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 partionnement des données (*clustering*).

Entrée :
- la précision ϵ pour le critère d'arrêt
- le nombre de clusters à obtenir k
- l'ensemble des données $X = (x_1, \ldots, x_n)$ avec n le nombre total de données

Initialisation :
- Initialiser les centroïdes $C^{(0)} = (c_1^{(0)}, \ldots, c_k^{(0)})$ par k données choisies aléatoirement dans X
- $l \leftarrow 0$

répéter

  `# Mise-à-jour de l'appartenance de chaque donnée à un cluster`

  pour $i = 1, \ldots, n$ **faire**

    $a_i^{(l)} = \underset{1 \leq j \leq k}{\operatorname{argmin}} ||x_i - c_j^{(l)}||$ `# Trouver le centroïde le plus proche de` x_i

  fin

  `# Mise-à-jour des centroïdes par la moyenne des données des clusters`

  pour $j = 1, \ldots, k$ **faire**

    $S_j = \left\{ i \,\middle|\, 1 \leq i \leq n \text{ et } a_i^{(l)} = j \right\}$ `# 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 $A^{(l)} = (a_1^{(l)}, \ldots, a_n^{(l)})$, centroïdes $C^{(l)} = (c_1^{(l)}, \ldots, c_k^{(l)})$

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 donneés *Iris* fournie dans `scikit-learn` :

```python
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 :

```python
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 :

```python
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` :

```python
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 ?