

# 6)a)

**Association Rule Mining** is a way to **find interesting relationships between items** in a large set of data.

It answers questions like:

* *"If a person buys bread, are they also likely to buy butter?"*

# Association Rule Mining as a Two-Step Process:

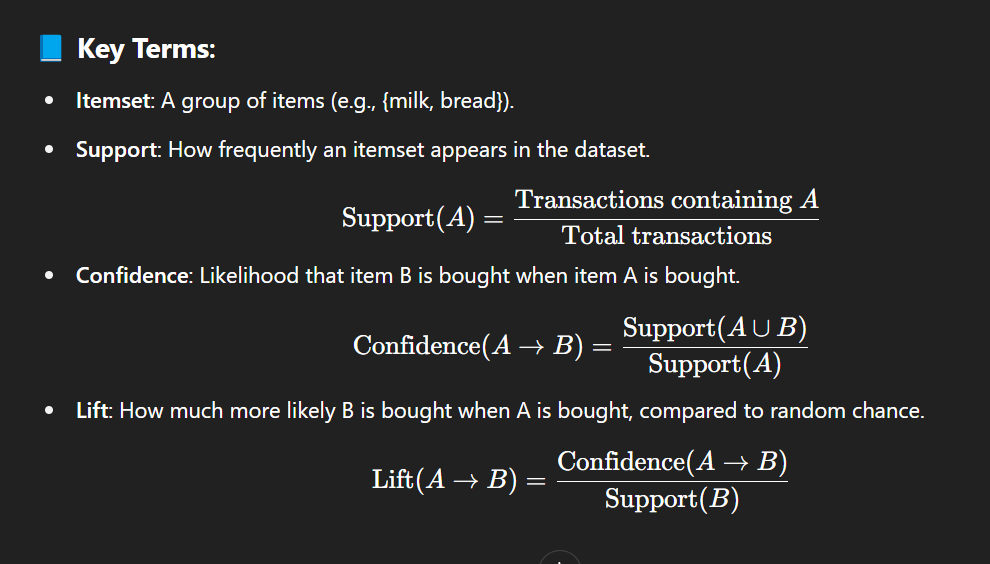
Association rule mining can be thought of in **two main steps**:

# Frequent Itemset Generation:

* + Find all item combinations (itemsets) that appear together **frequently** in the data (i.e., they satisfy a minimum support threshold).

# Rule Generation:

* + From these frequent itemsets, generate rules that predict the occurrence of an item based on the presence of other items.
  + Only rules that satisfy a minimum confidence threshold are kept.



# Example:

Suppose we have the following transactions in a store:

|  |  |
| --- | --- |
| **Transaction ID** | **Items Bought** |
| 1 | Milk, Bread, Butter |
| 2 | Milk, Bread |
| 3 | Milk, Bread, Eggs |
| 4 | Bread, Butter |
| 5 | Milk, Eggs |

# Step 1 (Frequent Itemset Generation):

Find itemsets like {Milk, Bread} that occur often (e.g., appear in at least 3 out of 5 transactions).

# Step 2 (Rule Generation):

From the frequent itemset {Milk, Bread}, generate rules like:

* + Milk → Bread (if someone buys Milk, they likely buy Bread)
  + Bread → Milk

Check if these rules meet the confidence threshold (e.g., at least 70%).

# b)

**Problem with Apriori algorithm:**

* + Apriori **generates too many candidate itemsets** (like 1-item, 2-item, 3-item sets...)
  + It **scans the whole database many times** (once for each size of itemsets).
  + This makes Apriori **slow and expensive** for large data.

# How FP-Tree solves it:

– − **FP-Tree (Frequent Pattern Tree)** builds a **tree** structure from the data **once** — no need to generate so many candidates again and again!

Here’s how FP-Tree works:

1. **First scan:** Count how often each item appears.
2. **Second scan:** Build a tree:
   * Items are **sorted by frequency** (most common first).
   * Common parts of transactions are **shared** in the tree (like branches).

# Mining:

* + Instead of generating candidates one by one, FP-Tree **directly extracts patterns** from the tree.

# Simple example:

Suppose you have transactions like:

* (milk, bread, butter)
* (milk, bread)
* (milk, butter)
* (bread, butter)

The FP-Tree would store common parts (like "milk → bread" shared) only once in the tree, saving memory and time.

# In short:

|  |  |
| --- | --- |
| **Apriori** | **FP-Tree** |
| Generates many candidates | No candidate generation |
| Scans database many times | Only two database scans |
| Slow for large data | Much faster for large data |

**Simple FP-Growth algorithm steps:**

1. Scan database once to find item frequencies.
2. Build an FP-Tree from transactions (ordered by frequency).
3. Recursively mine patterns from the tree (using smaller conditional trees).

### ****Apriori Algorithm****

**Definition**:  
The **Apriori Algorithm** is a classic algorithm used to **find frequent itemsets** and **generate association rules** from transactional data. It works based on the **"Apriori Property"**:

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

### 📘 ****Steps of Apriori Algorithm****:

1. **Scan the dataset** to find all **frequent 1-itemsets** (items that meet the minimum support).
2. **Generate candidate itemsets** of size k (called Ck) from frequent itemsets of size k-1 (called Lk-1).
3. **Count support** for each candidate itemset in Ck.
4. **Prune** the candidates that have support less than the minimum threshold.
5. **Repeat** until no more frequent itemsets are found.
6. **Generate association rules** from the frequent itemsets by calculating confidence and lift.

### 🔍 ****Example****:

Let’s say we have these transactions:

makefile

CopyEdit

T1: Milk, Bread

T2: Milk, Diaper, Beer, Eggs

T3: Milk, Diaper, Beer, Coke

T4: Bread, Milk, Diaper, Beer

T5: Bread, Milk, Diaper, Coke

If minimum support is 60%, Apriori may find:

* Frequent itemsets:
  + {Milk}, {Diaper}, {Beer}, {Milk, Diaper}, {Diaper, Beer}
* Association rule:
  + {Milk, Diaper} → {Beer}

### ✅ ****Advantages****:

* Simple to understand and implement.
* Uses **support** to efficiently reduce the search space.
* Works well for **market basket analysis**.

### ❌ ****Disadvantages****:

* Can be **slow on large datasets** due to multiple database scans.
* Generates many **candidate itemsets**, even if they are later discarded.

### ****Why FP-Tree?****

* Traditional methods like **Apriori** generate many candidate itemsets → slow and memory-intensive.
* FP-Tree avoids this by compressing the dataset and mining frequent patterns **directly**.

### ⚙️ ****How FP-Tree Works****:

1. **Scan 1 (Find Frequent Items)**:  
   Count how often each item appears in the dataset (get support count).  
   Discard items that are **below the support threshold**.
2. **Scan 2 (Build the Tree)**:
   * Sort items in each transaction by **descending frequency**.
   * Insert transactions into the tree, **sharing common prefixes**.
   * Each node stores:
     + Item name
     + Count
     + Link to next node with same item (via header table)
3. **Mining Patterns**:
   * Traverse the tree **bottom-up**.
   * Extract **conditional patterns** for each item.
   * Recursively mine smaller FP-trees (called **conditional FP-trees**).

### 📌 ****Example****:

Dataset (after filtering infrequent items):

less

CopyEdit

Transaction 1: A, B, C

Transaction 2: A, C

Transaction 3: B, C

Transaction 4: A, B

Sorted transactions (based on frequency):

makefile

CopyEdit

T1: A, B, C

T2: A, C

T3: B, C

T4: A, B

FP-Tree (simplified):

mathematica

CopyEdit

(root)

/

A:3

/ \

B:2 C:1

|

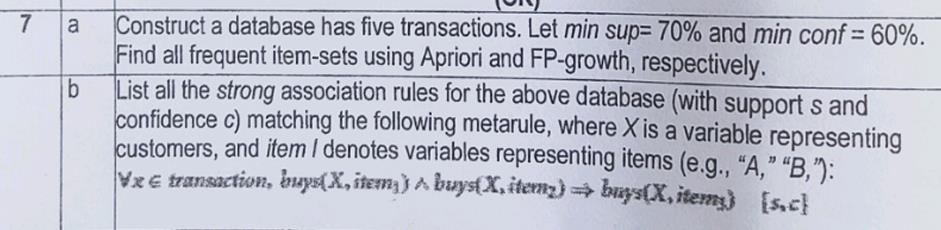
C:1

### ✅ ****Advantages****:

* No need to generate candidate sets.
* More efficient than Apriori for large datasets.
* Compact structure — reduces database size.

### ❌ ****Disadvantages****:

* More complex to implement.
* Tree can become large in dense datasets.



1. **a)**

**Step 1: Construct a database (5 transactions)**

|  |  |
| --- | --- |
| **TID** | **Items** |
| 1 | A, B, C |
| 2 | A, B |
| 3 | A, C |
| 4 | B, C |
| 5 | A, B, C |

**Given:**

* + Minimum Support (min sup) = 70% → **Support count must be ≥ 70% of 5 = 3.5 → 4 transactions.**
  + Minimum Confidence (min conf) = 60%

# Using Apriori

**Step 2: Find frequent 1-itemsets (single items)**

|  |  |
| --- | --- |
| **Item** | **Support Count** |
| A | 4 |
| B | 4 |
| C | 4 |

* + All have support count 4 → all are **frequent**.

# Step 3: Find frequent 2-itemsets

Possible 2-itemsets:

* + {A, B}
  + {A, C}
  + {B, C}

Check support:

|  |  |
| --- | --- |
| **Itemset** | **Support Count** |
| {A, B} | 3 |
| {A, C} | 3 |
| {B, C} | 4 |

**Only {B, C}** has support count 4 (≥ 4), so **only {B, C} is frequent**.

# Step 4: Find frequent 3-itemsets

Only possible 3-itemset is {A, B, C}.

Support for {A, B, C} = 2 (only TID 1 and 5) → not enough (needs 4). So **no frequent 3-itemsets**.

# ⬛ Frequent Itemsets using Apriori:

* + {A}
  + {B}
  + {C}
  + {B, C}

# Step 5: Generate Association Rules

From {B, C}:

Possible rules:

* + B → C
  + C → B Calculate confidence:
  + Confidence(B → C) = Support(B, C) / Support(B) = 4/4 = **100%** ⬛
  + Confidence(C → B) = Support(B, C) / Support(C) = 4/4 = **100%** ⬛

(Both above 60% → **valid rules**.)

# Now Using FP-Growth Step 1: First database scan:

* + Find frequencies:
    - A: 4
    - B: 4
    - C: 4

All are frequent.

# Step 2: Build FP-Tree:

Transactions (ordered by frequency A > B > C):

|  |  |
| --- | --- |
| **TID** | **Sorted Transaction** |
| 1 | A, B, C |
| 2 | A, B |
| 3 | A, C |
| 4 | B, C |

|  |  |
| --- | --- |
| **TID** | **Sorted Transaction** |
| 5 | A, B, C |

* + Build a tree:
    - A → B → C (TID 1)
    - A → B (TID 2)
    - A → C (TID 3)
    - B → C (TID 4)
    - A → B → C (TID 5 again) (Frequent paths visible.)

# Step 3: Mining from FP-Tree:

* + Find frequent patterns:
    - {A}: 4
    - {B}: 4
    - {C}: 4
    - {B, C}: 4

⬛ Same frequent itemsets as Apriori.

# Final Answer:

**Frequent Itemsets (both Apriori and FP-Growth):**

* + {A}
  + {B}
  + {C}
  + {B, C}

# Strong Association Rules:

* + B → C (confidence 100%)
  + C → B (confidence 100%)

# b)

**Strong Association Rules**

(only rules from frequent itemsets)

**Support threshold** = 70%

# Confidence threshold = 60%

From {A, B} (because {A} and {B} are frequent):

# Possible Rules:

* **Rule 1**: A → B
* **Rule 2**: B → A Find **confidence**:
* Confidence(A → B) = support(A, B) / support(A) = 5/5 = **100%**
* Confidence(B → A) = support(A, B) / support(B) = 5/5 = **100%**

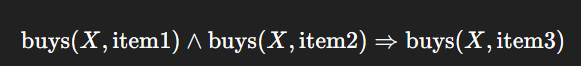
⬛ Both rules are **strong** (confidence ≥ 60%).

**Support** for both rules = 5/5 = 100%

**Confidence** for both rules = 100%

# Based on Metarule:

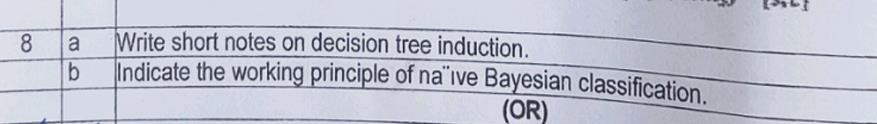
They want:



Here, there is no direct 3-itemset to split like that (because {A, B, C} was NOT frequent). So, **no rules** from 3 items.

Thus, final strong rules are:

|  |  |  |
| --- | --- | --- |
| **Rule** | **Support** | **Confidence** |
| A → B | 100% | 100% |
| B → A | 100% | 100% |



# a)

### What is Decision Tree Induction?

Decision tree induction is a supervised machine learning technique used for classification and regression. It builds a tree structure where internal nodes represent decisions (based on feature values), branches represent outcomes of those decisions, and leaf nodes represent final outputs (class labels or predicted values).

Think of it as a flowchart-like structure for making decisions based on data.

### 🌲 Key Components of a Decision Tree

1. 🔹 Root Node: The topmost node, representing the feature that best splits the data.
2. 🔹 Internal Nodes: Represent tests on features (e.g., "Age > 30?").
3. 🔹 Branches: The outcomes of a test (e.g., yes/no).
4. 🔹 Leaf Nodes: Final output (class label or prediction).

### 🧠 How Decision Tree Induction Works

1. Start with the entire training dataset.
2. Select the best attribute (feature) to split the data using a splitting criterion (like information gain or Gini index).
3. Split the data into subsets based on the chosen attribute.
4. Repeat the process recursively for each subset.
5. Stop when:
   * All records belong to the same class.
   * No more features are left to split.
   * A stopping condition is met (e.g., tree depth or minimum number of samples).

### 📊 Splitting Criteria

1. ✅ Information Gain (used in ID3 algorithm)
   * Based on entropy (measure of impurity).
   * Choose the feature that provides the highest reduction in entropy.
2. ✅ Gini Index (used in CART algorithm)
   * Measures impurity by probability of misclassification.
   * Lower Gini index is better.
3. ✅ Gain Ratio (used in C4.5)
   * Improves information gain by taking intrinsic information into account.

### 🔄 Example

Suppose you want to predict whether a person buys a laptop based on Age and Income.

The tree might look like:

yaml

CopyEdit

Age > 30?

/ \

Yes No

/ \

Income > 50k? Don't Buy

/ \

Yes No

Buy Don't Buy

### ✅ Advantages

* Easy to understand and interpret.
* Requires little data preparation.
* Can handle both numerical and categorical data.

### ❌ Disadvantages

* Can easily overfit the training data.
* Small changes in data can lead to different trees.
* May not be optimal (greedy splitting approach).

1. **b)**

**Working Principle of Naive Bayesian Classification**

* + **Naive Bayesian Classification** is based on **Bayes’ Theorem**.
  + It predicts the **class** of a given data point based on **probabilities**.
  + It is called "**naive**" because it **assumes** that all attributes (features) are **independent**

of each other (which is often not true, but works well in practice).

# Bayes' Theorem:

P(Class∣Data)=P(Data∣Class)×P(Class)P(Data)P(\text{Class} | \text{Data}) =

\frac{P(\text{Data} | \text{Class}) \times P(\text{Class})}{P(\text{Data})}P(Class∣Data)=P(Data)P(Data∣Class)×P(Class)

Where:

* + P(Class∣Data)P(\text{Class} | \text{Data})P(Class∣Data) = Probability that the data belongs to the class (this is what we want to find).
  + P(Data∣Class)P(\text{Data} | \text{Class})P(Data∣Class) = Probability of seeing the data given the class.
  + P(Class)P(\text{Class})P(Class) = Probability of the class itself.
  + P(Data)P(\text{Data})P(Data) = Probability of the data itself.

# Working Steps:

1. **Calculate** the prior probability P(Class)P(\text{Class})P(Class) for each class.
2. **Calculate** the likelihood P(Data∣Class)P(\text{Data} | \text{Class})P(Data∣Class) for each attribute assuming independence.
3. **Apply** Bayes’ theorem to compute P(Class∣Data)P(\text{Class} |

\text{Data})P(Class∣Data) for each class.

1. **Choose** the class with the **highest probability** as the prediction.

# Simple Example:

Suppose you want to predict if an email is **Spam** or **Not Spam** based on the word "Free" appearing.

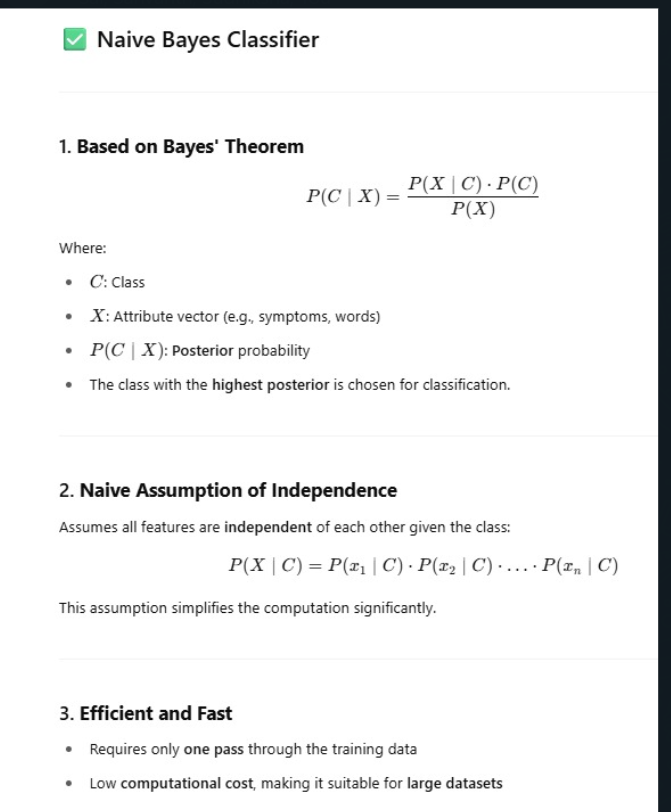
Given:

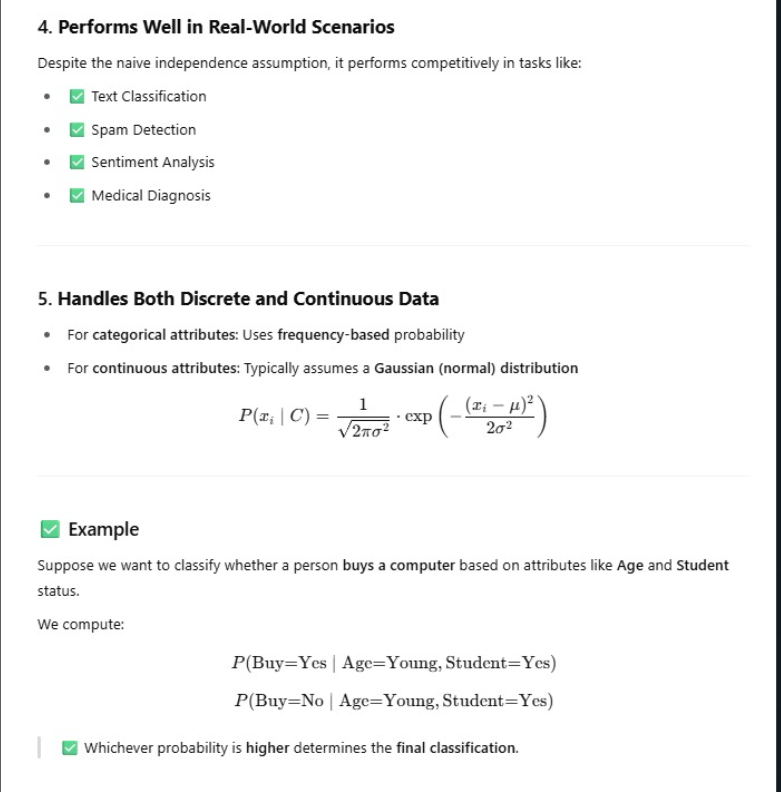
* + 80% of spam emails have "Free".
  + 10% of non-spam emails have "Free".
  + 30% of emails are spam.

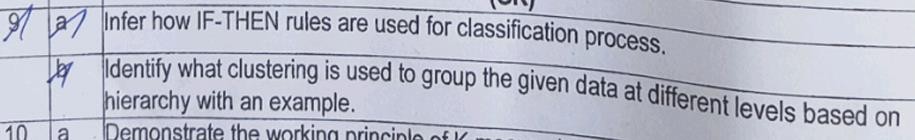
Using Bayes’ theorem, you calculate the probability for both "Spam" and "Not Spam" and pick the higher one.

**In Short:**

**Naive Bayes predicts the most likely class by calculating probabilities and assuming that all features are independent.**

****

****



1. **a)**

**IF-THEN Rules for Classification**

* + **IF-THEN rules** are a very intuitive way to **classify** data.
  + They work by **matching** a set of conditions (the "IF" part) and **assigning** a class label (the "THEN" part).
  + Each rule has the form:

IF (condition) THEN (class)\text{IF (condition) THEN (class)}IF (condition) THEN (class)

# How it works:

1. A set of **rules** is generated from training data.
2. For a **new instance**, the system checks which rule’s condition matches.
3. If a rule matches, the instance is classified into the **class** given in the "THEN" part.

# Example:

Suppose we are classifying whether a person will play tennis:

* + Rule 1:

**IF** (Outlook = Sunny) **AND** (Humidity = High) **THEN** Play = No

* + Rule 2:

**IF** (Outlook = Overcast) **THEN** Play = Yes

* + Rule 3:

**IF** (Outlook = Rain) **AND** (Wind = Strong) **THEN** Play = No

* + Rule 4:

**IF** (Outlook = Rain) **AND** (Wind = Weak) **THEN** Play = Yes

Now, for a new weather condition, we apply the matching rule to predict **Yes** or **No**.

# Advantages:

* + Very **easy to understand** and **interpret**.
  + **Flexible** — multiple rules can cover different parts of the data.

# Disadvantages:

* + Rules might **overlap** or **conflict**.
  + Need strategies to decide which rule to apply first (example: rule priority or most specific rule).

**In short:**

**IF-THEN rules classify by checking conditions on attributes and assigning a class if the conditions match.**

1. **b)**

**Clustering based on Hierarchy: Hierarchical Clustering**

* + **Hierarchical Clustering** is a method of clustering that builds a **tree-like structure** of groups (called a **dendrogram**).
  + It groups data **step-by-step**, either by **merging** smaller clusters into bigger ones (**agglomerative**) or by **splitting** bigger clusters into smaller ones (**divisive**).

# Types:

1. **Agglomerative (Bottom-Up)**:
   * Start with each data point as its own cluster.
   * Step-by-step, **merge** the closest clusters together.
   * End when all points are merged into one big cluster.

# Divisive (Top-Down):

* + Start with all data points in one big cluster.
  + Step-by-step, **split** clusters into smaller ones.
  + End when each point stands alone.

# Simple Example:

Imagine we have 4 points:

# A, B, C, D

Suppose distances between points are:

* A-B: 2
* A-C: 6
* A-D: 10
* B-C: 5
* B-D: 9
* C-D: 4

# Agglomerative clustering steps:

1. Start: A, B, C, D are separate clusters.
2. Merge A and B (smallest distance = 2).
3. Merge C and D (distance = 4).
4. Merge (A,B) and (C,D) clusters together.

At each step, clusters form higher and higher levels — like a **hierarchy**.

# Final Dendrogram (tree structure looks like):

css

Copy code

(A,B,C,D)

/ \

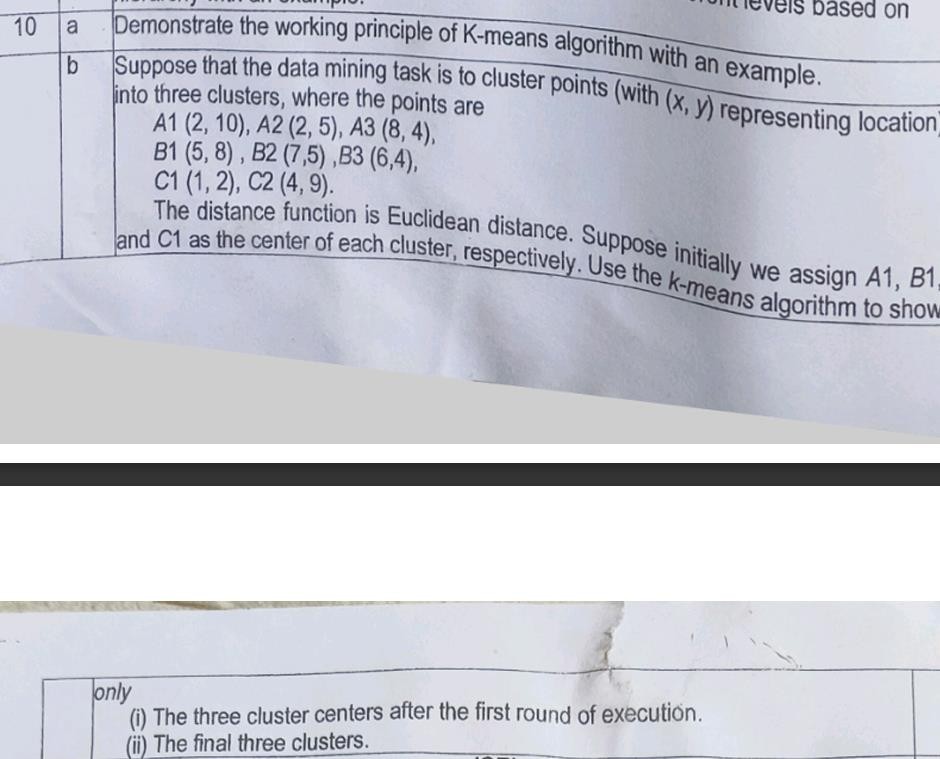
(A,B) (C,D)

/ \ / \

A B C D

# In short:

**Hierarchical clustering** groups data into a tree-like structure, either by merging small clusters or splitting big ones, showing relationships at different levels.



# a)

**K-Means Algorithm: Working Principle**

* + **K-Means** is a **clustering algorithm** that divides data into **K groups** (clusters).
  + It tries to **group similar points together** and find the "center" (called a **centroid**) of each cluster.

# How K-Means Works:

1. **Choose K** = number of clusters you want.
2. **Initialize K centroids** randomly.
3. **Assign** each data point to the **nearest centroid** (based on distance, usually Euclidean distance).
4. **Update centroids**: Calculate the **new center** (mean) of the points assigned to each cluster.
5. **Repeat** steps 3 and 4 until:
   * Centroids do not move much anymore, or
   * A maximum number of iterations is reached.

# Simple Example:

Imagine 6 points on a graph: (1,1),(2,1),(4,3),(5,4),(6,5),(8,8)

We want to cluster them into **K = 2** clusters.

# Steps:

**Step 1: Initialization**

* Pick two random points as initial centroids (say (1,1) and (8,8)).

# Step 2: Assignment

* Assign each point to the nearest centroid:

o (1,1) → closer to (1,1)

o (2,1) → closer to (1,1)

o (4,3) → closer to (1,1)

o (5,4) → closer to (8,8)

o (6,5) → closer to (8,8)

o (8,8) → closer to (8,8)

# Step 3: Update Centroids

* Find the new mean of each cluster:

o Cluster 1 (points: (1,1), (2,1), (4,3)) → new centroid = (2.33, 1.66)

o Cluster 2 (points: (5,4), (6,5), (8,8)) → new centroid = (6.33, 5.66)

# Step 4: Reassignment

* Reassign points to the new centroids.
* Update centroids again.
* Repeat until assignments do not change.

# Final Clusters:

* Cluster 1: (1,1), (2,1), (4,3)
* Cluster 2: (5,4), (6,5), (8,8)

**In Short:**

**K-Means groups data into K clusters by moving centroids and reassigning points until everything is stable.**

### Applications of K-Means

* Customer Segmentation
* Image Compression
* Document Clustering
* Anomaly Detection

### Advantages of K-Means

* Simple and Fast
* Scalable for large datasets
* Efficient convergence
* Works well with well-separated clusters

### Disadvantages of K-Means

* Need to choose number of clusters (K) beforehand
* Sensitive to outliers and noise
* Assumes spherical, equal-sized clusters
* Can get stuck in local minimum depending on initialization

1. **b)**

**Given:**

Points:

* + A1 (2,10)
  + A2 (2,5)
  + A3 (8,4)
  + B1 (5,8)
  + B2 (7,5)
  + B3 (6,4)
  + C1 (1,2)
  + C2 (4,9)

Clusters: Initially assigned centroids:

* + **Cluster 1**: A1 (2,10)
  + **Cluster 2**: B1 (5,8)
  + **Cluster 3**: C1 (1,2)

Distance function: **Euclidean distance**

# Step 1: Assignment (First Iteration)

We calculate the distance of each point to the 3 centroids.

# Distance formula:

d=(x2−x1)2+(y2−y1)2d = \sqrt{(x\_2 - x\_1)^2 + (y\_2 - y\_1)^2}d=(x2−x1)2+(y2−y1)2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Point** | **To A1 (2,10)** | **To B1 (5,8)** | **To C1 (1,2)** | **Nearest Cluster** |
| A1(2,10) | 0 | √(9+4) = 3.6055 | √(1+64) = 8.0623 | **Cluster 1** |
| A2(2,5) | √(0+25) = 5 | √(9+9) = 4.2426 | √(1+9) = 3.1623 | **Cluster 3** |
| A3(8,4) | √(36+36) = 8.4852 | √(9+16) = 5 | √(49+4) = 7.0711 | **Cluster 2** |
| B1(5,8) | √(9+4) = 3.6055 | 0 | √(16+36) = 7.2111 | **Cluster 2** |
| B2(7,5) | √(25+25) = 7.0711 | √(4+9) = 3.6055 | √(36+9) = 6.7082 | **Cluster 2** |
| B3(6,4) | √(16+36) = 7.2111 | √(1+16) = 4.1231 | √(25+4) = 5.3851 | **Cluster 2** |
| C1(1,2) | √(1+64) = 8.0623 | √(16+36) = 7.2111 | 0 | **Cluster 3** |
| C2(4,9) | √(4+1) = 2.2361 | √(1+1) = 1.4142 | √(9+49) = 7.8102 | **Cluster 2** |

# Assignment after 1st Iteration:

* + **Cluster 1**: A1 (2,10)
  + **Cluster 2**: B1 (5,8), A3 (8,4), B2 (7,5), B3 (6,4), C2 (4,9)
  + **Cluster 3**: C1 (1,2), A2 (2,5)

# Step 2: Update Centroids

Find the **new centroid** (mean point) for each cluster.

# New centroids:

* + **Cluster 1**:
    - Only A1(2,10)
    - New centroid = (2,10)

# Cluster 2:

* + - Points: (5,8), (8,4), (7,5), (6,4), (4,9)
    - Mean X = (5+8+7+6+4)/5 = 30/5 = 6
    - Mean Y = (8+4+5+4+9)/5 = 30/5 = 6
    - New centroid = (6,6)

# Cluster 3:

* + - Points: (1,2), (2,5)
    - Mean X = (1+2)/2 = 1.5
    - Mean Y = (2+5)/2 = 3.5
    - New centroid = (1.5, 3.5)

# Step 3: Assignment (Second Iteration)

Now reassign points based on new centroids: (2,10), (6,6), (1.5,3.5)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Point** | **To (2,10)** | **To (6,6)** | **To (1.5,3.5)** | **Nearest Cluster** |
| A1(2,10) | 0 | √(16+16) = 5.6569 | √(0.25+42.25) = 6.5192 | **Cluster 1** |
| A2(2,5) | √(0+25) = 5 | √(16+1) = 4.1231 | √(0.25+2.25) = 1.5811 | **Cluster 3** |
| A3(8,4) | √(36+36) = 8.4852 | √(4+4) = 2.8284 | √(42.25+2.25) = 6.5574 | **Cluster 2** |
| B1(5,8) | √(9+4) = 3.6055 | √(1+4) = 2.2361 | √(12.25+20.25) = 6.5 | **Cluster 2** |
| B2(7,5) | √(25+25) = 7.0711 | √(1+1) = 1.4142 | √(30.25+2.25) = 5.8309 | **Cluster 2** |
| B3(6,4) | √(16+36) = 7.2111 | √(0+4) = 2 | √(20.25+2.25) = 4.7434 | **Cluster 2** |
| C1(1,2) | √(1+64) = 8.0623 | √(25+16) = 6.4031 | √(0.25+2.25) = 1.5811 | **Cluster 3** |
| C2(4,9) | √(4+1) = 2.2361 | √(4+9) = 3.6055 | √(6.25+30.25) = 6.4031 | **Cluster 1** |

# New assignment:

* + **Cluster 1**: A1 (2,10), C2 (4,9)
  + **Cluster 2**: B1 (5,8), A3 (8,4), B2 (7,5), B3 (6,4)
  + **Cluster 3**: C1 (1,2), A2 (2,5)

# Step 4: Check if centroids changed

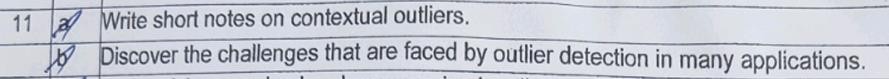
* + Clusters are **stable** now (no major change compared to previous step).
  + So **Final clusters** are:

´ ’◎●" **Final 3 Clusters:**

* + **Cluster 1**: A1(2,10), C2(4,9)
  + **Cluster 2**: B1(5,8), A3(8,4), B2(7,5), B3(6,4)
  + **Cluster 3**: C1(1,2), A2(2,5)

**Summary:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Iteration** | **Cluster 1** | **Cluster 2** | **Cluster 3** |
| After 1st Iteration | A1(2,10) | B1(5,8), A3(8,4), B2(7,5), B3(6,4), C2(4,9) | C1(1,2),  A2(2,5) |
| Final | A1(2,10),  C2(4,9) | B1(5,8), A3(8,4), B2(7,5), B3(6,4) | C1(1,2),  A2(2,5) |



1. **a)**

**Contextual Outliers (in Data Mining)**

* + A **contextual outlier** is a data point that is considered **normal** in some contexts but

**anomalous** in others.

* + It depends on the **contextual attributes** (like time, location, or conditions) and

**behavioral attributes** (actual values).

# Definition:

A data point that is unusual **with respect to the context** it appears in.

# Example:

* + A temperature of **30°C**:
    - **Normal** in **summer**.
    - **Outlier** in **winter**.
  + Context = season (summer or winter); behavior = temperature.

# Key Points:

* + Context matters — not just the value.
  + It is useful in applications like weather prediction, fraud detection, network monitoring, etc.
  + It is different from **global outliers** (which are always abnormal) and **collective outliers** (groups of points are unusual).

1. **b)**

**CHALLENGES IN OUTLIER DETECTION:**

1. **Modeling Normal and Outlier Objects**
   * **Challenge**: It's difficult to build a precise model for what counts as "normal" behavior.
   * **Why**: Normal behavior can vary widely, and it's hard to account for every possibility.
   * **Result**: Instead of labeling data strictly as "normal" or "outlier," some methods assign an **outlier score** to indicate the degree of abnormality.

# Application-Specific Requirements

* + **Challenge**: The meaning of an outlier can vary from one application to another.

# Examples:

* + - In **medical data**, even small deviations may be critical.
    - In **marketing**, larger variations are often acceptable.
  + **Implication**: A **one-size-fits-all** outlier detection method doesn't work. Each application might need a **custom approach** based on context.

# Noise vs. Outliers

* + **Challenge**: Real-world data is noisy (errors, missing values), which can confuse the detection process.

# Key Difference:

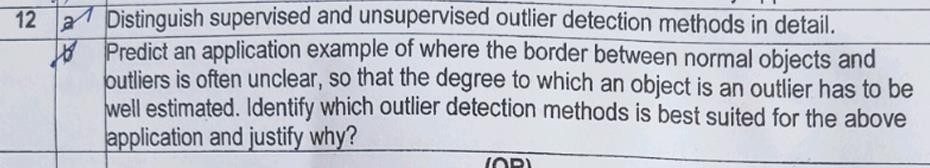
* + - **Noise**: Random errors or irrelevant data.
    - **Outliers**: Rare but meaningful anomalies.
  + **Problem**: Noise can:

# Hide real outliers.

* + - **Mimic outliers**, leading to **false positives**.

# Understandability

* + **Challenge**: Users may want to know **why** a data point was flagged as an outlier.
  + **Solution**: Good methods provide **justification**, such as:
    - Statistical reasoning (e.g., low likelihood of being generated by the same process as normal data).



# a)

⬛ **Supervised Methods**

* + **What it is**: Uses labeled data (normal vs. outlier) to **train a classifier**.
  + **Approach**: Treats outlier detection as a **classification problem**.

# Challenge:

* + - Outliers are **rare**, so the data is **imbalanced**.
    - It’s hard to get a good variety of outlier examples.
    - Sometimes uses techniques like **oversampling** or **artificial outlier generation**.
  + **Focus**: Usually more on **recall** (catching as many outliers as possible) than accuracy.

**Example**: In fraud detection, train a model on labeled fraudulent and normal transactions to detect new fraudulent ones.

# .’t Unsupervised Methods

* + **What it is**: No labeled data. Relies on the assumption that **normal data forms patterns (clusters)**, and **outliers don’t**.

# Approach:

* + - Objects that **don’t fit any pattern or cluster** are flagged as outliers.

# Limitations:

* + - Doesn’t work well when **normal data is diverse** or **outliers are similar to each other**.
    - High risk of **false positives** or **missing true outliers**.
  + **Examples of methods**: Clustering, proximity-based detection (discussed in later sections).

**Example**: In network traffic monitoring, unusual activity that doesn't fit the majority patterns is flagged—without any labels.

1. **b)**

**Application Example:**

**Credit Card Fraud Detection**

In credit card transactions, it is often difficult to clearly define the border between normal and fraudulent activities. Some transactions may appear slightly suspicious but not clearly fraudulent, and the degree of deviation from normal behavior must be carefully estimated.

**Best-Suited Outlier Detection Method: LOF (Local Outlier Factor)**

**Justification:**

* + **Handles Unclear Boundaries:**

LOF does not rely on a fixed global threshold; instead, it estimates how isolated a data point is with respect to its neighbors, which is ideal when boundaries are fuzzy.

# Degree of Outlierness:

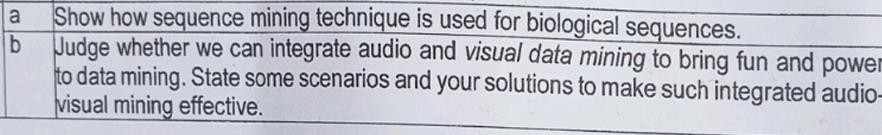
It assigns a **score** that indicates the degree to which an object is an outlier, which helps in ranking and identifying borderline cases.

# Local Context Awareness:

Unlike global methods, LOF considers local density, making it effective in datasets with varying densities — such as legitimate vs. suspicious transactions in different customer segments.

# Flexible and Non-Parametric:

LOF is non-parametric and works well without assuming any specific distribution, which fits real-world scenarios like financial data.



# a)

**First**, what is **sequence mining**?

Sequence mining means **finding patterns** that happen **often** in a **sequence of things**. Like in shopping:

* + Monday: Bread → Butter → Milk
  + Tuesday: Bread → Butter → Jam
  + It sees "Bread → Butter" is common.

**Now, in biology,** the sequences are things like:

* + **DNA sequences** (A, T, G, C letters)

# RNA sequences

* + **Protein sequences** (made of amino acids like A, C, G, T, etc.) Example of DNA: A T G C G A T

# How sequence mining is used in biological sequences:

|  |  |
| --- | --- |
| **Step** | **Simple Explanation** |
| 1. **Get biological data** | Collect lots of DNA, RNA, or protein sequences. |
| 2. **Apply sequence mining** | Use algorithms to find common **patterns**. For example:   * Maybe "A T G" often appears together. * Or "G A T" appears in many genes. |
| 3. **Understand the patterns** | These patterns might show:   * Important **gene functions** * **Mutation spots** * **Protein functions** * **Disease indicators** |
| 4. **Use the patterns** | Scientists use these to:   * Predict diseases * Discover new medicines * Understand how life works at a small level |

**Simple Example in Biology:**

Suppose you have these DNA sequences:

css

Copy code

1. A T G C G A T
2. A T G A A T
3. A T G G C A

**Sequence mining** notices that **A → T → G** happens **again and again** in all sequences!

➡ So, **A → T → G** is an **important pattern**.

# Famous sequence mining techniques used in biology:

* + **Apriori algorithm** (find frequent patterns)
  + **PrefixSpan algorithm** (find frequent subsequences)
  + **SPADE algorithm** (fast pattern mining)

# In very short:

In biological sequences, **sequence mining** finds important **patterns** that help in understanding **genes**, **proteins**, **mutations**, and even **diseases**.

# 13)b)

Integrating **audio** (sound) and **visual** (images/videos) data mining can make data mining

**more fun**, **more powerful**, and **more useful**.

* **Audio mining** = finding patterns in sounds (speech, music, noises).
* **Visual mining** = finding patterns in images/videos (faces, scenes, movements).
* Together ➔ **Audio-Visual mining** = smarter, deeper insights!

●◎’" ´ **Why integrate them?**

* **Human understanding is multi-sensory**: We don’t just see or hear — we do both!
* **Richer data**: Sounds and visuals together give **more clues**.
* **Better decisions**: More accurate results by combining senses.

|  |  |  |
| --- | --- | --- |
| **Scenario** | **What Happens** | **Your Solution** |
| `~fz’ˆ—˜ ˘ **Movie recommendation system** | Mining both dialogues (audio) and scenes (visual) to suggest movies people will love. | * Mine emotional tone from audio (happy, sad). * Analyze scene brightness/colors. * Recommend based on combined "mood" and "visual style." |
| ⎛¡˙ \_ – **Healthcare diagnosis** | Analyzing patient speech (audio) + facial expressions (visual) for mental health diagnosis. | * Use audio mining for voice stress. * Visual mining for facial micro-expressions. * Combine both for better diagnosis. |
| 2\_.¨ ›•< **Self-driving cars** | Detecting sounds (horns, sirens) and visuals (pedestrians, traffic signs). | * Train AI to process road sounds + video frames together. * Make faster and smarter   driving decisions. |

|  |  |  |
| --- | --- | --- |
| **Scenario** | **What Happens** | **Your Solution** |
| · \* ˙·\*·˙ **Games and VR** | Games that react to player's voice tone and face expressions! | * Capture happy/shocked sounds. * Detect smile or frown. * Change the game scene according to player emotion! |
| ˆ,·z,‘·μ±•.·ˆ-.ˆo·˙\*ˆ^†.'x¡T **Security and surveillance** | Recognizing dangerous situations (like a scream + people running). | * Analyze sudden loud audio events. * Detect fast moving objects in camera. * Alert security if both happen   together. |

# How to make audio-visual mining effective (solutions):

1. **Synchronize data**
   * Match audio and video exactly in time. (e.g., "this scream happened when that person ran.")

# Use deep learning

* + Use **CNNs** for images and **RNNs/LSTMs** for audio signals, and **combine** their outputs.

# Feature fusion

* + Extract important features separately from audio and visual, then **merge** them before mining.

# Context-aware mining

* + Understand the background: e.g., a scream in a concert is different from a scream in a hospital.

# Interactive visualizations

* + Create fun dashboards showing audio and visual patterns together (easy to understand and act upon).

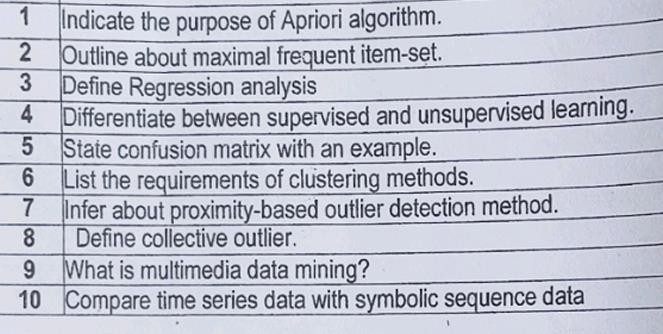
# Real-time processing

* + For games, cars, security — the mining must be **super-fast** (stream mining).

' ` - **In short:**

# Audio + Visual mining = more smart, fun, powerful data mining.

It needs **good synchronization**, **feature fusion**, and sometimes **AI models** to work well.



# Indicate the purpose of Apriori algorithm.

The Apriori algorithm is used to identify frequent item sets in a database and derive association rules for market basket analysis. It helps in finding patterns, such as items frequently bought together.

# Outline about maximal frequent item-set.

A maximal frequent item-set is a frequent item-set that has no frequent superset. In other words, it is a frequent set beyond which no item can be added without reducing its frequency below the minimum threshold.

# Define Regression analysis.

Regression analysis is a statistical method for modeling the relationship between a dependent (target) variable and one or more independent (predictor) variables, mainly to predict or forecast outcomes.

# Differentiate between supervised and unsupervised learning.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Supervised Learning** | **Unsupervised Learning** |
| Definition | Learning with labeled data | Learning with unlabeled data |
| Purpose | Predict outcomes or classify | Discover hidden patterns |
| Example | Email spam detection | Customer segmentation |

1. **State confusion matrix with an example.**

A confusion matrix is a table used to evaluate the performance of a classification model by comparing actual vs. predicted classifications.

Example for binary classification:

|  |  |  |
| --- | --- | --- |
|  | **Predicted Positive** | **Predicted Negative** |
| Actual Positive | True Positive (TP) | False Negative (FN) |
| Actual Negative | False Positive (FP) | True Negative (TN) |

# List the requirements of clustering methods.

* + Scalability
  + Ability to deal with different types of attributes
  + Discovery of clusters with arbitrary shapes
  + Minimal requirements for domain knowledge
  + Ability to handle noise and outliers
  + High-dimensionality handling
  + Interpretability and usability

# Infer about proximity-based outlier detection method.

Proximity-based outlier detection identifies an object as an outlier if it is far away from its neighboring points based on a distance metric. Common techniques include k-nearest neighbors (k-NN) and distance-based methods.

# Define collective outlier.

A collective outlier refers to a group of data instances that collectively deviate from the expected pattern, even though individually they may not be considered outliers. Example: A sudden change in network traffic behavior.

# What is multimedia data mining?

Multimedia data mining is the process of extracting useful and interesting patterns from large collections of multimedia data types like text, image, audio, and video.

1. **Compare time series data with symbolic sequence data.**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Time Series Data** | **Symbolic Sequence Data** |
| Nature | Continuous data over time | Discrete symbols in a sequence |
| Example | Stock prices, weather data | DNA sequences, clickstream data |
| Analysis Focus | Trends, seasonal patterns | Pattern matching, sequence mining |