K-means : TD + TP

Dr. EL BENANY Mohamed Mahmoud

January 23, 2025

1 Introduction

Dans ce TD, nous expliquons le code implémentant l'algorithme K-Means avec différentes distances, notamment la distance Euclidienne, la distance de Manhattan et la distance de Minkowski. Nous détaillons également chaque fonction du code, ainsi que les calculs impliqués.

2 Fonctions de Distance

2.1 Distance Euclidienne

La **distance Euclidienne** entre deux points dans un espace à deux dimensions est calculée comme la racine carrée de la somme des carrés des différences des coordonnées respectives des points. La formule mathématique est la suivante :

$$D_{\text{Euclidienne}} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Voici la fonction correspondante en Python :

```
def euclidean_distance(point1, point2):
    return math.sqrt((point2[0] - point1[0])**2 + (point2[1] -
        point1[1])**2)
```

2.2 Distance de Manhattan

La **distance de Manhattan** est la somme des différences absolues des coordonnées des deux points. Cette distance est souvent utilisée dans des situations où le mouvement est contraint à un réseau de type grille (comme à Manhattan, d'où son nom).

La formule mathématique est la suivante :

$$D_{\text{Manhattan}} = |x_2 - x_1| + |y_2 - y_1|$$

Voici la fonction correspondante en Python :

```
def manhattan_distance(point1, point2):
    return abs(point2[0] - point1[0]) + abs(point2[1] - point1[1])
```

2.3 Distance de Minkowski

La **distance de Minkowski** est une généralisation des distances Euclidienne et Manhattan. Elle est paramétrée par un exposant p, où pour $p = 2$ cela devient la distance Euclidienne et pour $p = 1$, elle devient la distance de Manhattan. La formule est la suivante :

$$D_{\text{Minkowski}} = (|x_2 - x_1|^p + |y_2 - y_1|^p)^{1/p}$$

Voici la fonction correspondante en Python :

```
def minkowski_distance(point1, point2, p=2):
    return (abs(point2[0] - point1[0])**p + abs(point2[1] - point1
        [1])**p)**(1/p)
```

3 Fonction d'Assignation des Clusters

La fonction suivante attribue chaque point de données au cluster le plus proche, en utilisant la fonction de distance fournie (par exemple, la distance de Manhattan).

```
def assign_clusters(data, centroids, distance_function):
    clusters = []
    for _, row in data.iterrows():
        distances = {key: distance_function((row['X'], row['Y']),
            value) for key, value in centroids.items()}
        cluster = min(distances, key=distances.get)
        clusters.append(cluster)
    data['Cluster'] = clusters
    return data
```

3.1 Paramètres

- `data` : DataFrame contenant les points de données avec les colonnes `X` et `Y`.
- `centroids` : Dictionnaire des centroïdes initiaux de chaque cluster.
- `distance_function` : Fonction de distance à utiliser pour calculer la proximité entre les points et les centroïdes.

3.2 Calcul

La fonction calcule la distance entre chaque point et chaque centroïde, puis attribue le point au cluster du centroïde le plus proche.

4 Fonction de Mise à Jour des Centroïdes

La fonction suivante met à jour les centroïdes de chaque cluster en fonction des points qui leur sont attribués. Le nouveau centroïde est calculé comme la moyenne des points du cluster.

```
def update_centroids(data):
    new_centroids = {}
    for cluster in data['Cluster'].unique():
        cluster_points = data[data['Cluster'] == cluster]
        new_x = cluster_points['X'].mean()
        new_y = cluster_points['Y'].mean()
        new_centroids[cluster] = (new_x, new_y)
    return new_centroids
```

4.1 Paramètres

- `data` : DataFrame contenant les points de données et leurs clusters.

4.2 Calcul

Pour chaque cluster, cette fonction extrait les points assignés à ce cluster et calcule la moyenne des coordonnées X et Y pour obtenir le nouveau centroïde.

5 Algorithme K-Means

L'algorithme K-Means est itératif. Il attribue les points aux clusters, met à jour les centroïdes et répète ces étapes jusqu'à ce que la convergence soit atteinte. La convergence est déterminée par l'absence de changements significatifs dans les centroïdes entre deux itérations successives.

```
def kmeans(data, initial_centroids, max_iterations=10, tolerance=1e
    -4):
    centroids = initial_centroids
    for i in range(max_iterations):
        print(f"\nIt ration␣{i␣+␣1}")
        data = assign_clusters(data, centroids, manhattan_distance)
              # Exemple de distance
        new_centroids = update_centroids(data)

        # V rifier la convergence
        convergence = all(
            euclidean_distance(centroids[key], new_centroids[key])
                <= tolerance
            for key in centroids.keys()
        )

        if convergence:
            print("Convergence␣atteinte.␣Les␣centro des␣ne␣
                changent␣plus.")
            break
```

```
        centroids = new_centroids
        print(f"Nouveaux centro des : {centroids}")

    return data, centroids
```

5.1 Paramètres

- `data` : DataFrame contenant les points de données à regrouper.
- `initial_centroids` : Dictionnaire des centroïdes initiaux.
- `max_iterations` : Nombre maximum d'itérations (par défaut 10).
- `tolerance` : Tolérance pour vérifier la convergence des centroïdes (par défaut $1e-4$).

5.2 Calcul

À chaque itération, les points sont réassignés aux clusters, les centroïdes sont mis à jour, et la convergence est vérifiée. L'algorithme s'arrête lorsque les centroïdes ne changent plus ou lorsque le nombre maximal d'itérations est atteint.

6 A retenir

La méthode de **K-Means** utilise différentes distances pour attribuer les points aux clusters et mettre à jour les centroïdes.

7 TD: Algorithme K-Means

7.1 Données Initiales

Les points considérés sont :

$$A_1(2,10),\ A_2(2,5),\ A_3(8,4),\ B_1(5,8),\ B_2(7,5),\ B_3(6,4),\ C_1(1,2),\ C_2(4,9).$$

Les centroïdes initiaux sont :

$$\mu_1 = (2,10),\ \mu_2 = (5,8),\ \mu_3 = (1,2).$$

7.2 Itération 1

Étape d'affectation

Les distances entre chaque point et les centroïdes sont calculées comme suit :

$$d((x_1,y_1),(x_2,y_2)) = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}.$$

Clusters après affectation :

$$C_1 = \{A_1\},\quad C_2 = \{A_3, B_1, B_2, B_3, C_2\},\quad C_3 = \{A_2, C_1\}.$$

Point	$d(\mu_1)$	$d(\mu_2)$	$d(\mu_3)$	Affectation
$A_1(2,10)$	0	3.61	8.06	C_1
$A_2(2,5)$	5.00	4.24	3.16	C_3
$A_3(8,4)$	7.21	4.47	7.07	C_2
$B_1(5,8)$	3.61	0	7.21	C_2
$B_2(7,5)$	5.83	3.16	6.08	C_2
$B_3(6,4)$	7.21	4.24	5.00	C_2
$C_1(1,2)$	8.06	7.21	0	C_3
$C_2(4,9)$	2.24	1.41	7.07	C_2

Table 1: Étape d'affectation - Itération 1

7.2.1 Étape de mise à jour

Les nouveaux centroïdes sont calculés comme la moyenne des points assignés à chaque cluster :

$$\mu_1 = \frac{1}{|C_1|} \sum_{x \in C_1} x = (2, 10).$$

$$\mu_2 = \frac{1}{|C_2|} \sum_{x \in C_2} x = \frac{1}{5} \left((8,4) + (5,8) + (7,5) + (6,4) + (4,9) \right) = (6, 6).$$

$$\mu_3 = \frac{1}{|C_3|} \sum_{x \in C_3} x = \frac{1}{2} \left((2,5) + (1,2) \right) = (1.5, 3.5).$$

Nouveaux centroïdes :

$$\mu_1 = (2, 10), \quad \mu_2 = (6, 6), \quad \mu_3 = (1.5, 3.5).$$

7.3 Itération 2

Étape d'affectation

Recalculons les distances avec les nouveaux centroïdes :

Clusters après affectation :

$$C_1 = \{A_1, C_2\}, \quad C_2 = \{A_3, B_1, B_2, B_3\}, \quad C_3 = \{A_2, C_1\}.$$

7.3.1 Étape de mise à jour

Les nouveaux centroïdes sont :

$$\mu_1 = \frac{1}{|C_1|} \sum_{x \in C_1} x = \frac{1}{2} \left((2,10) + (4,9) \right) = (3, 9.5).$$

Point	$d(\mu_1)$	$d(\mu_2)$	$d(\mu_3)$	Affectation
$A_1(2,10)$	0	5.66	6.52	C_1
$A_2(2,5)$	5.00	4.47	1.80	C_3
$A_3(8,4)$	7.21	2.83	6.52	C_2
$B_1(5,8)$	3.61	2.24	6.80	C_2
$B_2(7,5)$	5.83	1.41	5.70	C_2
$B_3(6,4)$	7.21	2.00	4.61	C_2
$C_1(1,2)$	8.06	6.40	1.12	C_3
$C_2(4,9)$	2.24	3.16	6.80	C_1

Table 2: Étape d'affectation - Itération 2

$$\mu_2 = \frac{1}{|C_2|} \sum_{x \in C_2} x = \frac{1}{4}\left((8,4) + (5,8) + (7,5) + (6,4)\right) = (6.5, 5.25).$$

$$\mu_3 = \frac{1}{|C_3|} \sum_{x \in C_3} x = \frac{1}{2}\left((2,5) + (1,2)\right) = (1.5, 3.5).$$

Nouveaux centroïdes :

$$\mu_1 = (3, 9.5), \quad \mu_2 = (6.5, 5.25), \quad \mu_3 = (1.5, 3.5).$$

7.4 Itération 3

Étape d'affectation

Point	$d(\mu_1)$	$d(\mu_2)$	$d(\mu_3)$	Affectation
$A_1(2,10)$	1.12	5.89	6.52	C_1
$A_2(2,5)$	4.58	4.61	1.80	C_3
$A_3(8,4)$	7.21	2.02	6.52	C_2
$B_1(5,8)$	2.50	2.02	6.80	C_2
$B_2(7,5)$	5.70	0.90	5.70	C_2
$B_3(6,4)$	6.40	1.25	4.61	C_2
$C_1(1,2)$	8.06	6.40	1.12	C_3
$C_2(4,9)$	1.12	3.60	6.80	C_1

Table 3: Étape d'affectation - Itération 3

Clusters après affectation :

$$C_1 = \{A_1, C_2\}, \quad C_2 = \{A_3, B_1, B_2, B_3\}, \quad C_3 = \{A_2, C_1\}.$$

7.4.1 Étape de mise à jour

Nouveaux centroïdes :

$$\mu_1 = (3, 9.5), \quad \mu_2 = (6.5, 5.25), \quad \mu_3 = (1.5, 3.5).$$

Les centroïdes ne changent pas à cette étape. L'algorithme a convergé.

7.5 Convergence

L'algorithme a convergé à la quatrième itération. Les clusters finaux sont :

$$C_1 = \{A_1, C_2\}, \quad C_2 = \{A_3, B_1, B_2, B_3\}, \quad C_3 = \{A_2, C_1\}.$$

Les centroïdes finaux sont :

$$\mu_1 = (3, 9.5), \quad \mu_2 = (6.5, 5.25), \quad \mu_3 = (1.5, 3.5).$$