Travaux pratiques #2 — Algorithme des k-moyennes

Exercice 1 — *Algorithme des k-moyennes en 2 dimensions.*

Description générale. Le problème des k-moyennes est un problème de **partitionnement de données**. Étant donnés un ensemble de `Point` (les **données**) et un entier $k \in \mathbb{N}$, l'objectif est de séparer les données en k groupes (**clusters**) tout en minimisant une certaine fonction. Une fois les clusters calculés, la **distance** d'une donnée à son cluster est calculée en prenant la distance avec la **moyenne** des points du cluster : la fonction à minimiser est alors la somme des carrés de ces distances.

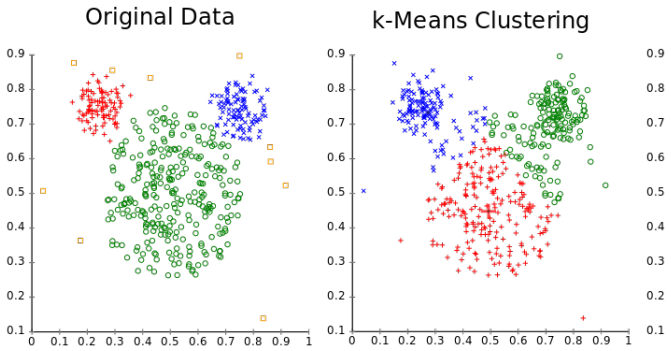

FIGURE 1 – Un exemple de partitionnement obtenu avec l'algorithme des k-moyennes à partir d'un ensemble de données étiqueté. Image extraite de Wikipedia.

> Trouver la meilleure partition possible est un problème NP-difficile, et il faut donc implémenter une heuristique pour obtenir un résultat satisfaisant.
>
> Un problème d'optimisation difficile.

L'algorithme pas-à-pas. L'algorithme des k-moyennes est un algorithme itératif, qui sera répété un nombre `iter` de fois fixé par l'utilisateur (choisir 100 pour les exemples). L'algorithme maintient des **centroïdes**, qui sont les coordonnées moyennes de tous les points d'un cluster calculé à l'itération `t`. Initialement, pour `t = 0`, k centroïdes sont choisis au hasard parmi les points de l'ensemble de données. Ensuite, les opérations suivantes sont répétées `iter` fois :

(i) chaque point choisit le centroïde dont il est le plus proche et s'intègre dans le cluster correspondant

(ii) les centroïdes sont mis à jour

L'étape (ii) consiste simplement à générer la moyenne des points affectés dans chaque cluster. Autrement dit, les coordonnées de chaque centroïde sont mises à jour en faisant la moyenne des abscisses et des ordonnées de tous les points du cluster.

La classe `kmeans`. Même si l'implémentation peut se faire sans utiliser une classe, nous choisissons ce fonctionnement en créant une classe `kmeans`. Cette dernière doit contenir comme attributs et méthodes :

- un `vector` de `Point` (les données à partitionner)
- une `map` associant à chaque `Point` son numéro de cluster (il faudra faire attention au choix du type de clé de la `map`).
- un attribut entier k représentant le nombre de clusters à générer
- un tableau de taille k représentant les centroïdes de chaque groupe
- une méthode `initialiser` permettant de choisir aléatoirement k centroïdes parmi les points
- une méthode `calculer` implémentant l'algorithme des k-moyennes

— une méthode `afficher` permettant d'afficher les clusters sur la console.

La classe `Point` peut être enrichie en enregistrant la distance *minimum* au centroïde du cluster associé.

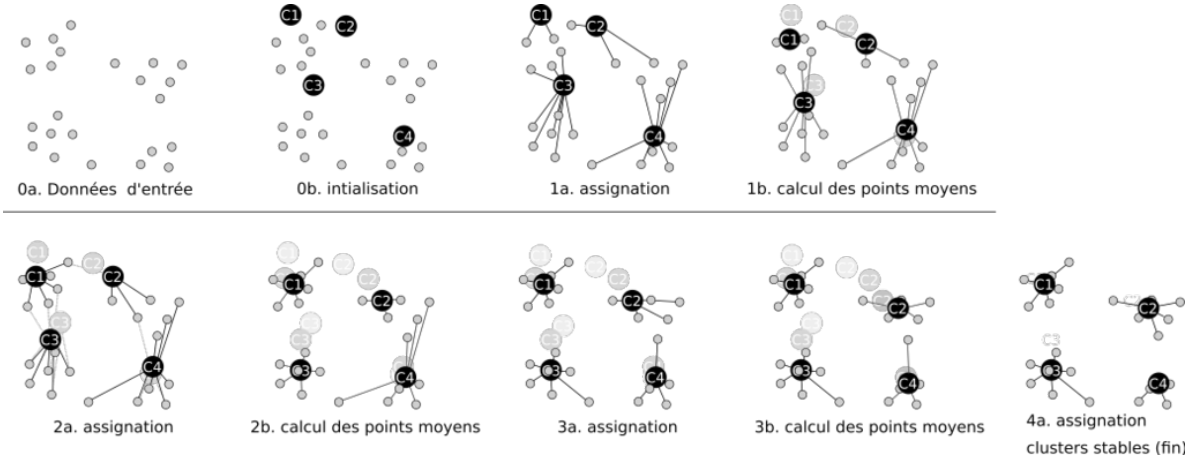

FIGURE 2 – Illustration du déroulement de l'algorithme des k-moyennes. Image extraite de Wikipedia.

Initialisation des centroïdes. Avant la première itération de l'algorithme, il est nécessaire de choisir k centroïdes pour initialiser les calculs. Pour cela, on choisit aléatoirement k points de l'ensemble de données.

> Il est nécessaire de générer k centroïdes différents : il faudra donc s'assurer de ne pas générer deux fois la même *valeur* aléatoire. De plus, il faudrait **idéalement** que ces k centroïdes soient choisis de façon uniforme. Deux solutions sont envisageables :
> — utiliser `rand()` (qui n'est pas uniforme)
> — utiliser la `STL`, notamment `shuffle` qui permet de mélanger aléatoirement les éléments d'un conteneur. Attention, cette fonction modifie l'ordre des éléments en place, il ne faut donc pas l'appliquer directement sur le `vector` de `Point` attribut de la classe `k-means`.
>
> La génération aléatoire.

Calcul des clusters. Une fois l'algorithme terminé, les clusters sont générés en affectant chaque point au cluster correspondant à son centroïde le plus proche. Il sera nécessaire de calculer la **distance euclidienne** entre deux points, soit dans la classe `Point` soit avec une méthode externe aux classes.

Les tests. Dans un premier temps des tests seront lancés sur un ensemble de 50 points générés aléatoirement, avec des coordonnées comprises entre 0 et 30, et la valeur de k sera fixée à 5.

Lecture des données. Afin de rendre l'utilisation du code plus pertinente, les points doivent être générés à partir d'un ensemble de coordonnées fournies dans un fichier. Le format est simple : chaque ligne contient deux valeurs entières séparées par un espace. À partir de la **documentation officielle**, modifier le programme pour que les points ne soient plus générés aléatoirement mais chargés depuis un fichier (voir `CELENE` pour un exemple de données).

Visualiser la solution. La visualisation en console n'est évidemment pas satisfaisante. Un fichier `python` est fourni sur `CELENE` et nécessite simplement l'existence d'un fichier `output.csv` (dans le même dossier) dont la première ligne est `x,y,c` et où chacune des lignes suivantes décrit un `Point` sous cette forme : abscisse,ordonnée,cluster. Le fichier `visualisation.py` attend comme argument le nombre de clusters à visualiser, qu'il faudra donc calculer dans la classe `kmeans`.