

# TP $k$ -Moyennes

M. Tami, T. Thonet,  
E. Gaussier

L'objectif de ce TP est d'étudier l'algorithme des  $k$ -Moyennes qui est une approche standard de partitionnement des données (*clustering*).

```
Entrée      :  
    — la précision  $\epsilon$  pour le critère d'arrêt  
    — le nombre de clusters à obtenir  $k$   
    — l'ensemble des données  $X = (x_1, \dots, x_n)$  avec  $n$  le nombre total de données  
Initialisation :  
    — Initialiser les centroïdes  $C^{(0)} = (c_1^{(0)}, \dots, c_k^{(0)})$  par  $k$  données choisies aléatoirement dans  $X$   
    —  $l \leftarrow 0$   
répéter  
    # Mise-à-jour de l'appartenance de chaque donnée à un cluster  
    pour  $i = 1, \dots, n$  faire  
         $a_i^{(l)} = \operatorname{argmin}_{1 \leq j \leq k} \|x_i - c_j^{(l)}\|$       # Trouver le centroïde le plus proche de  $x_i$   
    fin  
    # Mise-à-jour des centroïdes par la moyenne des données des clusters  
    pour  $j = 1, \dots, k$  faire  
         $S_j = \{i \mid 1 \leq i \leq n \text{ et } a_i^{(l)} = j\}$       # Données assignées au cluster  $j$   
         $c_j^{(l+1)} = \frac{1}{|S_j|} \sum_{i \in S_j} x_i$       # Centroïde = moyenne de ces données  
    fin  
     $l \leftarrow l + 1$   
jusqu'à  $\frac{1}{n} \sum_{i=1}^n |a_i^{(l+1)} - a_i^{(l)}| \leq \epsilon$       # Stabilité de l'appartenance aux clusters  
Sortie      : appartenances  $A^{(l)} = (a_1^{(l)}, \dots, a_n^{(l)})$ , centroïdes  $C^{(l)} = (c_1^{(l)}, \dots, c_k^{(l)})$ 
```

**Algorithme 1** : Algorithme des  $k$ -Moyennes

**Algorithme des  $k$ -Moyennes.** Implémenter en Python l'algorithme des  $k$ -Moyennes décrit dans l'Algorithme 1. Tester le programme avec différentes valeurs de  $k$  (par exemple  $k = 2, 3, 4, 5$ ) sur la collection de données *Iris* fournie dans `scikit-learn` :

```
from sklearn.datasets import load_iris
dataset = load_iris()
x = dataset.data # Liste contenant l'ensemble des donnees
```

**Visualisation de la partition.** Afin d'identifier une taille de partition  $k$  appropriée, visualisez en 2D avec `matplotlib` la partition obtenue, que l'on suppose ici stockée dans la variable `assignments`, comme suit :

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Projection des donnees dans le plan par PCA
pca = PCA(n_components=2)
x_r = pca.fit(x).transform(x) # x apres application de la PCA

# Visualisation des clusters
k = len(set(assignments)) # Nombre de clusters
print(k)
target_names = range(k) # Labels des clusters obtenus
plt.figure()
for i, target_name in zip(range(k), target_names):
    plt.scatter(x_r[assignments == i, 0], x_r[assignments == i, 1],
                label=target_name)
plt.legend(loc='best')
plt.title('Partition obtenue par k-means sur la collection Iris')
plt.show()
```

Lancez plusieurs fois votre implémentation des  $k$ -Moyennes avec le même  $k$  et visualisez le résultat. Que constatez-vous ? Comment expliquez-vous ce phénomène ?

**Complément : évaluation de la partition.** La partition réelle des données d'une collection de `scikit-learn` s'obtient de la manière suivante pour une variable `dataset` définie comme précédemment :

```
y = dataset.target # Liste contenant le cluster reel de chaque donnee
```

En utilisant la partition réelle des données, évaluez la partition obtenue par votre implémentation des  $k$ -Moyennes. On s'appuyera pour cela sur la mesure *Normalized Mutual Information* (NMI) qui permet de comparer deux partitions. Une implémentation de la NMI est disponible dans `scikit-learn` :

```
from sklearn.metrics.cluster import normalized_mutual_info_score
```

Lancez plusieurs fois votre implémentation des  $k$ -Moyennes avec le même  $k$  et noter la NMI obtenue. Est-ce que le résultat obtenu est cohérent avec votre observation précédente ?