Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2025), B.Sc. in CSE (Day)

Lab Report 04: K-Means Clustering

Course Title: Artificial Intelligence Lab

Course Code: CSE-316 Section:221-14

Student Details

	Name	ID
1.	Md.Mohibullah	221902083

Lab Date : 15-04-2025

Submission Date : 26-04-2025

Course Teacher's Name : Md. Sabbir Hosen Mamun

Lab Report Status

Marks: ……………………………… **Signature:**....................

Comments:.. **Date:**.............................

Lab Report Name: Modified K-Means Clustering Using Manhattan Distance

1. Introduction:

Clustering is an essential technique in unsupervised machine learning that groups similar data points together. The K-Means algorithm is a widely-used clustering method that typically uses Euclidean distance to assign points to the nearest cluster center. In this lab, we implement a modified K-Means algorithm using Manhattan distance instead of Euclidean distance. Furthermore, we visualize the result as a 2D grid using only the print() function, without any graphical libraries.

2. Objective:

- To implement a modified K-Means clustering algorithm in Python.
- To use Manhattan distance instead of the standard Euclidean distance for cluster assignment.
- To generate 100 Cartesian points and 10 initial cluster centers on a 2D grid.
- To visualize the final clusters using a matrix printed in the console using print().

3. Problem Statement:

The standard K-Means clustering algorithm utilizes Euclidean distance to group data points into clusters based on proximity. However, in grid-based environments or applications such as urban pathfinding, Manhattan distance is often a more appropriate metric for measuring distance between points.

In this lab, the task is to modify the traditional K-Means algorithm to use Manhattan distance for cluster assignment. The algorithm must be implemented in Python, where:

- 100 unique Cartesian points and 10 random cluster centers are generated on a 2D grid.
- Each point is assigned to the cluster with the minimum Manhattan distance.
- Cluster centers are updated as the average position of their assigned points until convergence.

Additionally, the final result must be visualized using a 2D matrix printed with the print() function, where:

- Points display their assigned cluster number.
- Cluster centers are shown with distinct capital letters.

This implementation demonstrates the adaptation of K-Means to grid-based domains and provides a textual spatial visualization of clustering results.

4. Procedure:

1. Generate 100 unique random points on a 2D grid (e.g., 10x10).
2. Initialize 10 cluster centers randomly.
3. For each point:
   - Compute the Manhattan distance to all cluster centers.
   - Assign the point to the cluster with the smallest distance.
4. Update each cluster center as the mean position (average x and y) of its assigned points.
5. Repeat steps 3–4 until cluster centers no longer change.
6. Display the grid:
   - Each cell shows either a point (with its cluster number) or a cluster center (with a letter A–J).

5. Implementation:

K-Means Clustering

```
import random
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.cluster = None

def manhattan_distance(p1, p2):
    return abs(p1.x - p2.x) + abs(p1.y - p2.y)

class KMeans:
    def __init__(self, total_points, total_clusters, grid_size=15):
        self.total_points = total_points
        self.total_clusters = total_clusters
        self.grid_size = grid_size

        all_positions = [(x, y) for x in range(grid_size) for y in range(grid_size)]
        random.shuffle(all_positions)
        if total_points > len(all_positions):
            raise ValueError("Grid too small for the number of points!")

        self.points = [Point(x, y) for x, y in all_positions[:total_points]]
        self.clusters = [Point(random.randint(0, grid_size - 1), random.randint(0, grid_size - 1)) for _ in range(total_clusters)]

        self.run_clustering()

    def run_clustering(self):
        while True:
            for p in self.points:
                distances = [manhattan_distance(p, center) for center in self.clusters]
                p.cluster = distances.index(min(distances))
```

```python

            old_centers = [(c.x, c.y) for c in self.clusters]

            for i in range(self.total_clusters):
                cluster_points = [p for p in self.points if p.cluster == i]
                if cluster_points:
                    avg_x = sum(p.x for p in cluster_points) // len(cluster_points)
                    avg_y = sum(p.y for p in cluster_points) // len(cluster_points)
                    self.clusters[i].x = avg_x
                    self.clusters[i].y = avg_y

            new_centers = [(c.x, c.y) for c in self.clusters]
            if new_centers == old_centers:
                break
        self.visualize()

    def visualize(self):
        grid = [["." for _ in range(self.grid_size)] for _ in range(self.grid_size)]

        for p in self.points:
            grid[p.y][p.x] = str(p.cluster)

        for i, center in enumerate(self.clusters):
            grid[center.y][center.x] = chr(65 + i)

        print("\nCluster Visualization (using Manhattan Distance):\n")
        print("     " + "   ".join(f"{i:02}" for i in range(self.grid_size)))
        for row_idx, row in enumerate(grid):
            row_str = "    ".join(row)
            print(f"{row_idx:02}   {row_str}")
def main():
    KMeans(total_points=100, total_clusters=10, grid_size=10)
if __name__ == "__main__":
    main()
```

6. Result:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SPELL CHECKER

PS E:\8th semester\AI Lab\Lab Report 04> & "C:/Program Files/Python313/python.exe" "e:/8th semester/AI Lab/Lab Report 04/K_means_clustering.py"

Cluster Visualization (using Manhattan Distance):

     00   01   02   03   04   05   06   07   08   09
00   5    5    F    5    5    9    9    J    9    9
01   3    3    5    5    0    0    0    9    9    9
02   3    D    3    3    0    0    A    0    0    0
03   3    3    3    4    4    0    0    0    6    6
04   8    3    4    E    4    4    0    6    6    6
05   I    8    4    4    H    7    6    6    G    6
06   8    2    2    4    7    7    1    6    6    6
07   2    C    2    2    7    1    1    1    6    6
08   2    2    2    1    1    1    B    1    1    1
09   2    2    2    1    1    1    1    1    1    1
PS E:\8th semester\AI Lab\Lab Report 04>
```

1. 100 data points were successfully assigned to 10 clusters using Manhattan distance.

2. The cluster centers were iteratively updated until convergence.

3. A matrix visualization was printed in the console:

- Numbers (0–9) represent point clusters.
- Letters (A–J) represent cluster centers.

4. The final output is a clear, structured grid showing the spatial distribution of clusters.

7. Conclusion

In this lab, we successfully implemented a modified version of the K-Means clustering algorithm using Manhattan distance. Unlike the traditional method, this approach is more suitable for grid-like data structures or urban planning problems. The cluster formation and convergence behavior were validated through console-based matrix visualization, offering a lightweight and intuitive understanding of the clustering process.

GitHub Link: https://github.com/programmermahi/Artificial-Intelligence/tree/main/LabReport04-K-Means%20Clustering