Darshan UNIVERSITY
योगः कर्मसु कौशलम्

14 Sep 2025

Data Mining : Lab - 14(A)

In [1]:
```python
import numpy as np
import pandas as pd

# ----------------- Step 1: Initialize -----------------
def initialize_centroids(data, k):
    """Randomly choose k data points as initial centroids"""
    return data.sample(n=k).to_numpy()

# ----------------- Step 2: Assign Clusters -----------------
def assign_clusters(data, centroids):
    """Assign each point to nearest centroid"""
    distances = np.linalg.norm(data.to_numpy()[:, None] - centroids, axis=2)
    return np.argmin(distances, axis=1)

# ----------------- Step 3: Update Centroids -----------------
def update_centroids(data, labels, k):
    """Recompute centroids as mean of assigned points"""
    new_centroids = []
    for i in range(k):
        cluster_points = data.to_numpy()[labels == i]
        if len(cluster_points) > 0:
            new_centroids.append(cluster_points.mean(axis=0))
        else:
            # reinitialize if cluster gets empty
            new_centroids.append(data.sample(n=1).to_numpy()[0])
    return np.array(new_centroids)

# ----------------- Step 4: K-Means Algorithm -----------------
def kmeans(data, k, max_iters=100):
    centroids = initialize_centroids(data, k)

    for _ in range(max_iters):
        labels = assign_clusters(data, centroids)
        new_centroids = update_centroids(data, labels, k)

        # stop if centroids don't change
        if np.allclose(centroids, new_centroids):
```

```
            break
        centroids = new_centroids

    return labels, centroids

df = pd.DataFrame({
    "x": [1, 1.5, 3, 5, 3.5, 4.5, 3.5],
    "y": [1, 2, 4, 7, 5, 5, 4.5]
})

labels, centroids = kmeans(df, k=2)

print("Cluster Labels:", labels)
print("Centroids:\n", centroids)
```

```
Cluster Labels: [1 1 0 0 0 0 0]
Centroids:
 [[3.9  5.1 ]
 [1.25 1.5 ]]
```

Darshan UNIVERSITY
योग: कर्मसु कौशलम्

14 Sep 2025

Data Mining : Lab - 14(B)

In [3]:
```
import numpy as np
import pandas as pd

# ----------------- Step 1: Initialize Medoids -----------------
def initialize_medoids(data, k):
    """Randomly choose k points as initial medoids"""
    return data.sample(n=k).index.to_numpy()

# ----------------- Step 2: Assign Clusters -----------------
def assign_clusters(data, medoids):
    """Assign each point to nearest medoid"""
    distances = np.zeros((len(data), len(medoids)))
    for i, medoid in enumerate(medoids):
        distances[:, i] = np.linalg.norm(data.to_numpy() - data.loc[medoid].
    return np.argmin(distances, axis=1)

# ----------------- Step 3: Update Medoids -----------------
def update_medoids(data, labels, medoids, k):
    """Update medoids by choosing the point in each cluster with minimum tot
    new_medoids = []
    for i in range(k):
        cluster_points = data[labels == i]
        if len(cluster_points) > 0:
            distances = np.sum(np.linalg.norm(cluster_points.to_numpy()[:, N
            new_medoid_index = cluster_points.index[np.argmin(distances)]
            new_medoids.append(new_medoid_index)
        else:
            # if cluster empty, reinitialize
            new_medoids.append(data.sample(n=1).index[0])
    return np.array(new_medoids)

# ----------------- Step 4: K-Medoids Algorithm -----------------
def kmedoids(data, k, max_iters=100):
    medoids = initialize_medoids(data, k)

    for _ in range(max_iters):
        labels = assign_clusters(data, medoids)
```

```
        new_medoids = update_medoids(data, labels, medoids, k)

        if np.array_equal(medoids, new_medoids):
            break
        medoids = new_medoids

    return labels, medoids

df = pd.DataFrame({
    "x": [8,3,4,9,8,5,7,8,7,4],
    "y": [7,7,9,6,5,8,3,4,5,5]
})

labels, medoids = kmedoids(df, k=2)

print("Cluster Labels:", labels)
print("Medoid Indices:", medoids)
print("Medoid Points:\n", df.loc[medoids])
```

```
Cluster Labels: [1 0 0 1 1 0 1 1 1 0]
Medoid Indices: [1 4]
Medoid Points:
    x  y
1  3  7
4  8  5
```