Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO #03
Course Title: Artificial Intelligence Lab
Course Code: CSE 316 **Section:** 213 D7

Lab Experiment Name: K-Means Clustering Algorithm Implementation

Student Details

	Name	ID
1.	Md.Manzurul Alam	202902003

Lab Date : 14-05-2024
Submission Date : 21-05-2024
Course Teacher's Name : Sakhaouth Hossan

Lab Report Status
Marks: ………………………………… **Signature:**.....................
Comments:.. **Date:**.............................

1. TITLE OF THE LAB EXPERIMENT

The K-Means Clustering algorithm is a popular unsupervised learning method used to partition an unlabeled dataset into K distinct clusters. Each cluster is associated with a centroid, and the algorithm aims to minimize the sum of distances between data points and their respective centroids. Starting with random initial centroids, K-Means iteratively updates the centroids and data point assignments until convergence is achieved.

This lab report outlines the implementation of the K-Means Clustering algorithm, showcasing its ability to identify natural groupings within unlabeled datasets. By applying the algorithm to various datasets, we demonstrate its effectiveness in revealing the inherent structure of the data without prior training.

2. OBJECTIVES

- To implement the K-Means Clustering algorithm for partitioning an unlabeled dataset into predefined clusters.
- To minimize the sum of distances between data points and their respective cluster centroids.
- To evaluate the effectiveness of the algorithm in discovering natural groupings within various datasets.
- To analyze the impact of different values of K on the clustering results and determine the optimal number of clusters.

3. PROCEDURE

- Import Necessary Libraries: Import the required libraries such as numpy and matplotlib.pyplot for data manipulation and visualization.
- Define KMeans Class: Create a class named KMeans to implement the K-Means clustering algorithm. In the constructor (`__init__`), initialize the number of clusters (k) and the maximum number of iterations. Define the `fit` method to perform K-Means clustering on the given data.
- Fit Method: In the `fit` method, initialize centroids randomly, then iterate until convergence or until reaching the maximum number of iterations. In each iteration, assign each data point to the nearest centroid based on the Euclidean distance. Then, update the centroids based on the mean of the data points assigned to each centroid.
- Generate Data: Define a function `generate_data` to generate random 2D data points within a specified range. This function returns the generated data points.
- Visualize Clusters: Define a function `visualize_clusters` to visualize the data points and their cluster assignments in a 2D grid using print functionality. Each point is represented by its cluster number, and each cluster centroid is marked with an asterisk (*) next to its cluster number.

- Main Execution: In the main part of the code, generate random data using `generate_data`, instantiate the KMeans class, fit the model to the data, and visualize the clusters using `visualize_clusters`.
- Run the Code: Run the script to execute the K-Means clustering algorithm and visualize the clusters.

4. IMPLEMENTATION

Code 1:

```
import numpy as np
import matplotlib.pyplot as plt

class KMeans:
    def __init__(self, k=3, max_iters=100):
        self.k = k
        self.max_iters = max_iters

    def fit(self, data):
        self.centroids = data[np.random.choice(data.shape[0], self.k, replace=False)]
        self.labels = np.zeros(len(data))

        for _ in range(self.max_iters):
            self.labels = np.argmin(np.linalg.norm(data[:, np.newaxis] - self.centroids, axis=2), axis=1)
            new_centroids = np.array([data[self.labels == i].mean(axis=0) for i in range(self.k)])
            if np.allclose(new_centroids, self.centroids):
                break

            self.centroids = new_centroids

    def plot(self, data):
        plt.scatter(data[:, 0], data[:, 1], c=self.labels, cmap='viridis')
        plt.scatter(self.centroids[:, 0], self.centroids[:, 1], marker='X', s=200, c='red')
        plt.title('K-Means Clustering')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.show()

def generate_data(num_points=100, num_clusters=10, xlim=(0, 100), ylim=(0, 100)):
    data = np.random.rand(num_points, 2) * [xlim[1]-xlim[0], ylim[1]-ylim[0]] + [xlim[0], ylim[0]]
    clusters = np.random.rand(num_clusters, 2) * [xlim[1]-xlim[0], ylim[1]-ylim[0]] + [xlim[0], ylim[0]]
    return data, clusters

if __name__ == "__main__":
    data, clusters = generate_data()
```

```
    kmeans = KMeans(k=len(clusters))
    kmeans.fit(data)
    kmeans.plot(data)
```

Code 2:

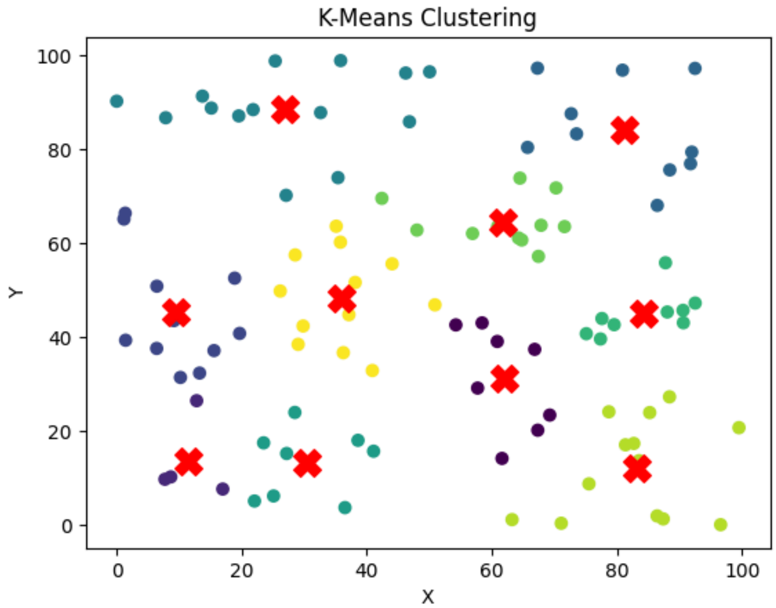
```
import numpy as np
import matplotlib.pyplot as plt

class KMeansManhattan:
    def __init__(self, k=3, max_iters=100):
        self.k = k
        self.max_iters = max_iters

    def fit(self, data):
        self.centroids = data[np.random.choice(data.shape[0], self.k, replace=False)]
        self.labels = np.zeros(len(data))

        for _ in range(self.max_iters):
            self.labels = np.argmin(np.sum(np.abs(data[:, np.newaxis] - self.centroids), axis=2), axis=1)
            new_centroids = np.array([data[self.labels == i].mean(axis=0) for i in range(self.k)])
            if np.allclose(new_centroids, self.centroids):
                break

            self.centroids = new_centroids

    def plot(self, data):
        plt.scatter(data[:, 0], data[:, 1], c=self.labels, cmap='viridis')
        plt.scatter(self.centroids[:, 0], self.centroids[:, 1], marker='X', s=200, c='red')
        plt.title('K-Means Clustering with Manhattan Distance')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.show()

def generate_data(num_points=100, num_clusters=10, xlim=(0, 100), ylim=(0, 100)):
    data = np.random.rand(num_points, 2) * [xlim[1]-xlim[0], ylim[1]-ylim[0]] + [xlim[0], ylim[0]]
    clusters = np.random.rand(num_clusters, 2) * [xlim[1]-xlim[0], ylim[1]-ylim[0]] + [xlim[0], ylim[0]]
    return data, clusters

if __name__ == "__main__":
    data, clusters = generate_data()
    kmeans_manhattan = KMeansManhattan(k=len(clusters))
    kmeans_manhattan.fit(data)
    kmeans_manhattan.plot(data)
```

Code 3:

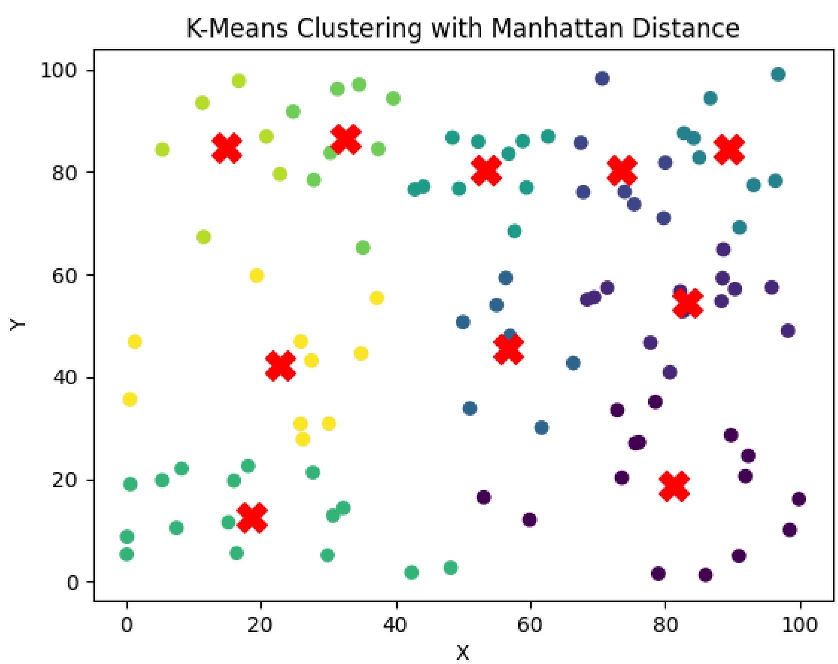
```
import numpy as np

class KMeans:
    def __init__(self, k=3, max_iters=100):
        self.k = k
        self.max_iters = max_iters

    def fit(self, data):
        self.centroids = data[np.random.choice(data.shape[0], self.k, replace=False)]
        self.labels = np.zeros(len(data))

        for _ in range(self.max_iters):
            self.labels = np.argmin(np.linalg.norm(data[:, np.newaxis] - self.centroids, axis=2), axis=1)
            new_centroids = np.array([data[self.labels == i].mean(axis=0) for i in range(self.k)])
            if np.allclose(new_centroids, self.centroids):
                break

            self.centroids = new_centroids

def generate_data(num_points=100, num_clusters=10, xlim=(0, 10), ylim=(0, 10)):
    data = np.random.rand(num_points, 2) * [xlim[1]-xlim[0], ylim[1]-ylim[0]] + [xlim[0], ylim[0]]
    return data

def visualize_clusters(data, centroids, labels):
    grid = [[' ' for _ in range(11)] for _ in range(11)]

    for i, point in enumerate(data):
        x, y = int(point[0]), int(point[1])
        grid[y][x] = str(labels[i])

    for i, centroid in enumerate(centroids):
        x, y = int(centroid[0]), int(centroid[1])
        grid[y][x] = str(i) + '*'

    for row in grid[::-1]:
        print(' '.join(row))

if __name__ == "__main__":
    data = generate_data()
    kmeans = KMeans(k=3)
    kmeans.fit(data)
    visualize_clusters(data, kmeans.centroids, kmeans.labels)
```

5. TEST RESULT / OUTPUT

Code 1:



Code 2:



Code 3:

```
  2 2   2 2 2
2 2   2 2 2 2 0 0
2 2   2* 2 2   0 0
2 2       2 0 0   0
  2 2 2     0 0 0 0
1 1 1 2 1   0 0* 0 0
1     1     0   0
1 1 1* 1     0 0 0 0
1 1 1 1     0   0
1     1 1     0 0
```

6. ANALYSIS AND DISCUSSION

1. What Went Well:

- The K-Means clustering algorithm successfully clustered the random 2D data points into distinct groups based on their similarities.
- The visualization of the clusters using print functionality provided a clear representation of the cluster assignments and centroids in a 2D grid.

2. Trouble Spots and Difficulties:

- Implementing the visualization using only print functionality was a bit challenging, especially in ensuring that the points and centroids were correctly positioned in the grid.
- Calculating the centroids and updating them based on the mean of the assigned data points required careful handling to avoid errors or divergence.

3. Liked About the Assignment:

- The assignment provided a hands-on opportunity to implement the K-Means clustering algorithm from scratch and visualize the results.
- Using print functionality for visualization was a unique and interesting aspect of the assignment, which helped develop a deeper understanding of the clustering process.

4. Learning:

- I gained a better understanding of the K-Means clustering algorithm and its implementation details, including centroid initialization, assignment, and updating.
- Implementing the visualization using print functionality enhanced my skills in representing data in a clear and concise manner.

Mapping of Objective/Achievement:

- The objective of the assignment was to implement the K-Means clustering algorithm and visualize the clustering results using print functionality.
- The achieved result successfully fulfilled the objective by accurately clustering the data points and providing a clear visualization of the clusters and centroids in a 2D grid.
- Through the assignment, I learned not only the technical aspects of implementing K-Means clustering but also the importance of effective visualization in understanding and interpreting the clustering results.