TD : ALGORITHMES POUR L'INTELLIGENCE ARTIFICIELLE
== ALGORITHME DES K MOYENNES ==

Exercice basé sur le sujet CCINP - 2024 - MPI – INFORMATIQUE

On cherche à classer n = 5 documents textuels $doc_i, i \in [\![1, n]\!]$ dans lesquels les termes T_1, T_2 et T_3 apparaissent un certain nombre de fois, ces occurrences étant décrites dans le tableau suivant :

	doc_1	**doc_2**	**doc_3**	**doc_4**	**doc_5**
T_1	1	2	0	1	1
T_2	3	1	1	3	0
T_3	3	0	0	1	1

Chaque document doc_i est donc représenté par un ensemble de p = 3 valeurs. On notera $d_i, i \in [\![1, n]\!]$ un vecteur de $\mathbb{N}^3$, de composantes $d_{ij}, j \in [\![1, p]\!]$, d_{ij} indiquant le nombre d'occurrences du terme T_j dans le document doc_i.

On cherche à voir si des textes traitent des mêmes thématiques, en faisant l'hypothèse que des textes sont sémantiquement proches si des termes communs apparaissent.

I) ALGORITHME DES K-MOYENNES

L'algorithme des k-moyennes est un algorithme de classification itératif. Partant de k centres initiaux, il affecte chaque point au centre le plus proche (au sens de la distance δ), puis recalcule les centres comme la moyenne des points affectés à chaque classe. Il s'arrête quand les classes ne changent plus.

1. Recopier et remplir le tableau suivant en appliquant l'algorithme des k-moyennes en prenant k = 2 et d_2 et d_5 comme centres de classe initiaux. On utilisera la distance $\delta(d_i, d_j)$:

$$\delta(d_i, d_j) = \sum_{\ell=1}^{p} |d_{i\ell} - d_{j\ell}|$$

On notera c_i les centres de classe et $\mathcal{A}_i$ les classes correspondantes.

Itération	$\boldsymbol{c_1}$	$\boldsymbol{c_2}$	$\boldsymbol{\mathcal{A}_1}$	$\boldsymbol{\mathcal{A}_2}$
1	d_2	d_5		
2				
3				

2. Donner la complexité de l'algorithme en fonction du nombre de documents n, de la dimension caractéristique des documents p et du nombre de classes (clusters) k et du nombre d'itérations I.

Voici le jeu de données à utiliser dans toutes les fonctions de ce TD qui est déjà implémenté dans le fichier source « 6. TD4 - K-moyennes.py » :

```python
import numpy as np

# Jeu de données : 5 documents, 3 termes (T1, T2, T3)
documents = [
    np.array([1, 3, 3]),   # d1
    np.array([2, 1, 0]),   # d2
    np.array([0, 1, 0]),   # d3
    np.array([1, 3, 1]),   # d4
    np.array([1, 0, 1]),   # d5
]
```

3. Implémenter la fonction distance `delta(di,dj)` calculant la distance de Manhattan entre deux vecteurs NumPy. Tester avec δ`(d1,d2)=6.0` et δ`(d2,d3)=2.0`.

4. Écrire une fonction `affectation(documents,centres)` qui retourne, pour chaque document, l'indice du centre le plus proche au sens de δ. La fonction retourne une liste d'entiers de même longueur que documents.

   Vérifier :
   ```
   >>> c1 = documents[1]
   >>> c2 = documents[4]
   >>> affectation(documents,[c1,c2])
   [1, 0, 0, 1, 1]
   ```

5. Écrire une fonction `recalcul_centres(documents,indices_centres,k)` qui calcule les nouveaux centres de classe comme la moyenne des documents affectés à chaque classe.

   Vérifier :
   ```
   >>> recalcul_centres(documents,[1, 0, 0, 1, 1],2)
   [array([1., 1., 0.]),
    array([1.        , 2.        , 1.66666667])]
   ```

6. À l'aide des fonctions précédentes, implémenter l'algorithme complet `k_moyennes(documents, k, centres_init)`. L'algorithme s'arrête quand les affectations des documents dans les classes ne changent plus entre deux itérations. Retourner les affectations finales et les centres.

   Vérifier :
   ```
   >>> centres_init=[documents[1], documents[4]]
   >>> k_moyennes(documents, 2, centres_init)
   ([1, 0, 0, 1, 0],
    [array([1.        , 0.66666667, 0.33333333]),
     array([1., 3., 2.])])
   ```

7. Appliquer `k_moyennes` sur `documents_grands`. Pour `k_moyennes`, utiliser `k=3` et trois documents du début de chaque groupe comme centres initiaux.
   ```
   Vérifier : Labels finaux doc_grands: [0, 0, 0, 1, 1, 1, 1, 2, 2, 2,
                                         2, 2, 2, 1, 1, 1, 1, 0, 0, 0]
   ```

II) ALGORITHME « SINGLE – PASS »

On souhaite maintenant traiter ce problème par une méthode dite « single pass » décrite dans l'algorithme 1.

L'algorithme « Single Pass » est une méthode de classification à une seule passe sur les données : chaque document est traité une seule fois, dans l'ordre. Un seuil θ pilote la création de nouvelles classes. Si un document est trop éloigné de tous les centres existants (distance > θ), une nouvelle classe est créée. Sinon, il est affecté au centre le plus proche, qui est immédiatement recalculé.

Algorithme 1 - Algorithme Single Pass

Entrées : $(d_1 \cdots d_n)$ les vecteurs des documents, θ un seuil appartient à $\mathbb{R}$.
Sorties : $(\mathcal{A}_1 \cdots \mathcal{A}_j)$ les classes de centres $(c_1 \cdots c_j)$
début

$c_1 = d_1$; // Initialisation
$\mathcal{A}_1 = \{d_1\}$;
$j = 1$;
pour *i de 2 à n* **faire**
  pour *k de 1 à j* **faire**
    Étape **(i)** Calculer $\delta(d_i, c_k)$;
  si Étape **(ii)** $\delta(d_i, c_k) > \theta \; \forall c_k$ **alors**
    $j = j + 1$; // Création d'une nouvelle classe
    $\mathcal{A}_j = \{d_i\}$;
    $c_j = d_i$;
  sinon
    // Indice du centre de classe le plus proche de d_i au sens de δ
    Étape **(iii)** $\ell = \arg\min_{1 \le k < i} (\delta(d_i, c_k))$;
    $\mathcal{A}_\ell = \mathcal{A}_\ell \cup \{d_i\}$; // Affectation de d_i à la classe ℓ
    $c_\ell = \frac{1}{|A_\ell|} \sum_{d_i \in \mathcal{A}_\ell} d_i$; // Recalcul de c_ℓ

8. Appliquer l'algorithme 1 avec θ = 5.0. Détailler les résultats des étapes de l'algorithme.

9. Analyser la complexité temporelle de l'algorithme « Single Pass » en fonction de n (nombre de documents), p (dimension des vecteurs) et j (nombre de classes créées). Comparer avec la complexité des k-moyennes.

10. Écrire une fonction `dist_max(di,centres,theta)` qui vérifie si le document d_i est plus éloigné que θ de tous les centres existants. La fonction retourne `True` si $\delta(\mathrm{d}_i, \mathrm{c}_\mathrm{k}) > \theta$ pour tout c_k, et `False` sinon.

```
Tester : >>> c_init = [documents[1]]
         >>> dist_max(documents[0], c_init, 5.0) # True (δ=6 > 5)
         True
         >>> dist_max(documents[2], c_init, 5.0) # False (d3,c2)=2
         False
```

11. Écrire une fonction `dist_min(di,centres)` qui retourne l'indice ℓ du centre le plus proche de d_i au sens de δ (étape (iii) de l'algorithme « Single Pass »).

Tester :
```
>>> dist_min(documents[0],documents)
0
>>> dist_min(documents[1],documents)
1
>>> dist_min(documents[4],documents)
4
```

12. Écrire une fonction `recalcul_centre_sp(centres,l,classe_l)` qui met à jour le centre c_l après l'ajout du document d_i à la classe $\mathcal{A}_l$. Le nouveau centre est la moyenne de tous les documents de $\mathcal{A}_l$.

Tester :
```
>>> centres = [documents[0], documents[1]]
>>> classes = [[documents[0]], [documents[1],documents[2]]]
>>> recalcul_centre_sp(centres,1,classes[1])
[array([1, 3, 3]), array([1., 1., 0.])]
```

13. Implémenter l'algorithme « Single Pass » complet. La fonction `single_pass(documents,theta)` prend en entrée la liste des documents et le seuil θ, et retourne les centres finaux et les classes.

```
Tester : >>> single_pass(documents,5.0)
           ([array([1., 3., 2.]),
             array([1.        , 0.66666667, 0.33333333])],
            [[array([1, 3, 3]), array([1, 3, 1])],
             [array([2, 1, 0]), array([0, 1, 0]),array([1, 0, 1])]])
```