Algorithme des k-moyennes

September 27, 2026

Le partitionnement en k-moyennes est une méthode de partitionnement d'objets en k *clusters* (groupe en français) de façon à faire des regroupements d'objets *qui se ressemblent* ou qui potentiellement appartiennent à la même catégorie.

On suppose que chaque objet est caractérisé par un vecteur de R^d comme dans l'algorithme des k plus proches voisins. On a un certain nombre n d'objets que l'on veut répartir en k clusters. Dans l'exemple qui suit $d = 2$, $n = 12$ et on prendra $k = 3$.

Etant donné une répartition des objets en clusters $S_1, S_2, \dots S_k$, on considère les points moyens $\mu_1, \mu_2, \dots, \mu_k$ de ces clusters et on s'intéresse à la quantité, appelée parfois *fonction de coût*:

$$C = \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

où $\|\cdot\|$ désigne la norme euclidienne de R^d.

L'objectif est de trouver le partionnement qui rende cette quantité C minimale. Malheureusement, lorsque le nombre de données est trop grand, il n'est pas possible de chercher tous les partionnements pour trouver le meilleur. Le temps de calcul serait trop long!

Exercice 1 Donner l'ordre de grandeur du nombre de partionnements de n objets en 3 clusters numérotés 0, 1, 2.

On obtient en général un partionnement *acceptable* en procédant de la façon suivante.

On commence par choisir un ensemble E_0 de k points $\mu_1, \mu_2, \dots, \mu_k$ pris, en général, au hasard (ou arbitrairement) dans le jeu de données et on fait un premier partionnement en mettant dans le cluster i, les données les plus proches de μ_i pour tout i compris entre 1 et k. On calcule les points moyens de chaque cluster, ce qui donne un ensemble de k nouveaux points donnant lieu à un nouveau partionnement du jeu de données en k clusters avec une fonction coût inférieur. A chaque itération du processus, la fonction coût diminue. Donc, après un certain d'itérations, on peut espérer avoir un bon partionnement du jeu de données en k clusters.

Comme mentionné plus haut, cette algorithme ne fournit pas obligatoirement le meilleur partionnement en k clusters mais il donne souvent de bons résultats. Le résultat peut d'ailleurs dépendre du choix initial des points $\mu_1, \dots, \mu_k$.

On dispose de 12 objets que l'on veut répartir en trois catégories.

[1]:
```
Objets=[[0,0],[1,1],[3,4],[3,6],[4,2],[10,6],
        [0,2],[10,7],[8,4],[6,8],[1,0],[9,6]]
```

On dit qu'une liste d'entiers définit une partition de la liste *Objets* en k clusters lorsqu'elle est de même longueur que la liste *Objets* et contient des entiers compris entre 0 à~$k-1$.

Par exemple, la liste $(0,1,0,0,1,2,2,2,1,1,0,2)$ définit une partition de *Objets* en trois clusters.

Le cluster 0 contient les objets de coordonnées $(0,0)$, $(3,4)$, $(3,6)$, $(1,0)$.

Exercice 2

Ecrire une fonction `clusters` qui prend en argument une liste d'objets, une partition de cette liste et un numéro de cluster. Cette fonction renvoie la liste des objets du cluster et les coordonnées du point moyen de ce cluster.

[6]:
```
def clusters(Objets,partition,i):
    ...
    return L,[xm,ym]
```

[7]:
```
clusters(Objets,[0,1,0,0,1,2,2,2,1,1,0,2],0)
```

[7]:
```
([[0, 0], [3, 4], [3, 6], [1, 0]], [1.75, 2.5])
```

Exercice 3

Ecrire une fonction `affiche_clusters` qui prend en argument des objets et une partition, et qui affiche les objets d'un même cluster de la même couleur.

[10]:
```
import matplotlib.pyplot as plt
Couleurs=['blue','green','red','yellow']

def affiche_clusters(Objets,partition):
    for k in range(len(Objets)):
        plt.plot(...)
```

[ ]:
```
affiche_cluster(Objets,[0,1,0,0,1,2,2,2,1,1,0,2])
plt.show()
```

Exercice 4

Ecrire une fonction `cout` qui prend en argument des objets et une partition, et qui calcule le coût de la partition.

[12]:
```
def dist2(A,B):
    s=0
    for i in range(len(A)):
        s+=(A[i]-B[i])**2
    return s

def cout(Objets,partition):
    C=0
    k=max(partition)+1
    ...
    return C
```

[13]:
```
cout(Objets,[0,1,0,0,1,2,2,2,1,1,0,2])
```

[13]: 174.75

[14]:
```
cout(Objets,[0, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0])
```

[14]: 196.5

A chaque cluster d'une partition, on associe un point. Si la partition est composée de k clusters, on dispose d'une liste C de k points (représentés par une liste de deux réels) et on associe le point $C[0]$ au cluster 0, le point $C[1]$ au cluster 1, etc.

Les points $C[0], \dots, C[k-1]$ doivent être distincts.

Dans l'exercice ci-dessous, on pourra prendre la liste des points $(0,0)$, $(0,1)$, $(0,2)$.

Exercice 5

Ecrire une fonction `affichage` qui prend en argument une liste d'objets et une partition de cette liste, et qui relie chaque point d'un même cluster à son point moyen.

[20]:
```
import matplotlib.pyplot as plt
# plt.plot([x0,x1],[y0,y1]) pour tracer le segment dont les extrémités
# sont les points de coordonnées (x0,y0) et (x1,y1)

def affichage(Objets, partition):
    k=max(partition)+1
    ...
```

[ ]:
```
affichage(Objets,[0,1,0,0,1,2,2,2,1,1,0,2])
plt.show()
```

Exercice 6

Ecrire une fonction `repartition` qui prend en argument deux listes de points `L1` et `L2` qui cherche pour chaque point de `L1` le numéro du point de `L2` le plus proche.

La fonction renvoie la liste `D` de ces numéros.

Par exemple si `D[2]` vaut 0 cela signifie que le point numéro 0 de `L2` est le point de `L2` le plus proche du point numéro 2 de `L1`.

[22]:
```
def repartition(L1,L2):
    n=len(L1)
    D=[]
    for i in range(n):
        ....
    return D
```

[23]:
```
repartition(Objets,[Objets[3],Objets[1],Objets[4]])
```

[23]: [1, 1, 0, 0, 2, 0, 1, 0, 2, 0, 1, 0]

Exercice 7

Ecrire une fonction `kmoyennes` qui prend en paramètre une liste d'objet, une partition de cette liste et un entier $N > 0$, et qui réalise N itérations de l'algorithme des k moyennes.

Pour les partitions proposées dans les applications numériques, on supposera qu'aucun cluster ne devient vide.

[ ]:
```
def kmoyennes(Objets,partition,N):
    n=max(partition)+1
    for l in range(N):
        ....
    affichage(Objets,partition)
```

[ ]:
```
kmoyennes(Objets,[0, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0],6)
plt.show()
```

Exercice 8

Afficher l'évolution de la fonction coût pour mettre en évidence qu'elle diminue à chaque itération de l'algorithme des k moyennes et qu'elle converge.

[40]:
```
def kmoyennes2(Objets,partition,N):
    Couts=[cout(Objets,partition)]
    n=max(partition)+1
    for l in range(N):
        ...
    return Couts
```

[ ]:
```
L=kmoyennes2(Objets,[0, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0],9)
plt.plot(L)
```