Darshan UNIVERSITY

Data Mining

Implement K- Means without Library

Sample data points

data = [[1, 2], [2, 3], [3, 4], [10, 11], [11, 12], [12, 13], [50, 51], [51, 52], [52, 53]]

In [1]:
```
import math
```

In [9]:
```
data = [
    [1, 2], [2, 3], [3, 4],
    [10, 11], [11, 12], [12, 13],
    [50, 51], [51, 52], [52, 53]
]
```

In [2]:
```
def distance(x1,x2):
    return math.sqrt(((x1[0] - x2[0])**2) + ((x1[1] - x2[1])**2))
```

In [3]:
```
distance([1,1],[1,1])
```

Out[3]:
```
0.0
```

In [4]:
```
def update_cluster_center(cluster_data):
    sum = [0,0]
    for i in cluster_data:
        sum[0] =  sum[0] + i[0]
        sum[1] =  sum[1] + i[1]
    return [sum[0]/len(cluster_data),sum[1]/len(cluster_data)]
```

In [5]:
```
update_cluster_center([[1,1],[2,2],[1,1]])
```

Out[5]:
```
[1.3333333333333333, 1.3333333333333333]
```

Now Implement code

In [7]:
```
import numpy as np

def kmeans_du(k,data):
    # select random center
    center_data = [data[np.random.randint(0,len(data))] for i in range(0,k)]
    print(center_data)

    #cluster data
    cluster_data = [[] for i in range(0,k)]
    for i in range(0,k):
        cluster_data[i].append(center_data[i])
    print(cluster_data)

    for j in range(0,5):
        cluster_data = [[] for i in range(0,k)]
        for d in data:
            mindistance = []
```

```python
            for i in range(0,k):
                mindistance.append(distance(center_data[i],d))
            print(d ,"-->",mindistance)
            cluster_data[mindistance.index(min(mindistance))].append(d)

        # print Cluster data

        for i in range(0,k):
            print(i,"-->",cluster_data[i])

        # update Cluster center
        for i in range(0,k):
            center_data[i] = update_cluster_center(cluster_data[i])
        print("NEW Cluster Center",center_data)
```

In [10]:

```python
kmeans_du(3,data)
```

```
[[1, 2], [11, 12], [10, 11]]
[[[1, 2]], [[11, 12]], [[10, 11]]]
[1, 2] --> [0.0, 14.142135623730951, 12.727922061357855]
[2, 3] --> [1.4142135623730951, 12.727922061357855, 11.313708498984761]
[3, 4] --> [2.8284271247461903, 11.313708498984761, 9.899494936611665]
[10, 11] --> [12.727922061357855, 1.4142135623730951, 0.0]
[11, 12] --> [14.142135623730951, 0.0, 1.4142135623730951]
[12, 13] --> [15.556349186104045, 1.4142135623730951, 2.8284271247461903]
[50, 51] --> [69.29646455628166, 55.154328932550705, 56.568542494923804]
[51, 52] --> [70.71067811865476, 56.568542494923804, 57.982756057296896]
[52, 53] --> [72.12489168102785, 57.982756057296896, 59.39696961966999]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[11, 12], [12, 13], [50, 51], [51, 52], [52, 53]]
2 --> [[10, 11]]
NEW Cluster Center [[2.0, 3.0], [35.2, 36.2], [10.0, 11.0]]
[1, 2] --> [1.4142135623730951, 48.366103833159855, 12.727922061357855]
[2, 3] --> [0.0, 46.95189027078676, 11.313708498984761]
[3, 4] --> [1.4142135623730951, 45.53767670841366, 9.899494936611665]
[10, 11] --> [11.313708498984761, 35.638181771802, 0.0]
[11, 12] --> [12.727922061357855, 34.223968209428904, 1.4142135623730951]
[12, 13] --> [14.142135623730951, 32.80975464705581, 2.8284271247461903]
[50, 51] --> [67.88225099390856, 20.9303607231218, 56.568542494923804]
[51, 52] --> [69.29646455628166, 22.344574285494897, 57.982756057296896]
[52, 53] --> [70.71067811865476, 23.758787847867993, 59.39696961966999]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[50, 51], [51, 52], [52, 53]]
2 --> [[10, 11], [11, 12], [12, 13]]
NEW Cluster Center [[2.0, 3.0], [51.0, 52.0], [11.0, 12.0]]
[1, 2] --> [1.4142135623730951, 70.71067811865476, 14.142135623730951]
[2, 3] --> [0.0, 69.29646455628166, 12.727922061357855]
[3, 4] --> [1.4142135623730951, 67.88225099390856, 11.313708498984761]
[10, 11] --> [11.313708498984761, 57.982756057296896, 1.4142135623730951]
[11, 12] --> [12.727922061357855, 56.568542494923804, 0.0]
[12, 13] --> [14.142135623730951, 55.154328932550705, 1.4142135623730951]
[50, 51] --> [67.88225099390856, 1.4142135623730951, 55.154328932550705]
[51, 52] --> [69.29646455628166, 0.0, 56.568542494923804]
[52, 53] --> [70.71067811865476, 1.4142135623730951, 57.982756057296896]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[50, 51], [51, 52], [52, 53]]
2 --> [[10, 11], [11, 12], [12, 13]]
NEW Cluster Center [[2.0, 3.0], [51.0, 52.0], [11.0, 12.0]]
[1, 2] --> [1.4142135623730951, 70.71067811865476, 14.142135623730951]
[2, 3] --> [0.0, 69.29646455628166, 12.727922061357855]
[3, 4] --> [1.4142135623730951, 67.88225099390856, 11.313708498984761]
[10, 11] --> [11.313708498984761, 57.982756057296896, 1.4142135623730951]
[11, 12] --> [12.727922061357855, 56.568542494923804, 0.0]
[12, 13] --> [14.142135623730951, 55.154328932550705, 1.4142135623730951]
[50, 51] --> [67.88225099390856, 1.4142135623730951, 55.154328932550705]
[51, 52] --> [69.29646455628166, 0.0, 56.568542494923804]
[52, 53] --> [70.71067811865476, 1.4142135623730951, 57.982756057296896]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[50, 51], [51, 52], [52, 53]]
2 --> [[10, 11], [11, 12], [12, 13]]
NEW Cluster Center [[2.0, 3.0], [51.0, 52.0], [11.0, 12.0]]
[1, 2] --> [1.4142135623730951, 70.71067811865476, 14.142135623730951]
[2, 3] --> [0.0, 69.29646455628166, 12.727922061357855]
[3, 4] --> [1.4142135623730951, 67.88225099390856, 11.313708498984761]
[10, 11] --> [11.313708498984761, 57.982756057296896, 1.4142135623730951]
[11, 12] --> [12.727922061357855, 56.568542494923804, 0.0]
[12, 13] --> [14.142135623730951, 55.154328932550705, 1.4142135623730951]
[50, 51] --> [67.88225099390856, 1.4142135623730951, 55.154328932550705]
[51, 52] --> [69.29646455628166, 0.0, 56.568542494923804]
[52, 53] --> [70.71067811865476, 1.4142135623730951, 57.982756057296896]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[50, 51], [51, 52], [52, 53]]
2 --> [[10, 11], [11, 12], [12, 13]]
NEW Cluster Center [[2.0, 3.0], [51.0, 52.0], [11.0, 12.0]]
```

Implement K-Medoids without Library

Sample data points

data = [[1, 2], [2, 3], [3, 4], [10, 11], [11, 12], [12, 13], [50, 51], [51, 52], [52, 53]]

In [17]:
```
import random
import math
```

In [12]:
```
def euclidean_distance(p1, p2):
    return math.sqrt(sum((x - y) ** 2 for x, y in zip(p1, p2)))
```

In [13]:
```python
def assign_points(data, medoids):
    clusters = {i: [] for i in range(len(medoids))}
    for point in data:
        distances = [euclidean_distance(point, medoid) for medoid in medoids]
        nearest = distances.index(min(distances))
        clusters[nearest].append(point)
    return clusters
```

In [14]:
```python
def calculate_cost(clusters, medoids):
    cost = 0
    for i, points in clusters.items():
        for p in points:
            cost += euclidean_distance(p, medoids[i])
    return cost
```

In [15]:
```python
def k_medoids(data, k, max_iter=100):
    # Step 1: Randomly select initial medoids
    medoids = random.sample(data, k)

    for _ in range(max_iter):
        clusters = assign_points(data, medoids)
        current_cost = calculate_cost(clusters, medoids)

        best_medoids = medoids[:]
        improved = False

        # Step 2: Try swapping medoids with non-medoids
        for i in range(len(medoids)):
            for candidate in data:
                if candidate not in medoids:
                    new_medoids = medoids[:]
                    new_medoids[i] = candidate
                    new_clusters = assign_points(data, new_medoids)
                    new_cost = calculate_cost(new_clusters, new_medoids)

                    if new_cost < current_cost:
                        best_medoids = new_medoids
                        current_cost = new_cost
                        improved = True

        medoids = best_medoids
        if not improved:
            break  # convergence

    final_clusters = assign_points(data, medoids)
    return medoids, final_clusters
```

In [18]:
```python
k = 3
medoids, clusters = k_medoids(data, k)
```

In [19]:
```python
print("Final Medoids:", medoids)
print("Clusters:")
for i, points in clusters.items():
    print(f"Cluster {i+1}: {points}")
```

```
Final Medoids: [[51, 52], [11, 12], [2, 3]]
Clusters:
Cluster 1: [[50, 51], [51, 52], [52, 53]]
Cluster 2: [[10, 11], [11, 12], [12, 13]]
Cluster 3: [[1, 2], [2, 3], [3, 4]]
```

In [ ]: