K-moyennes vs. les Sup

On utilisera pour ce TP `Spyder`. On pourra télécharger les notes anonymisés des Sup ici.

I Fonctions auxiliaires

Les notes des étudiants de Sup au projet musiques sont donnée dans un fichier `csv` ayant 15 champs. Le premier champ correspond à un identifiant numérique de l'étudiant et les 14 champs suivants sont les notes, sur 9, à chacune des 14 questions du projet. L'identifiant numérique est un entier entre 1 et `133`.

Ses données seront d'abord importée dans le programme sous la forme d'un dictionnaire dont les clés sont les identifiant numérique de chaque étudiant et les valeurs associées, des tuples de 14 flottants représentant les notes de l'étudiant dans le même ordre que dans le fichier `csv`.

On notera `vec` le type tuples de 14 flottants.

Exercice 1. *Chargement des données.*

Écrire une fonction `chargement(chemin : str) -> dict[int,vec]` qui prend en entrée le chemin du fichier `csv` contenant les données et qui renvoie le dictionnaire décris ci-dessus.

On utilisera `float(s)` pour convertir en floattant une chaîne de caractère représentant un floattant. On utilisera `s.split(',')` pour séparer une chaîne de caractère selon les virgules et obtenir la liste des chaînes entre chaque virgule.

N.B. : ici.

Exercice 2. *Tirage aléatoire*

On aura besoin du tirage aléatoire de `n` étudiants pour initialiser les centroïde.

Implémenter une fonction `init_centroide(notes : dict, k : int) -> list[vec]` qui prend le dictionnaire des données en entrée, tire au hasard `k` clés *distinctes* du dictionnaire et renvoie la liste des vecteurs de notes des étudiants tirés au hasard.

N.B : documentation du module random

Exercice 3. *Distances*

On aura besoin de la distance entre deux vecteurs de $\mathbb{R}^{14}$.

Implémenter une fonction `distance(v1 : vec, v2 : vec) -> float` qui prend deux vecteurs de notes et renvoie la distance euclidienne de $\mathbb{R}^{14}$ entre ces deux vecteurs.

Exercice 4. *Barycentre*

On aura besoin du barycentre (moyenne) d'un ensemble de vecteurs de $\mathbb{R}^{14}$. Comme les vecteurs seront toujours pris parmi les vecteurs de notes des étudiants, l'ensemble de vecteur sera représenté par la liste des identifiant des étudiants dont les vecteurs sont dans l'ensemble.

Implémenter une fonction `moyenne(notes : dict, cat : list[int]) -> vec` qui prend le dictionnaire des données en entrée, un ensemble d'identifiants sous forme de liste, et renvoie le barycentre des vecteurs de notes des étudiants dont l'identifiant est dans `cat`.

II k-moyenne

L'algorithme des k-moyennes se déroule selon les étapes suivantes :

1. Initialiser `k` centroïdes aléatoirement parmi l'ensemble de données.
2. Associer chaque donnée au centroïde dont elle est le plus proche, ce qui crée `k` catégories.
3. Recalculer le centroïde de chaque catégorie.
4. Recommencer l'étape 2 tant que l'un au moins des centroïde a bougé.

Dans cet algorithme l'ensemble des données comme l'ensemble représentant chacune des `k` catégories, sera représenté par une liste d'entier, les entiers étant des identifiants d'étudiants.

Les différentes structures utilisées par l'algorithme seront :

— `notes : dict` le dictionnaire de l'ensemble des notes des étudiants,
— `moy : list[vec]` la liste des `k` centroïdes utilisés de chacune des `k` catégories,
— `cat : list[list[int]]` la liste des `k` catégories (ensembles d'identifiants) telle que la catégorie à l'indice `i` soit celle associée au centroïde d'indice `i`.

La fin de l'algorithme survient à la convergence du processus. Informatiquement parlant, les flottants ayant une précision ne dépassant pas 10^{-16}, on arrêtera l'algorithme lorsque les centroïdes n'ont pas été modifiés entre deux itérations.

Exercice 5. *Plus proche*

Implémenter une fonction `mind(p : vec, moy : list[vec]) -> int` qui prend en entrée un vecteur de l'espace et la liste des `k` centroïdes calculés à une étape donnée et renvoie l'indice du centroïde le plus proche du point.

Exercice 6. *Catégories*

Implémenter une fonction `cat(notes : dict, moy : list[vec]) -> list[list[int]]` qui prend en entrée le dictionnaire des données et la liste des `k` centroïdes calculés à une étape donnée et renvoie la liste des catégories associées à chaque centroïdes.

Exercice 7. *Actualisation*

Implémenter une fonction `actu(notes : dict, cat : list[list[int]]) -> list[vec]` qui prend en entrée le dictionnaire des données et la liste des `k` catégories calculés à une étape donnée et renvoie la liste des nouveau centroïdes de chaque catégories.

Exercice 8. *Algorithme*

Implémenter la fonction `k-moyenne(notes : dict, k : int) -> list[list[int]]` qui implémente l'algorithme des `k-moyenne` et renvoie l'ensemble des `k` catégories trouvées.