with highest information gain from attributes
- Create a decision node with A as the splitting attribute
- for each value vi of attribute A do
- Di = Subset of D where A = vi
- if Di is empty then
- Add a leaf node with the most common class label in D
- else
- Add the subtree DecisionTreeInduction(Di, attributes - A) to the current node
- return the decision node

10. Write a note on Naïve Baye's Classifier.

Naïve Bayes Classifier

The **Naïve Bayes Classifier** is a probabilistic machine learning model based on **Bayes' Theorem**. It is widely used for classification tasks due to its simplicity, efficiency, and effectiveness, especially with large datasets. Despite its simplicity, it performs surprisingly well for various real-world applications like spam filtering, text classification, sentiment analysis, and medical diagnosis.

Key Assumption: Conditional Independence

The "naïve" part of Naïve Bayes comes from the assumption that all the features (or attributes) are **independent** of each other, given the class label. This means that the presence or absence of a particular feature in a class is assumed to be independent of the presence or absence of any other feature. While this assumption rarely holds in real-world situations, Naïve Bayes still performs well in many cases, even when the assumption is violated.

Bayes' Theorem

Naïve Bayes is based on **Bayes' Theorem**, which relates the probability of a class given a set of features to the probability of the features given the class. Bayes' Theorem is stated as:

P(C|X)=P(X|C)·P(C)P(X)P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}P(C|X)=P(X)P(X|C)·P(C)

Where:

- P(C|X)P(C|X)P(C|X) is the **posterior probability**: the probability of class C given the feature vector X.
- P(X|C)P(X|C)P(X|C) is the **likelihood**: the probability of the feature vector X given the class C.
- P(C)P(C)P(C) is the **prior probability**: the overall probability of class C.
- P(X)P(X)P(X) is the **evidence**: the overall probability of the feature vector X.

Working of Naïve Bayes Classifier

1. **Training Phase**:
    - Compute the prior probability for each class.
    - For each feature, compute the conditional probability of the feature value given the class label.
2. **Prediction Phase**:
    - For a new data point, calculate the posterior probability for each class using Bayes' Theorem.
    - Assign the class label with the highest posterior probability.

Advantages

- **Efficient**: It is computationally efficient and works well with large datasets.
- **Simple to Implement**: Easy to understand and implement, even with basic mathematical knowledge.
- **Works Well with High-Dimensional Data**: It performs well in text classification tasks where the data has many features.

Limitations

- **Strong Assumption of Independence**: The assumption that all features are independent is often unrealistic in real-world data.
- **Zero Probability Problem**: If a particular feature value is missing in the training dataset for a class, the model will assign a zero probability to it. This is often mitigated by using techniques like Laplace smoothing.

Applications

- **Spam Filtering**: Classifies emails as spam or non-spam based on the frequency of words.
- **Text Classification**: Categorizes documents into predefined categories, such as news articles.

- **Sentiment Analysis**: Determines the sentiment (positive/negative) of text, such as customer reviews.