

K-Means

1 But

Le but de ce travail est d'implémenter la méthode des k-moyennes pour partitionner des données de façon non supervisée.

2 Théorie

2.1 Le partitionnement de données

Le partitionnement de données (*clustering*) consiste à grouper des données en *clusters* de façon à ce que les éléments d'un même groupe soient similaires les uns aux autres ou d'un élément virtuel représentant le centre de gravité du groupe. On entendra "similarité" au sens de la fonction de calcul de similarité choisie (voir section "calcul de similarité").

2.2 Type des données

Chaque point sera un vecteur de valeurs numériques à N dimensions. Ces valeurs seront des entiers dans un premier temps mais il est simple d'étendre les définitions aux types à virgule flottante.

2.3 Présentation de l'algorithme

La méthode k-moyennes (k-means en anglais) est un algorithme qui permet de partitionner un ensemble de "points". Ce processus de partitionnement s'effectue de manière non supervisée, c'est-à-dire que l'algorithme est appliqué sur des données non étiquetées et sans avoir recours à un retour externe sur les résultats.

2.3.1 Initialisation des clusters

Une fois qu'on a défini le nombre de clusters souhaité (arbitrairement), il s'agit de choisir les *centroïdes* (points représentant les centres de gravité de chaque *cluster*) initiaux. Ces derniers peuvent être choisis en faisant recours à un degré variable d'aléatoire, selon les résultats souhaités. Il faut simplement s'assurer qu'ils ne soient pas trop proches les uns des autres.

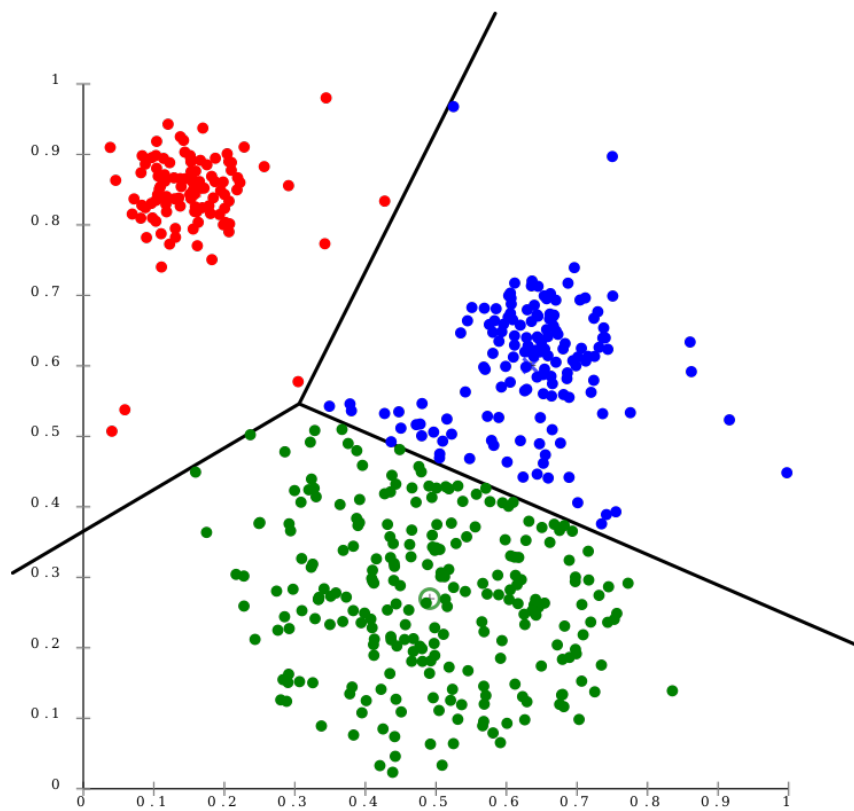


Figure 1: kmeans clustering

2.3.2 Etapes

L’assignation des points à leurs *clusters* se déroule de la manière suivante:

0. Pour chaque point, il faut:
 1. Calculer la distance entre le point et les *centroïdes*.
 2. Prendre le centroïde qui correspond à la distance trouvée la plus petite.
 3. Assigner le point au *cluster* du *centroïde* ainsi trouvé.
 4. Calculer le nouveau centre de gravité (*centroïde*) du *cluster*.
5. Si la position d’au moins un *centroïde* a changé durant le parcours de l’ensemble des points, réitérer cette suite d’opérations depuis l’étape 0.

2.4 Calcul de similarité

Les fonctions de calcul de similarité (qu’on peut vulgariser comme “fonction distance”) de deux points entre eux sont:

- indépendantes de l’algorithme k-means
- choisies en fonction du résultat souhaité

Elles prennent en paramètre deux points et retournent une valeur numérique. Une grande valeur de retour indiquera que les deux points sont éloignés. Inversement, une petite valeur de retour indiquera qu’ils sont proches.

2.4.1 Exemples

Voici quelques exemples de fonctions de calcul de similarité.

2.4.1.1 Distance Euclidienne

La distance euclidienne entre deux points est définie par:

$$d_e(a, b) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

2.4.1.2 Distance de Manhattan

La distance de Manhattan entre deux points est donnée par la formule suivante:

$$d_m(a, b) = \sum_{i=1}^n |a_i - b_i|$$

2.4.1.3 Distance de Chebyshev

On calcule la distance de Chebyshev ainsi:

$$d_c(a, b) = \max(|a_1 - b_1|, \dots, |a_n - b_n|)$$

En d'autres termes, on sélectionne la position à laquelle les coefficients des deux vecteurs sont les plus éloignés et cette différence (en valeur absolue) est la distance de Chebyshev.

3 Implémentation

3.1 Format d'entrée

L'entrée sera un fichier spécifié en argument au programme. Si aucun argument est fourni, le fichier d'entrée sera l'entrée standard (stdin).

Le fichier contiendra:

- sur la première ligne, le nombre de dimensions des points de l'ensemble
- sur la deuxième ligne, le nombre de clusters voulu
- un point par ligne, sous forme de N valeurs séparées par des virgules

```
<nombre de dimensions>
<nombre de clusters>
x1,x2,...,xn
y1,y2,...,yn
...
```

3.2 Format de sortie

La sortie sera écrite dans un fichier spécifié en argument au programme. Si aucun argument est fourni, le fichier de sortie sera la sortie standard (stdout).

Le fichier contiendra:

- sur la première ligne, le nombre de dimensions des points de l'ensemble
- sur la deuxième ligne, le nombre de clusters
- une étoile "*" signifie le début d'un cluster
- pour chaque étoile, une liste de points appartenant au même cluster, avec le même format que les points d'entrée

```
<nombre de dimensions>
<nombre de clusters>
*
x1,x2,...,xn
y1,y2,...,yn
...
*
```

z_1, z_2, \dots, z_n
 t_1, t_2, \dots, t_n
...

3.3 Interface utilisateur

Le programme fonctionnera entièrement en ligne de commande. Les formats d'entrée et de sortie ont été choisis pour être "pipe-friendly", ce qui donne un degré de flexibilité supplémentaire à l'utilisateur.

3.4 Déroulement de l'exécution du programme

- appel de la commande avec éventuellement le passage des fichiers d'entrée et de sortie en argument
- initialisation du nombre de dimensions de notre univers
- allocation de la mémoire pour les K groupes
- chargement des données d'entrée en mémoire (possibilité d'intervertir ce point avec le précédent)
- application de l'algorithme k-means
- écriture des résultats formatés dans le fichier de sortie (ou de la sortie standard, selon les arguments passés au programme)

3.5 Fonctions à implémenter

Il faut absolument implémenter les fonctionnalités suivantes:

- gestion des arguments en ligne de commande
- lecture et écriture de fichiers formatés
- initialisation aléatoire et/ou intelligente des centroïdes
- au moins trois fonctions de calcul de similarité / "distance" entre deux points
- calcul du centre de gravité (*centroïde*) d'une partition
- les étapes de l'algorithme k-means

3.6 Travail supplémentaire possible

3.6.1 Affichage en temps réel

Au lieu d'afficher uniquement le résultat final, il s'agirait d'afficher chaque étape effectuée afin de pouvoir observer l'algorithme en action.

3.6.2 Déterminer le nombre de clusters optimal

Faire des recherches sur les méthodes permettant d'optimiser le nombre de clusters et les implémenter. Si l'utilisation de cette fonctionnalité est le souhait de l'utilisateur, un seul caractère non-numérique est utilisé en lieu du nombre de clusters souhaité. De légères modifications du format sont envisageables pour autant qu'elles soient justifiées.

3.6.3 Interface graphique

On peut imaginer créer une interface graphique permettant de visualiser le résultat dans une fenêtre graphique. Pour cela, une option serait de mettre à profit la librairie `gfx` utilisée pendant les labos de physique.

3.7 Vérification du travail réalisé

Il serait bon de mettre en place des tests unitaires pour les fonctions implémentées.

On implémente un algorithme non supervisé. Il est donc difficile de vérifier si le résultat obtenu est “correct”. Néanmoins, un point placé dans le mauvais groupe est facile à identifier par comparaison des distances du point aux différents centroïdes.

4 Travail à rendre

- repo git contenant le code réalisé
- présentation du travail à l’aide d’un support (projection)
- démonstration du programme

5 Références

<https://www.editions-eni.fr/livre/le-machine-learning-avec-python-de-la-theorie-a-la-pratique-9782409031816/extrait-du-livre.pdf>

<https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a>

<https://towardsmachinelearning.org/k-means/>

<https://vitalflux.com/elbow-method-silhouette-score-which-better/>

Figure 1 : <https://stats.stackexchange.com/questions/146339/what-does-cluster-size-mean-in-context-of-k-means>