# Data-Mining-Module-3-Important-Topics-PYQs

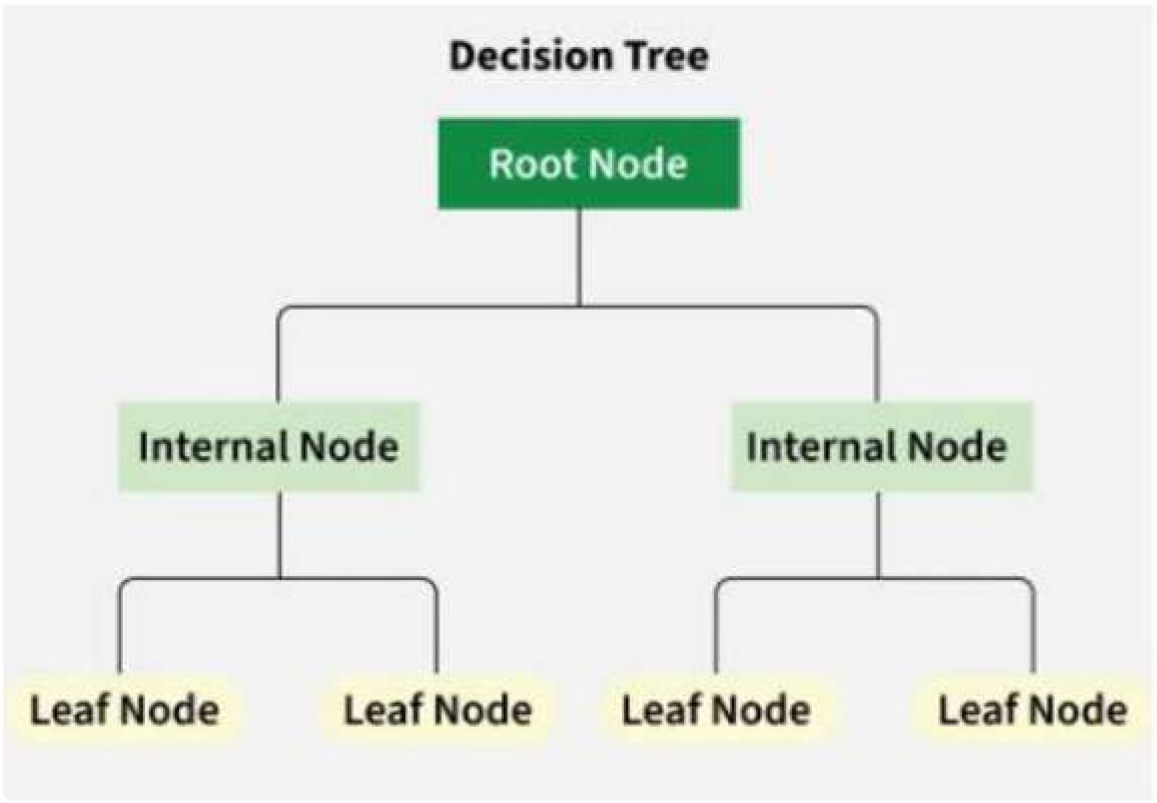
## *1. What are the requirements for a good clustering algorithm?*

1. **Scalability**
   It should work well even if we have a large amount of data (both in terms of rows and columns).
2. **Ability to handle different shapes and sizes**
   It should be able to form clusters of different shapes (round, long, etc.) and sizes (big or small).
3. **High Quality Clusters**
   Objects in the same cluster should be **similar** to each other, and objects in different clusters should be **different** from each other.

4. **Noise and Outlier Handling**
   It should deal well with unusual data points (called outliers) and not get confused by them.
5. **Minimal Requirements for Domain Knowledge**
   It should not require too much input like the number of clusters unless it's necessary.
6. **Interpretability**
   The output (clusters) should be easy to understand and explain.
7. **Deterministic Results**
   If we run it multiple times on the same data, it should give the same results (or almost the same).
8. **Minimal Parameters**
   It should not require setting too many parameters to work well.

---

## *2. Discuss the issues regarding the implementation of decision tree.*



A **decision tree** is like a **flowchart** that helps you make decisions or predictions by asking **yes/no questions** step by step.

- It starts at the **top** with one main question (called the **root node**).

- Based on the answer, it follows a **branch** to the next question.
- This continues until it reaches a **final answer** (called a **leaf node**).

### 1. Overfitting

- The tree becomes **too complex**, capturing even small noise in the data.
- It works well on training data but gives **poor results on new (test) data**.

### 2. Instability

- **Small changes** in the data can cause **big changes** in the tree structure.
- This makes the model **less reliable** in real-world scenarios.

### 3. Bias Toward Features with Many Categories

- If a feature has many different values (like zip codes), the tree might focus on it too much.
- This can **ignore other important features** and reduce accuracy.

### 4. Hard to Prune

- After building the tree, we often need to **remove unnecessary branches** to avoid overfitting.
- But deciding what to prune is **not always easy** and can affect performance.

### 5. Greedy ApproachIn SLIQ (and in decision trees in general), after we split once (for example, by Temperature), we again apply the splitting logic inside each branch.

- First split: based on **Temperature** (example: Temperature < 72)
- Now, **inside** the new group (after splitting on Temperature),
  ➔ **SLIQ checks again**: "Can we split this further using other attributes?"

That's how **Humidity < 80** comes **after** Temperature split!

---In **SLIQ** (and in decision trees in general), after we split once (for example, by **Temperature**), we **again apply the splitting logic inside each branch**.

- First split: based on **Temperature** (example: Temperature < 72)
- Now, **inside** the new group (after splitting on Temperature),
  ➔ **SLIQ checks again**: "Can we split this further using other attributes?"

That's how **Humidity < 80** comes **after** Temperature split!

- Decision trees use a **greedy method**, choosing the best split at each step.
- This doesn't always give the **best overall tree** (might miss better combinations).

**6. Data Fragmentation**

- With too many splits, the data in each node becomes **very small**.
- This can lead to **less reliable decisions** at the lower levels of the tree.

## *3. How is Gain Ratio calculated? What is the advantage of Gain Ratio over Information Gain?*

Gain Ratio is a measure used in decision trees to choose the **best attribute for splitting** the data. It improves on **Information Gain** by removing a common problem

### Steps to Calculate Gain Ratio:

1. **Step 1: Calculate Information Gain (IG)**
    1. $$IG(S, A) = Entropy(S) - \sum \left( \frac{|S_v|}{|S|} \times Entropy(S_v) \right)$$
    2. Where:
        1. S = full dataset
        2. A = attribute to split on
        3. Sv = subset of data where attribute A = value v
2. **Step 2: Calculate Split Information (Split Info)**
    1. $$SplitInfo(A) = -\sum \left( \frac{|S_v|}{|S|} \times \log_2 \left( \frac{|S_v|}{|S|} \right) \right)$$
3. **Step 3: Calculate Gain Ratio**

1. $$GainRatio(A) = \frac{Information\ Gain}{Split\ Info}$$

## Advantage of Gain Ratio over Information Gain:

### Problem with Information Gain:

Information Gain likes attributes that give a **clear split**, even if that split is **not useful**.
For example:

- Suppose you have a column like **Student ID**.
- Each student has a **unique ID**, so if you split based on that, each group will have only **one student**.
- This will give **high Information Gain**, because every group is "pure" (only one student = no uncertainty).
- But it’s **not helpful** for prediction, because we can’t use ID to predict anything about other students!

So, **Information Gain gets tricked** by attributes with many unique values.

### How Gain Ratio Fixes This:

Gain Ratio looks at **how many branches** a split creates.

- If an attribute creates **too many small groups**, Gain Ratio **penalizes** it.
- It prefers splits that are **useful and balanced**, not just ones that look good by splitting too much.

So, it helps us choose attributes that actually make **meaningful decisions**, not just those that make perfect but **useless** splits.

---

## *4. Given two objects are represented by the tuples (22,1,42,10) and (20, 0, 36, 8). Compute (i) Euclidean distance ii) Manhattan distance (iii) Minkowski distance of order 3.*

## Given:

Two objects (points):

- A = (22, 1, 42, 10)
- B = (20, 0, 36, 8)

## i) Euclidean Distance

Euclidean distance is the **straight-line distance** between two points:

$$\text{Euclidean} = \sqrt{(22-20)^2 + (1-0)^2 + (42-36)^2 + (10-8)^2}$$
$$= \sqrt{2^2 + 1^2 + 6^2 + 2^2} = \sqrt{4+1+36+4} = \sqrt{45} \approx 6.708$$

## ii) Manhattan Distance

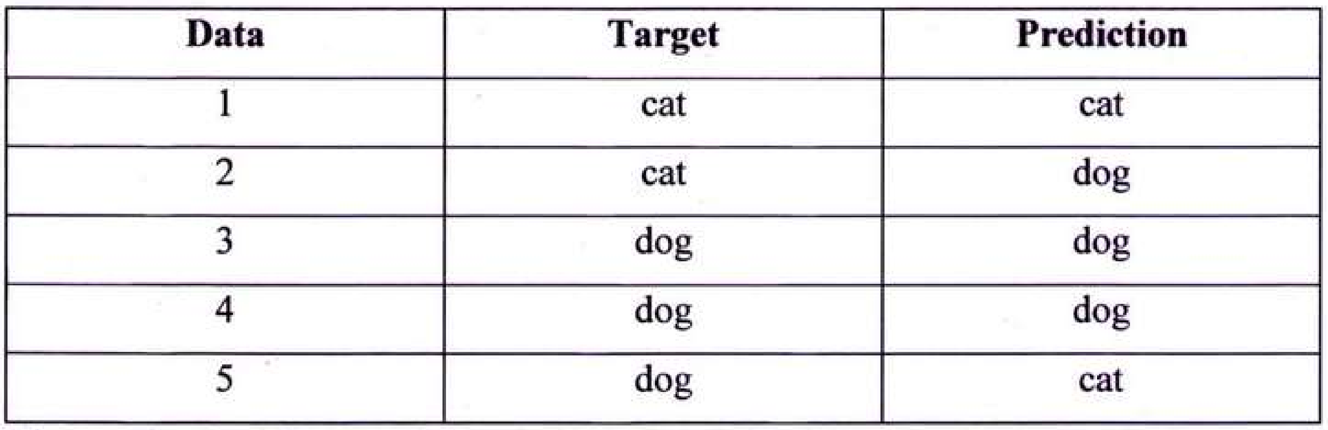
Manhattan distance is the **sum of absolute differences** of all features:

$$\text{Manhattan} = |22-20| + |1-0| + |42-36| + |10-8| = 2+1+6+2 = 11$$

## iii) Minkowski Distance of order 3 (p = 3)

- Minkowski formula:
  - $$\left(\sum |x_i - y_i|^p\right)^{1/p}$$
  - $$= \left(|2|^3 + |1|^3 + |6|^3 + |2|^3\right)^{1/3} = (8+1+216+8)^{1/3} = (233)^{1/3}$$
$$\approx 6.2$$
-

## *5. Draw the confusion matrix and calculate precision and recall of the given data.*



| Data | Target | Prediction |
|---|---|---|
| 1 | cat | cat |
| 2 | cat | dog |
| 3 | dog | dog |
| 4 | dog | dog |
| 5 | dog | cat |

### What are we solving?

You are given some data where the **actual** (target) values and the **predicted** values are compared. We want to evaluate how well the model is performing using terms like **precision** and **recall**, which we calculate from the **confusion matrix**.

### 1. What is a Confusion Matrix?

- A **confusion matrix** is a tool used to **evaluate the performance** of a classification model by showing the relationship between the actual and predicted values.
- It's like a **comparison table** where we can see how often the model gets predictions right or wrong.
- The **confusion matrix** has four main terms:
  - **True Positive (TP):** The model correctly predicted the positive class (e.g., Cat is predicted as Cat).
  - **False Positive (FP):** The model incorrectly predicted a negative as positive (e.g., Dog is predicted as Cat).
  - **True Negative (TN):** The model correctly predicted the negative class (e.g., Dog is predicted as Dog).
  - **False Negative (FN):** The model incorrectly predicted a positive as negative (e.g., Cat is predicted as Dog).

### 2. Create the Confusion Matrix

Let's look at your data and see how the actual and predicted values compare:

| Data Point | Target (Actual) | Prediction | Match? |
|---|---|---|---|
| 1 | Cat | Cat | ✅ (True Positive) |
| 2 | Cat | Dog | ❌ (False Negative) |
| 3 | Dog | Dog | ✅ (True Negative) |
| 4 | Dog | Dog | ✅ (True Negative) |
| 5 | Dog | Cat | ❌ (False Positive) |

From the above table, we can create the confusion matrix:

| Predicted Cat | Predicted Dog | |
|---|---|---|
| **Actual Cat** | 1 (TP) | 1 (FN) |
| **Actual Dog** | 1 (FP) | 2 (TN) |

## 3. Calculate Precision and Recall

**Precision:**

**Precision** is the proportion of correct positive predictions (Cat) out of all the predictions where the model predicted **Cat**.

> **Precision** tells us: "Of all the times the model predicted Cat, how many were actually Cat?"

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

From the confusion matrix:

- **TP** (True Positives) = 1 (The model correctly predicted Cat as Cat)
- **FP** (False Positives) = 1 (The model incorrectly predicted Dog as Cat)

$$\text{Precision} = \frac{1}{1+1} = \frac{1}{2} = 0.5$$

So, the **precision** is **0.5** (50%).

**Recall:**

**Recall** (also called **Sensitivity** or **True Positive Rate**) is the proportion of correct positive predictions (Cat) out of all **actual positives** (all the actual Cat cases).

> **Recall** tells us: "Of all the actual Cats, how many did the model correctly predict as Cat?"

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

From the confusion matrix:

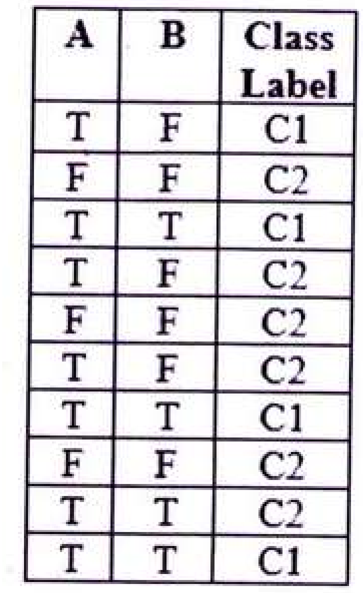
- **TP** (True Positives) = 1 (The model correctly predicted Cat as Cat)
- **FN** (False Negatives) = 1 (The model incorrectly predicted Cat as Dog)

$$\text{Recall} = \frac{1}{1+1} = \frac{1}{2} = 0.5$$

So, the **recall** is **0.5** (50%).

---

## *6. Consider the following dataset for a binary classification problem with class label as Cl and C2.*



| A | B | Class Label |
|---|---|---|
| T | F | C1 |
| F | F | C2 |
| T | T | C1 |
| T | F | C2 |
| F | F | C2 |
| T | F | C2 |
| T | T | C1 |
| F | F | C2 |
| T | T | C2 |
| T | T | C1 |

**i) Calculate the gain in Gini index when splitting on A and B respectively. which attribute would the decision tree induction algorlthm choose?**
**ii) Calculate the information gain when splitting on A and B. which attribute would the decision tree induction algorithm choose?**

## What are we solving?

We’re solving a **binary classification problem** where we have a dataset and we want to **split** the data based on two attributes (A and B). We're interested in two main things:

1. **Gini Index** - A measure of how often a randomly chosen element would be incorrectly classified. We want to see how splitting based on **A** and **B** affects this.
2. **Information Gain** - A measure of how much uncertainty (or impurity) is reduced when we split the data based on an attribute. We will compare the splits based on **A** and **B** to determine which one is the best.

## 1. Gini Index

### What is Gini Index?

- The **Gini Index** measures the "impurity" or "impurity of the dataset" after a split. If the Gini index is 0, it means the dataset is **pure**, i.e., all elements belong to the same class.
- The formula to calculate the Gini Index is:

- $$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2$$
- Where:
  - pi is the proportion of the class iii in the dataset.
- The Gini index is calculated for each subset of the data after splitting on an attribute, and then we calculate the **weighted average Gini index** across all subsets.
- Let's calculate the **Gini index** when splitting based on **A** and **B**.

| A | B | Class Label |
|---|---|---|
| T | F | C1 |
| F | F | C2 |
| T | T | C1 |
| T | F | C2 |
| F | F | C2 |
| T | F | C2 |
| T | T | C1 |
| F | F | C2 |
| T | T | C2 |
| T | T | C1 |

### Step 1: Calculate Gini Index when Splitting on A (Attribute A)

**Split the data based on A:**
When **A = T**: The subset is:

| A | B | Class Label |
|---|---|---|
| T | F | C1 |
| T | T | C1 |
| T | F | C2 |
| T | F | C2 |
| T | T | C1 |
| T | T | C2 |

We have 6 instances in this subset. The class distribution is:

- 3 instances of **C1**
- 3 instances of **C2**

Gini for **A = T**

$$Gini(T) = 1 - \left(\left(\frac{3}{6}\right)^2 + \left(\frac{3}{6}\right)^2\right) = 1 - (0.25 + 0.25) = 0.5$$

When **A = F**: The subset is:

| A | B | Class Label |
|---|---|---|
| F | F | C2 |
| F | F | C2 |
| F | F | C2 |
| F | F | C2 |
| F | F | C2 |

We have 5 instances in this subset, all labeled **C2**.
Gini for **A = F**:

$$Gini(F) = 1 - \left(\left(\frac{5}{5}\right)^2\right) = 1 - 1 = 0$$

**Weighted Gini Index for A**:

$$Gini(A) = \frac{6}{10} \times 0.5 + \frac{5}{10} \times 0 = 0.3$$

### Step 2: Calculate Gini Index when Splitting on B (Attribute B)

**Split the data based on B:**

When **B = F**: The subset is:

| A | B | Class Label |
|---|---|---|
| T | F | C1 |
| F | F | C2 |
| T | F | C1 |
| T | F | C2 |
| F | F | C2 |
| T | F | C2 |

We have 6 instances in this subset. The class distribution is:

- 2 instances of **C1**
- 4 instances of **C2**

Gini for **B = F**

$$Gini(F) = 1 - \left(\left(\frac{2}{6}\right)^2 + \left(\frac{4}{6}\right)^2\right) = 1 - (0.1111 + 0.4444) = 0.4444$$

When **B = T**: The subset is:

| A | B | Class Label |
|---|---|---|
| T | T | C1 |
| T | T | C1 |
| T | T | C1 |
| T | T | C2 |
| T | T | C2 |

We have 5 instances in this subset. The class distribution is:

- 3 instances of **C1**
- 2 instances of **C2**

Gini for **B = T**:

$$Gini(T) = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2\right) = 1 - (0.36 + 0.16) = 0.48$$

**Weighted Gini Index for B**:

$$Gini(B) = \frac{6}{10} \times 0.4444 + \frac{5}{10} \times 0.48 = 0.4622$$

### Which Attribute Would the Decision Tree Choose?

- **Gini Index for A** = 0.3
- **Gini Index for B** = 0.4622

Since the **lower the Gini index, the better** the split, the decision tree would choose **Attribute A** for the split because **A** has a lower Gini index (0.3).

## 2. Information Gain

### What is Information Gain?

**Information Gain** is another measure that tells us how much **uncertainty** is reduced after splitting the data based on an attribute. The higher the information gain, the better the attribute for splitting.

We calculate **Information Gain** using the formula:

$$IG(S,A) = Entropy(S) - \left(\sum_{v \in A} \frac{|S_v|}{|S|} \times Entropy(S_v)\right)$$

$S$ is the entire dataset.

$S_v$ are the subsets created by splitting on attribute $A$.

$|S_v|$ is the size of subset $S_v$.

$Entropy(S)$ is the entropy of the entire dataset.

### Step 1: Calculate Entropy for the Whole Dataset

There are 10 instances, with:

- 4 instances of **C1**
- 6 instances of **C2**

$$Entropy(S) = -\left(\frac{4}{10}\log_2\frac{4}{10} + \frac{6}{10}\log_2\frac{6}{10}\right)$$

$$Entropy(S) = -\left(0.4\log_2 0.4 + 0.6\log_2 0.6\right) \approx 0.971$$

### Step 2: Information Gain for Attribute A

➤ Split the dataset by A:

✅ **When A = T:**

6 entries → 3 C1, 3 C2
So,

$$Entropy(A = T) = -(0.5\log_2 0.5 + 0.5\log_2 0.5) = 1.0$$

### When A = F:

4 entries → 0 C1, 4 C2

$$Entropy(A = F) = -(1.0 \log_2 1.0 + 0 \log_2 0) = 0$$

Weighted Average Entropy after split on A:

$$Entropy_A = \frac{6}{10} \times 1.0 + \frac{4}{10} \times 0 = 0.6$$

Now, compute Information Gain for A:

$$IG(A) = Entropy(S) - Entropy_A = 0.971 - 0.6 = 0.371$$

### Step 3: Information Gain for Attribute B

Split the dataset by B:

When B = F:

6 entries → 2 C1, 4 C2

$$Entropy(B = F) = -\left(\frac{2}{6} \log_2 \frac{2}{6} + \frac{4}{6} \log_2 \frac{4}{6}\right) \approx -(0.333 \cdot -1.585 + 0.666 \cdot -0.585) \approx 0.918$$

When B = T:

4 entries → 2 C1, 2 C2

$$Entropy(B = T) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1.0$$

**Weighted Average Entropy after split on B:**

$$Entropy_B = \frac{6}{10} \times 0.918 + \frac{4}{10} \times 1.0 = 0.9508$$

**Now, compute Information Gain for B:**

$$IG(B) = Entropy(S) - Entropy_B = 0.971 - 0.9508 = 0.0202$$

## Final Decision:

| Attribute | Gini Index | Information Gain |
|---|---|---|
| A | **0.3** | **0.371** |
| B | 0.4622 | 0.0202 |

Both the **Gini Index** and the **Information Gain** suggest that **Attribute A** is better for splitting the data.

## Important formulas

### 1. Entropy (S)

Entropy measures how *mixed* or *pure* the dataset is.

$$Entropy(S) = -p_1 \log_2(p_1) - p_2 \log_2(p_2) - \dots$$

- $p_1, p_2, \dots$ = proportion of classes (like C1, C2) in S.
- Entropy is **0** when pure (only one class), and **high** when mixed.

### 2. Gini Index (S)

Gini Index measures the chance of incorrectly classifying a randomly chosen element.

$$Gini(S) = 1 - (p_1^2 + p_2^2 + \dots)$$

- $p_1, p_2, \dots$ = proportion of classes.
- Gini is **0** when pure (perfect classification).

### 3. Weighted Entropy after Split

(For any attribute A)

$$Entropy_A = \sum \left( \frac{|Subset|}{|Total|} \times Entropy(Subset) \right)$$

- Subsets are the groups formed when splitting on A.

### 4. Information Gain (IG)

Measures how much "confusion" (entropy) was reduced after the split.

$$IG(A) = Entropy(S) - Entropy_A$$

- Higher IG → Better attribute for splitting.

### 5. Gain Ratio

Used when attributes have many unique values.

$$Gain\ Ratio(A) = \frac{Information\ Gain(A)}{Split\ Information(A)}$$

Where:

- Split Information = Entropy based on how data is divided among different branches.

## *7. Explain the concept of DBSCAN argorithm along with its advantages.*

**DBSCAN** stands for:

**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise.

It’s a **clustering algorithm**—which means it groups similar data points together.

Unlike other clustering methods like **K-Means** (which needs to know the number of clusters), **DBSCAN finds clusters based on how tightly the data points are packed (density)**.

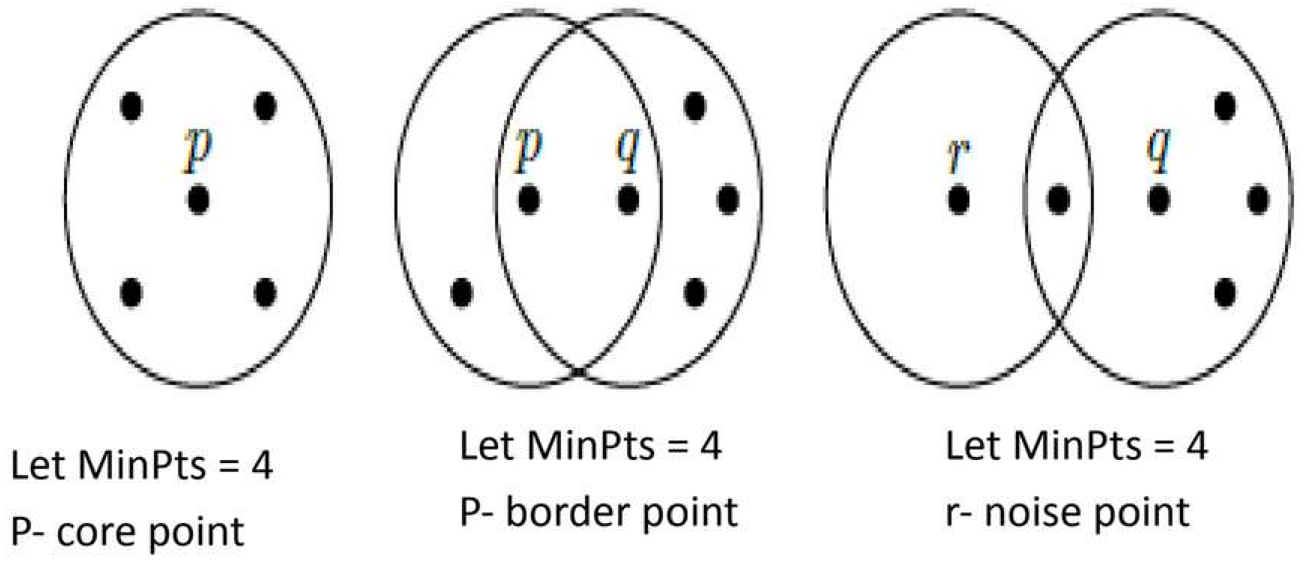
## Key Concepts

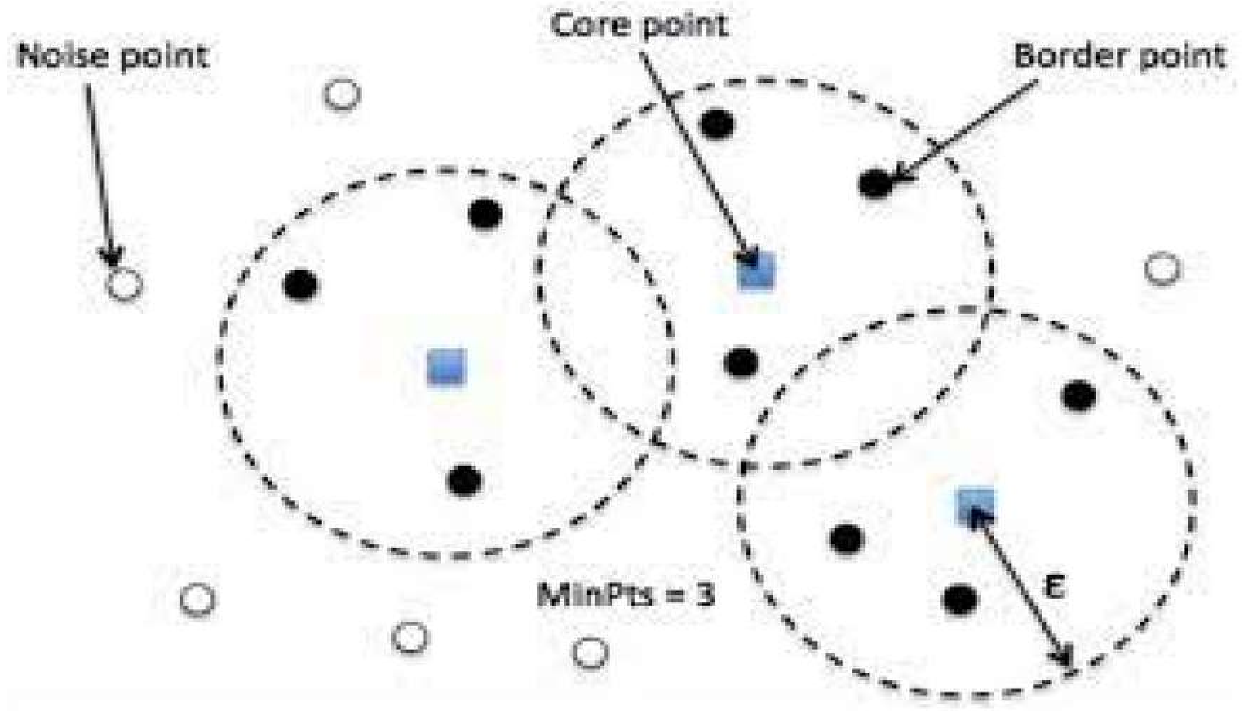
Imagine a map full of dots (data points). Some dots are close together (dense areas), and some are spread out (sparse areas). DBSCAN uses this idea of density to form clusters.

### DBSCAN needs two settings:

1. **Eps (ε):**
   This is the **distance** to look around each point.
   Like drawing a circle around a point — "Who’s close to me?"
2. **MinPts (Minimum Points):**
   The **minimum number of neighbors** required to say:
   “Yes, this is a dense region.”

## Types of Points in DBSCAN





1. **Core Point:**
   A point that has **at least MinPts** within its ε-radius.
   → It's in a dense area.
2. **Border Point:**
   Not enough neighbors to be core, **but still near a core point**.
   → It's on the edge of a cluster.
3. **Noise Point (Outlier):**
   Neither a core nor near any core.
   → It's alone and doesn't belong to any cluster.

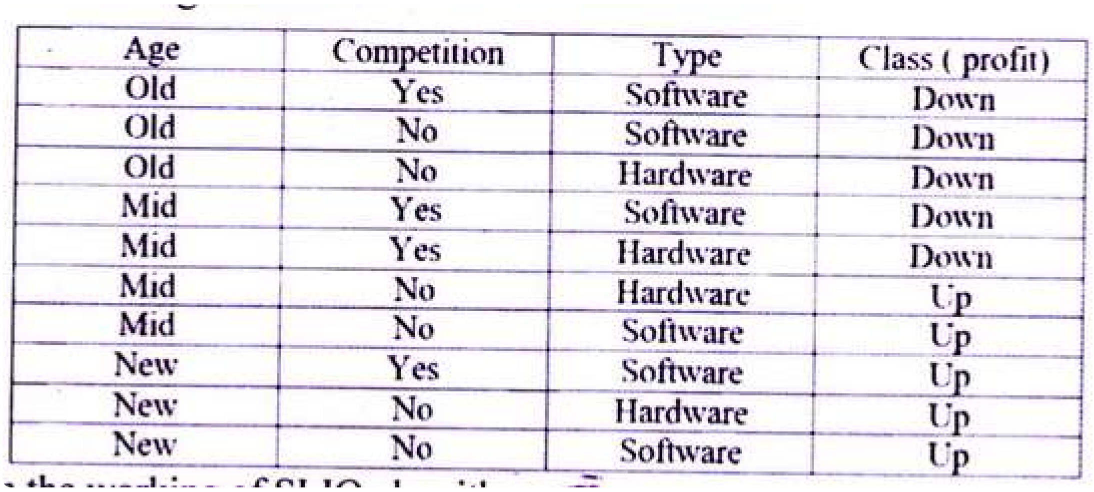
## How DBSCAN Works

1. Start with any point that hasn't been visited.
2. Check how many neighbors it has within ε distance.
   - If it has **MinPts or more** → It's a **core point** → Create a new cluster.
   - If fewer than MinPts → It may be a **border** or **noise**.
3. Add all neighbors of the core point to the same cluster.
4. For each new neighbor:
   - If it's a core point, check *its* neighbors and expand the cluster.
5. Repeat this process until all points are either:

- Assigned to a cluster, or
- Labeled as noise.

| Advantage | Explanation |
|---|---|
| **No need to set number of clusters** | Unlike K-Means, you don't need to say "I want 3 clusters." |
| **Finds clusters of any shape** | Great for odd or curved clusters (like spirals, moons, etc.) |
| **Handles noise/outliers** | Automatically ignores isolated points. |
| **Works well with spatial data** | Excellent when data has location-based patterns. |

## *8. Find the first splitting attribute for the decision tree by using the ID3 algorithm with the following dataset.*



| Age | Competition | Type | Class ( profit) |
|---|---|---|---|
| Old | Yes | Software | Down |
| Old | No | Software | Down |
| Old | No | Hardware | Down |
| Mid | Yes | Software | Down |
| Mid | Yes | Hardware | Down |
| Mid | No | Hardware | Up |
| Mid | No | Software | Up |
| New | Yes | Software | Up |
| New | No | Hardware | Up |
| New | No | Software | Up |

### What are we solving?

You are trying to **build a decision tree** using the **ID3 algorithm**.

- A **Decision Tree** is like a flowchart that helps you make decisions step-by-step based on questions.
- The **ID3 Algorithm** helps you **decide which attribute (column) to split first** by calculating **Information Gain**.
- **Information Gain** tells us:
  ➔ *"Which attribute gives the most 'clarity' (least confusion) when we split the data?"*
  **Our Goal**: Find **which attribute to split first** among **Age**, **Competition**, or **Type**

## What is ID3 Algorithm

1. **Start with all your data.**
2. **Find the best attribute (feature) to split on** — the one that gives the most "clarity" (least confusion).
   - To find this, **ID3 uses something called *Information Gain***.
3. **Split** the data into groups based on that attribute's values.
4. **Repeat** the same steps inside each group:
   - Again find the best attribute to split.
   - Keep splitting until:
     - All the examples in a group are the same class (pure), **or**
     - No more attributes are left.

## Given Data

| Age | Competition | Type | Class (Profit) |
|---|---|---|---|
| Old | Yes | Software | Down |
| Old | No | Software | Down |
| Old | No | Hardware | Down |
| Mid | Yes | Software | Down |
| Mid | Yes | Hardware | Down |
| Mid | No | Hardware | Up |
| Mid | No | Software | Up |
| New | Yes | Software | Up |
| New | No | Hardware | Up |
| New | No | Software | Up |

## Step 1: Find Overall Entropy of the Dataset

(Entropy = how "mixed up" the data is)

## How many "Down" and "Up"?

- "Down" = 5

- "Up" = 5
  (half-half, 50%-50% mixed)

**Entropy formula:**

$$\text{Entropy} = -p(\text{Down})\log_2 p(\text{Down}) - p(\text{Up})\log_2 p(\text{Up})$$

**Where:**

$$p(\text{Down}) = 5/10 = 0.5$$

$$p(\text{Up}) = 5/10 = 0.5$$

**Calculation:**

$$\text{Entropy} = -(0.5)\log_2(0.5) - (0.5)\log_2(0.5)$$

Since $\log_2(0.5) = -1$,

$$\text{Entropy} = -(0.5)(-1) - (0.5)(-1) = 0.5 + 0.5 = 1$$

So **Overall Entropy = 1**.

## Step 2: Find Information Gain for each attribute

(We want to see which attribute gives the biggest "clarity" after split.)

### Attribute 1: Age

**How data is split?**

- **Old** → 3 samples → all **Down**
- **Mid** → 4 samples → 2 Down, 2 Up
- **New** → 3 samples → all **Up**

Let's find Entropy for each group:

- **Old** group: (3 Down)
  ➔ Entropy = 0 (no mix, all same class)

- **Mid** group: (2 Down, 2 Up)
  ➔ Entropy = 1 (perfect mix)
- **New** group: (3 Up)
  ➔ Entropy = 0 (no mix)

**Weighted Average Entropy after split:**

$$\text{Entropy after Age} = \left(\frac{3}{10} \times 0\right) + \left(\frac{4}{10} \times 1\right) + \left(\frac{3}{10} \times 0\right)$$
$$= 0 + 0.4 + 0 = 0.4$$

✅ **Entropy after splitting on Age = 0.4**

**Information Gain for Age:**

$$\text{Gain} = \text{Original Entropy} - \text{New Entropy}$$
$$= 1 - 0.4 = 0.6$$

✅ **Information Gain (Age) = 0.6**

### Attribute 2: Competition

**How data is split?**

- **Yes** → 4 samples (2 Down, 2 Up)
- **No** → 6 samples (3 Down, 3 Up)

Find Entropy:

- **Yes** group: (2 Down, 2 Up) ➔ Entropy = 1
- **No** group: (3 Down, 3 Up) ➔ Entropy = 1

**Weighted Average Entropy:**

$$\text{Entropy after Competition} = \left(\frac{4}{10} \times 1\right) + \left(\frac{6}{10} \times 1\right)$$

$$= 0.4 + 0.6 = 1$$

✅ **Entropy after splitting on Competition = 1**

## Information Gain for Competition:

$$\text{Gain} = 1 - 1 = 0$$

✅ **Information Gain (Competition) = 0**

### Attribute 3: Type

**How data is split?**

- **Software** → 6 samples (3 Down, 3 Up)
- **Hardware** → 4 samples (2 Down, 2 Up)
  Find Entropy:
- **Software** group: (3 Down, 3 Up) ➔ Entropy = 1
- **Hardware** group: (2 Down, 2 Up) ➔ Entropy = 1

**Weighted Average Entropy:**

$$\text{Entropy after Type} = \left(\frac{6}{10} \times 1\right) + \left(\frac{4}{10} \times 1\right)$$

$$= 0.6 + 0.4 = 1$$

✅ **Entropy after splitting on Type = 1**

**Information Gain for Type:**

$$\text{Gain} = 1 - 1 = 0$$

✅ **Information Gain (Type) = 0**

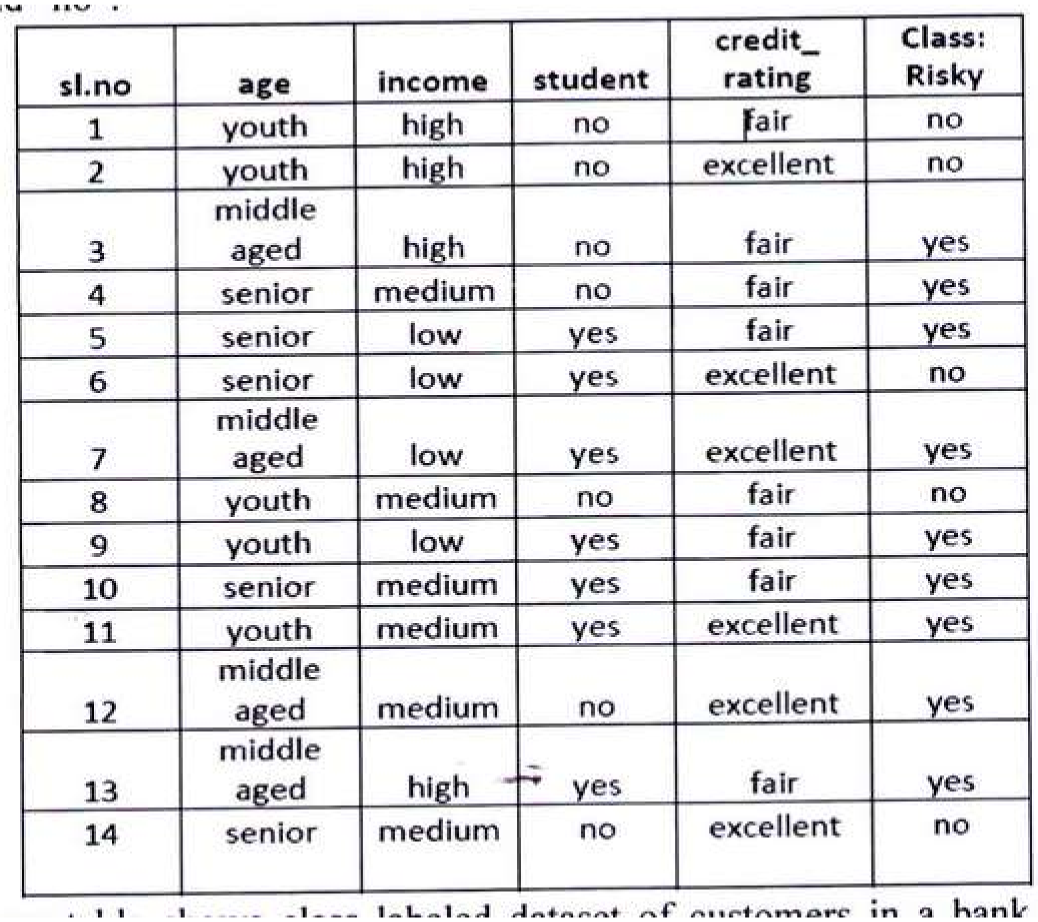
### Final Decision

| Attribute | Information Gain |
|---|---|
| Age | 0.6 |
| Competition | 0 |
| Type | 0 |

- **The highest Information Gain is for *Age*!**
- So, the **first splitting attribute will be**: **Age**.

---

## *9. Consider the following dataset for a binary classification problem with class label 'yes" and 'no'*



| sl.no | age | income | student | credit_ rating | Class: Risky |
|---|---|---|---|---|---|
| 1 | youth | high | no | fair | no |
| 2 | youth | high | no | excellent | no |
| 3 | middle aged | high | no | fair | yes |
| 4 | senior | medium | no | fair | yes |
| 5 | senior | low | yes | fair | yes |
| 6 | senior | low | yes | excellent | no |
| 7 | middle aged | low | yes | excellent | yes |
| 8 | youth | medium | no | fair | no |
| 9 | youth | low | yes | fair | yes |
| 10 | senior | medium | yes | fair | yes |
| 11 | youth | medium | yes | excellent | yes |
| 12 | middle aged | medium | no | excellent | yes |
| 13 | middle aged | high | yes | fair | yes |
| 14 | senior | medium | no | excellent | no |

The above table shows class labeled dataset of customers in a bank. Explain information gain attribute selection measure, and find the information gain of the attribute "age".

| | | | | | |
|---|---|---|---|---|---|
| sl.no | age | income | student | credit rating | Class: Risky |
| 1 | youth | high | no | fair | no |
| 2 | youth | high | no | excellent | no |
| 3 | middle aged | high | no | fair | yes |
| 4 | senior | medium | no | fair | yes |
| 5 | senior | low | yes | fair | yes |
| 6 | senior | low | yes | excellent | no |
| 7 | middle aged | low | yes | excellent | yes |
| 8 | youth | medium | no | fair | no |
| 9 | youth | low | yes | fair | yes |
| 10 | senior | medium | yes | fair | yes |
| 11 | youth | medium | yes | excellent | yes |
| 12 | middle aged | medium | no | excellent | yes |

| | | | | | |
|---|---|---|---|---|---|
| 13 | middle aged | high | yes | fair | yes |
| 14 | senior | medium | no | excellent | no |

## What is information gain

- **Information Gain (IG)** tells us **how much "information" a feature (like "age") gives us about the class** (whether the person is "risky" (yes) or "not risky" (no)).
- In short: **higher IG = better attribute** for splitting the dataset.
- We first calculate **Entropy** (a measure of "impurity" or "disorder") before and after splitting, and **IG = Entropy(before split) - weighted Entropy(after split)**.

## Step 1: Find the Overall Entropy of the dataset

First, find the proportion of `yes` and `no`:

- Count of `yes`: 9 (at sl.no 3, 4, 5, 7, 9, 10, 11, 12, 13)
- Count of `no`: 5 (at sl.no 1, 2, 6, 8, 14)
  Total = 14

Now, Entropy formula:

$$Entropy(S) = -p_{yes}\log_2(p_{yes}) - p_{no}\log_2(p_{no})$$

$$p_{yes} = \frac{9}{14}$$

$$p_{no} = \frac{5}{14}$$

Let's calculate:

$$Entropy(S) = -p_{yes}\log_2(p_{yes}) - p_{no}\log_2(p_{no})$$

where:

- $p_{yes} = 9/14 \approx 0.6429$
- $p_{no} = 5/14 \approx 0.3571$

Now calculate properly:

- $\log_2(0.6429) \approx -0.643$
- $\log_2(0.3571) \approx -1.485$

Now substitute:

$$Entropy(S) = -(0.6429 \times -0.643) - (0.3571 \times -1.485)$$

$$= 0.4136 + 0.5298$$

$$= 0.9434$$

✅ So, **correct Entropy(S) ≈ 0.940**.

## Step 2: Find Entropy after splitting by "age"

"Age" has 3 categories:

- **youth**
- **middle aged**
- **senior**

### (a) Youth

Entries: 1, 2, 8, 9, 11 → 5 records
Class labels:

| sl.no | Class |
|---|---|
| 1 | no |
| 2 | no |
| 8 | no |

| sl.no | Class |
|---|---|
| 9 | yes |
| 11 | yes |

Counts:

- Yes = 2
- No = 3

  Entropy of Youth group:

$$Entropy(youth) = -\left(\frac{2}{5}\right)\log_2\left(\frac{2}{5}\right) - \left(\frac{3}{5}\right)\log_2\left(\frac{3}{5}\right)$$

Using approximate log values:

- $\log_2(2/5) \approx -1.322$
- $\log_2(3/5) \approx -0.737$

Now:

$$Entropy(youth) = -(0.4 \times -1.322) - (0.6 \times -0.737)$$

$$= 0.5288 + 0.4422 = 0.971$$

✅ So:

$$Entropy(youth) \approx 0.971$$

---

## (b) Middle aged

Entries: 3, 7, 12, 13 → 4 records
Class labels:

| sl.no | Class |
|---|---|
| 3 | yes |
| 7 | yes |
| 12 | yes |
| 13 | yes |

All are **yes**.

Entropy when all examples are from one class:

$$Entropy(middle\ aged) = 0$$

✅ So:

$$Entropy(middle\ aged) = 0$$

---

### (c) Senior

Entries: 4, 5, 6, 10, 14 → 5 records
Class labels:

| sl.no | Class |
|---|---|
| 4 | yes |
| 5 | yes |
| 6 | no |
| 10 | yes |
| 14 | no |

Counts:

- Yes = 3
- No = 2

Entropy of Senior group:

$$Entropy(senior) = -\left(\frac{3}{5}\right)\log_2\left(\frac{3}{5}\right) - \left(\frac{2}{5}\right)\log_2\left(\frac{2}{5}\right)$$

(Notice: this is same as Youth, just the yes/no flipped.)

$$Entropy(senior) \approx 0.971$$

✅ So:

$$Entropy(senior) \approx 0.971$$

## Step 3: Find Weighted Entropy after splitting by "age"

Now, combine:

$$Entropy_{age} = \frac{5}{14} \times Entropy(youth) + \frac{4}{14} \times Entropy(middle\ aged) + \frac{5}{14} \times Entropy(senior)$$

Substitute:

$$Entropy_{age} = \frac{5}{14} \times 0.971 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.971$$

$$= 0.346 + 0 + 0.346$$

$$= 0.692$$

## Step 4: Find Information Gain for "age"

$$Information\ Gain(age) = Entropy(S) - Entropy_{age}$$

$$= 0.940 - 0.692$$

$$= 0.248$$

✅ Final Answer:

$$\boxed{Information\ Gain(age) \approx 0.248}$$

## *10. A database contains 80 records on a particular topic of which 55 are relevant to a certain investigation. A search was conducted on that topic and 50 records were retrieved. Of the 50 records retrieved, 40 were relevant. Construct the confusion matrix and calculate the precision and recall scores for the search*

### What was given to us:

- There are **80 records** in total in the database.
- Out of those, **55 are relevant** to our investigation.
- A search was done, and **50 records were retrieved** (search result).
- Out of the 50 retrieved, **40 were relevant**.

Now, let's slowly figure out the rest:

### Step 1: Find irrelevant records (in the whole database)

- Total records = 80
- Relevant records = 55
- So, **irrelevant = total - relevant = 80 - 55 = 25** records.

### Step 2: Find irrelevant records retrieved

- Retrieved records = 50
- Out of 50 retrieved, 40 were relevant.
- So, irrelevant retrieved = 50 - 40 = **10 irrelevant records retrieved**.

### Step 3: Find relevant records not retrieved

- We know 55 records are relevant overall.
- We retrieved 40 relevant ones.
- So, relevant not retrieved = 55 - 40 = **15 relevant records not retrieved**.

### Step 4: Find irrelevant records not retrieved

- There were 25 irrelevant records overall.
- 10 of them were retrieved.
- So, irrelevant not retrieved = 25 - 10 = **15 irrelevant records not retrieved**.

### Given Information:

- **Total records in the database** = 80
- **Relevant records** = 55
- **Irrelevant records** = 80 - 55 = 25
- **Records retrieved** = 50
- **Relevant records retrieved** = 40
- **Irrelevant records retrieved** = 50 - 40 = 10
- **Relevant records not retrieved** = 55 - 40 = 15
- **Irrelevant records not retrieved** = 25 - 10 = 15

### Building the confusion matrix

- **True Positive (TP)** = relevant and retrieved = 40
- **False Positive (FP)** = irrelevant but retrieved = 10
- **False Negative (FN)** = relevant but NOT retrieved = 15
- **True Negative (TN)** = irrelevant and NOT retrieved = 15

| Predicted Relevant | Predicted Irrelevant | |
|---|---|---|
| **Actual Relevant** | **True Positives (TP)** = 40 | **False Negatives (FN)** = 15 |

| Predicted Relevant | Predicted Irrelevant | |
|---|---|---|
| **Actual Irrelevant** | **False Positives (FP)** = 10 | **True Negatives (TN)** = 15 |

## Precision and recall calculation

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{40}{40 + 10} = \frac{40}{50} = 0.8$$

So, the **precision** is **0.8** or **80%**.

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{40}{40 + 15} = \frac{40}{55} \approx 0.727$$

So, the **recall** is approximately **0.727** or **72.7%**.

---

## *11. Explain K-Means Clustering*

### What is K-Means?

- **K-Means** is a **clustering algorithm**.
- It groups data points into **k clusters** based on **distance** to cluster centers (called **centroids**).
- The centroid is **the mean (average)** position of all points in the cluster — **not necessarily an actual data point**.

### Steps in K-Means:

1. **Choose k** (number of clusters).
2. **Randomly select** k points as **initial centroids**.
3. **Assign** each data point to the **nearest centroid**.
4. **Update centroids** by calculating the **mean** of all points in each cluster.
5. **Repeat** steps 3 and 4 until centroids **stop changing** (or change very little).
   ✅ Final clusters are formed when the centroids become stable.

### Example:

Suppose you have these points:

```
Points: (2, 6), (3, 4), (3, 8), (4, 7), (6, 2)
```

- Choose k = 2.
- Pick two random points as centroids, say (2,6) and (6,2).
- Assign points based on who they are closer to.
- Update the centroids (find mean of cluster points).
- Repeat until centroids stabilize.

## *12. Discuss PAM algorithm for clustering with an example.*

- **PAM** is similar to **k-means**, **but** instead of using centroids (average points), it uses **medoids**.
- A **medoid** is an **actual point** from the dataset that **best represents** the cluster (has minimum distance to others in the cluster).
- More **robust to outliers** than k-means because it doesn't create new imaginary points (like k-means centroids do).

### Key Terms:

| Term | Meaning |
|---|---|
| **Medoid** | A real data point with the lowest total distance to other points in the cluster. |
| **Cluster** | Group of points close to a medoid. |
| **Dissimilarity** | Difference between points (using Manhattan or Euclidean distance). |

### PAM Algorithm Steps

#### 1. Build Phase (Initialization)

- Choose **k random points** from the dataset as **medoids**.
- Assign each data point to the **nearest medoid** (based on distance).

### 2. Swap Phase (Optimization)

- Try **swapping** medoids with non-medoids.
- **If swapping reduces** the total distance (cost), accept the swap.
- Repeat until **no swap reduces** the cost.

## Example to Understand PAM

Suppose we have **5 points**:

```
Points: A, B, C, D, E
Coordinates:
A (2, 6)
B (3, 4)
C (3, 8)
D (4, 7)
E (6, 2)
```

And we want to form **k = 2 clusters**.

## Step 1: Build Phase

Randomly select 2 medoids, say A and E.
Now, calculate distances of each point to A and E, and **assign** points to the nearest medoid.
Example distances (using simple Euclidean formula):

| Point | Distance to A | Distance to E | Assigned to |
|---|---|---|---|
| A | 0 (itself) | Far | A |
| B | Close to A | Closer to E | Maybe A or E depending on actual values |
| C | Close to A | Far from E | A |
| D | Close to A | Far from E | A |
| E | 0 (itself) | - | E |

## Step 2: Swap Phase

Now, check if swapping a medoid (say A) with a non-medoid (say B, C, or D) **reduces total distance**

- If **yes**, do the swap and reassign points.
- If **no**, keep the current medoids.

Repeat until **no more swaps help**.

### End Result:

- Points grouped into **2 clusters**, centered around the 2 **best medoids**.
- Medoids are real points (A and E or maybe new ones after swap).

## *13. Explain the working of SLIQ algorithm*

### What is SLIQ?

- **SLIQ** stands for **Supervised Learning In Quest**.
- It is a **decision tree algorithm** made for handling **large datasets**.
- **Special feature**: It **presorts** numeric data before building the tree → making the tree building **much faster**.

### How SLIQ Works — Step by Step:

#### Step 1: Presort Numerical Attributes

- Before starting to build the tree, **SLIQ sorts** all **numerical columns** (like Temperature, Humidity) **only once**.
- Example:
  - Temperature: 64, 65, 68, 70, 72, 80, 83, 85
  - Humidity: 65, 70, 78, 80, 85, 90, 95, 96
- **Why?**
  ➔ This saves time later when searching for the best split.

#### Step 2: Compute Splitting Criteria

- For **categorical attributes** (like Outlook, Wind):
  - Use **Information Gain** or **Gini Index** to decide splits.
- For **numerical attributes** (like Temperature, Humidity):
  - Quickly find the best split using the **presorted list**.

### Step 3: Select the Best Splitting Attribute

- Among all attributes, **choose** the one that gives:
  - Highest **Information Gain** (if using ID3 method)
  - OR
  - Lowest **Gini Index** (if using CART method)
- This attribute becomes the **split at the node**.

### Step 4: Create a Class List

- Instead of copying full data at each node:
  - **SLIQ keeps a "class list"**: it **only tracks class labels** (like Yes, No).

### Step 5: Recursively Grow the Decision Tree

- Keep repeating:
  - Split nodes using the best attribute.
  - Use the class list to guide.
- **Stop** when:
  - All points in a node belong to the same class
  - OR no good split can be found (then assign the majority class).

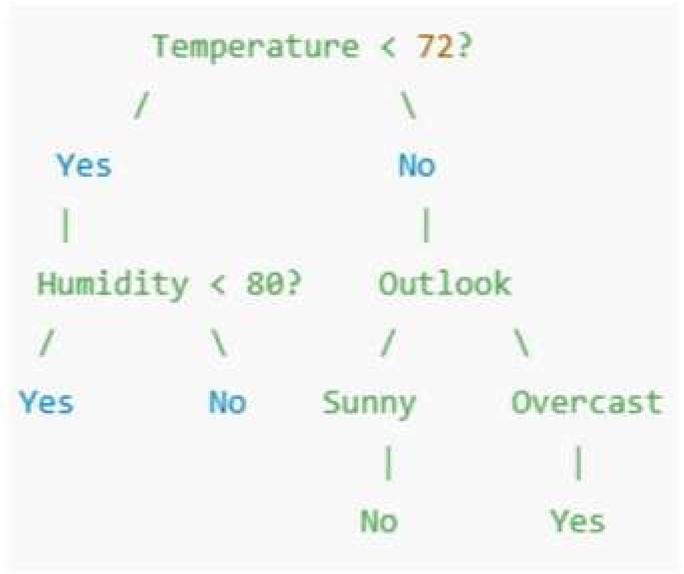
### Step 6: Breadth-First Tree Building

- Unlike ID3 (which build **depth-first**),
  ➔ **SLIQ builds level-by-level** (breadth-first).
- ➔ This makes it easier and faster for **large datasets**.

## Example with "Play Tennis" Dataset

| Outlook | Temperature | Humidity | Wind | Play Tennis? |
|---|---|---|---|---|
| Sunny | 85 | 85 | Weak | No |
| Sunny | 80 | 90 | Strong | No |
| Overcast | 83 | 78 | Weak | Yes |
| Rainy | 70 | 96 | Weak | Yes |
| Rainy | 68 | 80 | Weak | Yes |
| Rainy | 65 | 70 | Strong | No |

| Outlook | Temperature | Humidity | Wind | Play Tennis? |
|---|---|---|---|---|
| Overcast | 64 | 65 | Strong | Yes |
| Sunny | 72 | 95 | Weak | No |

- Step 1: Temperature and Humidity are **presorted**.
- Step 2: Find the best attribute to split (maybe Temperature < 72).
- Step 3: Tree grows level-by-level until all are classified.
- 

```
         Temperature < 72?
        /              \
    Yes                 No
    |                    |
   Humidity < 80?     Outlook
   /        \        /      \
 Yes         No    Sunny     Overcast
                     |          |
                     No        Yes
```

  In **SLIQ** (and in decision trees in general), after we split once (for example, by **Temperature**), we **again apply the splitting logic inside each branch**.
- First split: based on **Temperature** (example: Temperature < 72)
- Now, **inside** the new group (after splitting on Temperature),
  **➔ SLIQ checks again**: "Can we split this further using other attributes?"
  That's how **Humidity < 80** comes **after** Temperature split