Apprentissage non-supervisé

ITC PC

M. Charles

13/01/2025

Un peu de Python

Exercice - Mines Ponts 2024

1. Écrire une fonction nommée `nbCaracteres(c:str, s:str) -> int` qui prend comme arguments un caractère `c`, une chaîne `s` et qui renvoie le nombre d'occurences (c'est-à-dire le nombre d'apparitions) de `c` dans `s`. La fonction doit avoir une complexité linéaire en `n`, la longueur de la chaîne `s`.
2. Pour déterminer la liste des caractères utilisés à l'intérieur d'une chaîne `s` on utilise la fonction définie ci-dessous :

```
def listeCaracteres(s: str):
    listeCar = []
    n = len(s)
    for i in range(n):
        c = s[i]
        if not(c in listeCar):
            listeCar.append(c)
    return listeCar
```

   Que renvoie cette fonction lorsque `s='abaabaca'` ? Expliquer succinctement le principe de fonctionnement de cette fonction.
3. En fonction de la longueur `n` de la chaîne et du nombre `k` de caractères distincts dans celle-ci, déterminer la complexité asymptotique dans le pire des cas de la fonction de la question précédente. Par exemple pour `s = 'abaabaca'`, on a $n = 8$ et $k = 3$.
   On négligera la complexité des `append` mais pas celle des tests d'appertenance de la forme `i in L`.

Présentation du problème

Introduction

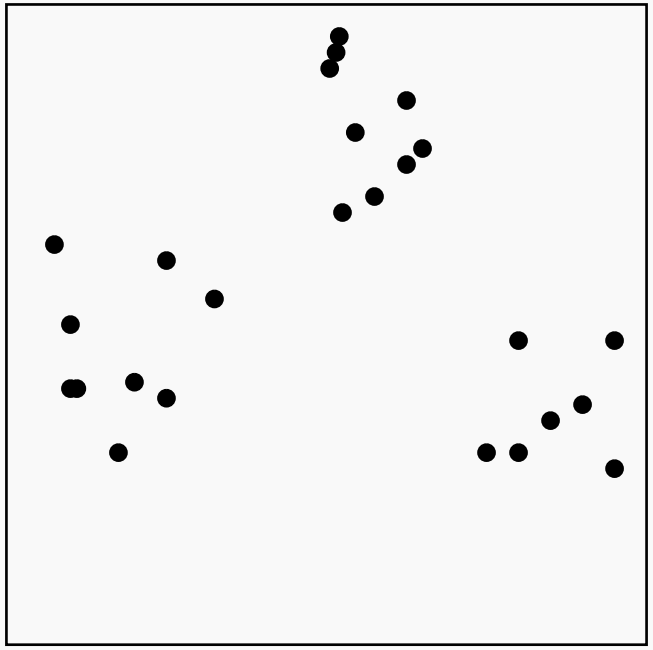
Une équipe scientifique s'est rendue en zone 51 pour étudier les restes de 25 martiens retrouvés dans l'épave d'un ovni. La question se pose de savoir s'il y a plusieurs sous-espèces de martiens.

Introduction

Une équipe scientifique s'est rendue en zone 51 pour étudier les restes de 25 martiens retrouvés dans l'épave d'un ovni. La question se pose de savoir s'il y a plusieurs sous-espèces de martiens.

L'équipe n'a pu faire que deux mesures morphologiques communes à chaque spécimen : la longueur x de l'antenne droite et la longueur y du fémur gauche.

Ainsi, à chaque spécimen on peut associer un point (x, y) du plan :



On a ainsi des données brutes : on ne dispose plus de données d'entrainement.

$\rightarrow$ De quel type d'apprentissage s'agit-il ?

Objectif

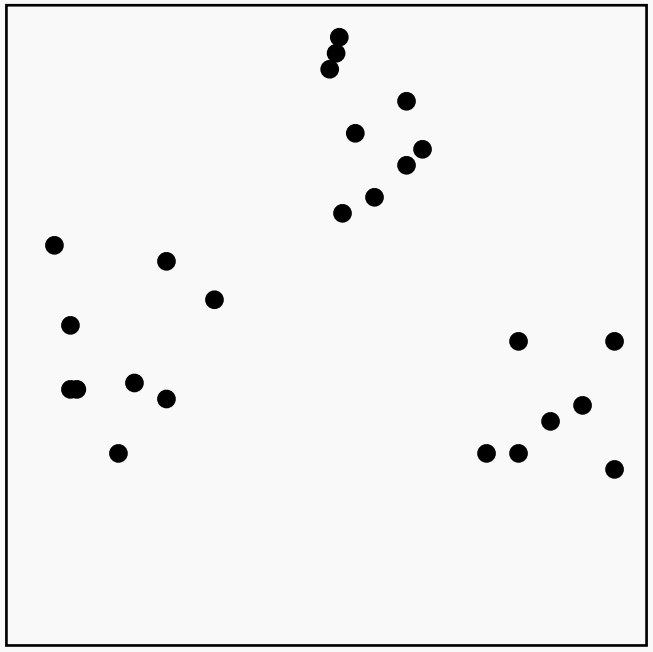
On cherche à partitionner les données : on veut regrouper les données **proches** entre elles. On parle de ***clustering***.

Il s'agit donc d'attribuer une **classe** (ou ***cluster***) à chaque donnée, puis d'interpréter…

Modélisation ; partitionnement

- entrées :
  - des données $x_1, ..., x_n$ dans $\mathbb{R}^d$, et
  - le nombre k de clusters (*c'est un paramètre que l'on choisit*) ;
- sortie :
  - une partition $C_1, ..., C_k$ de $[\![1, n]\!]$ en classes, C_j étant la liste des indices du j-ième cluster.

Exemple pour différentes valeurs de k



La condition d'optimisation

Qu'est-ce qu'une bonne partition ?

On veut que la partition $(C_1, ..., C_k)$ de $[\![1, n]\!]$ minimise la **somme des moments d'inertie**

$$\sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - \mu_j\|^2,$$

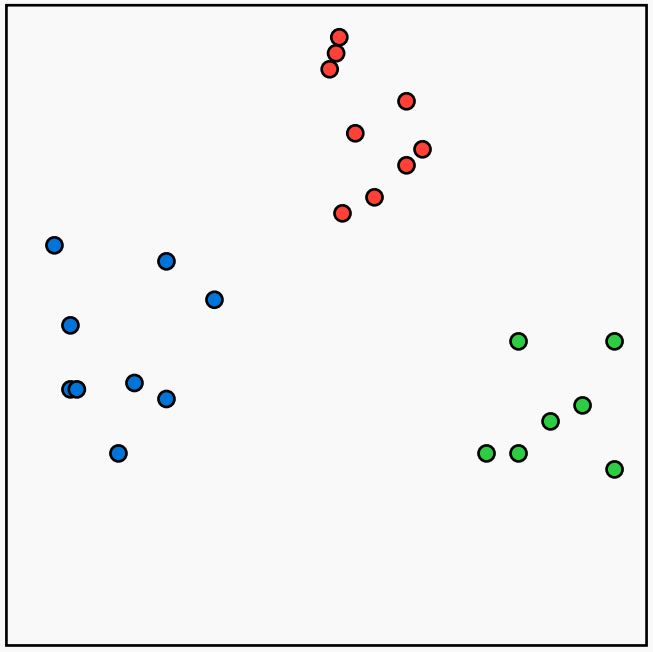
où μ_j est l'**isobarycentre** du cluster numéro j :

$$\mu_j = \frac{1}{|C_j|} \sum_{i \in C_j} x_i.$$

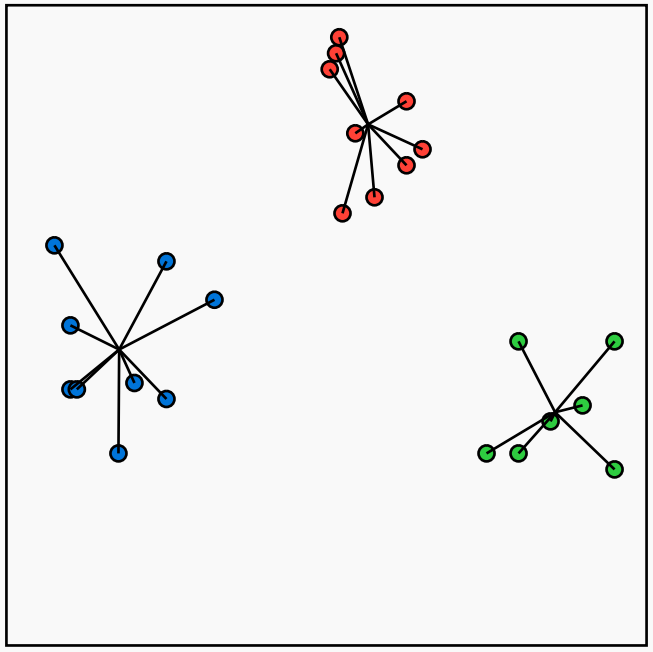
Remarques

- La recherche de la partition optimale est un problème *NP-difficile* dans le cas général.
- Ici k est *fixé*. Si k est quelconque, la partition optimale est la partition triviale avec autant de clusters que de points, ce qui est idiot.

Illustration



Illustration



Algorithme des k-moyennes

Généralités

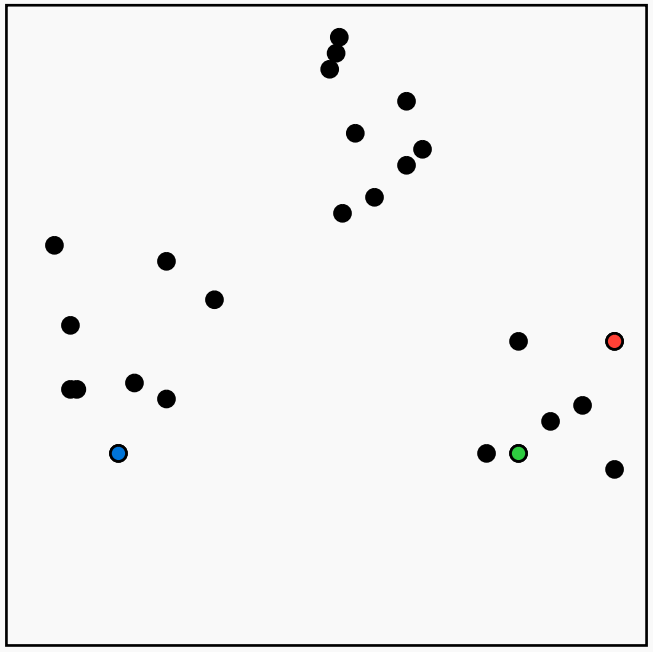
Il s'agit d'un algorithme

- **itératif**,
- basé sur une stratégie **gloutonne**,
- donnant une solution **approchée**.

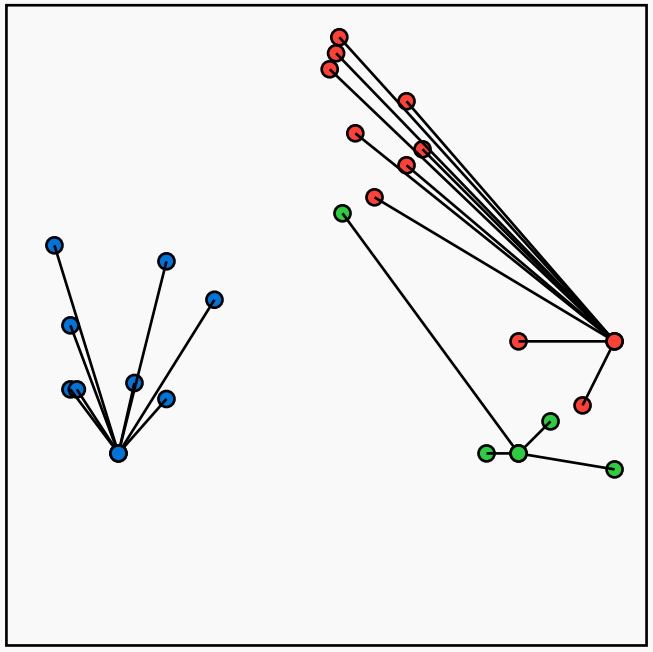
Description de l'algorithme

1. **Initialisation :** on choisit k points M_1, ..., M_k aléatoirement parmi les données. C'est une première approximation des centres des clusters.
2. **Itération** : tant que les centres M_j varient :
   1. on crée les clusters : chaque donnée est affectée au centre M_j le plus proche ;
   2. on met à jour les M_j : chaque M_j devient l'isobarycentre (= moyenne) du cluster que l'on vient de créer.

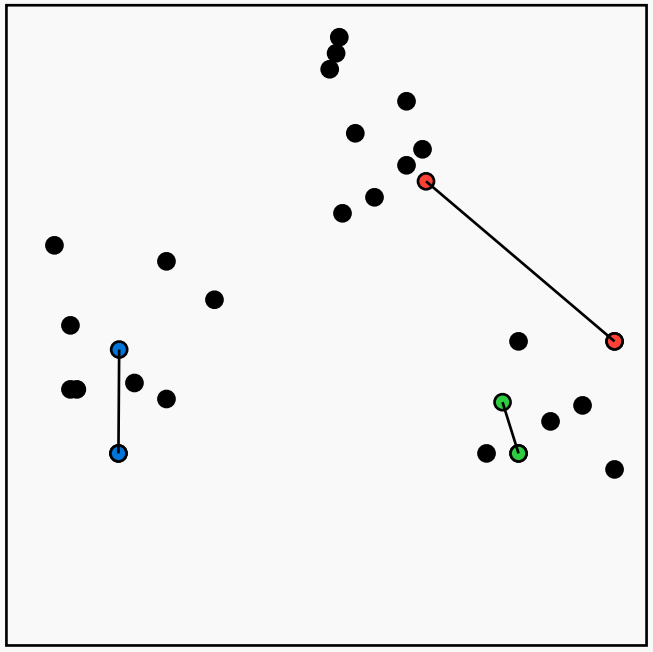
Illustration



Illustration



Illustration



Exercice

1. Écrire une fonction `barycentre(C: list) -> list` qui calcule le barycentre de la liste des points passés en paramètre.
2. On suppose qu'il existe une fonction `classification(p: list, M: list) -> int` renvoie le numéro du cluster pour `p` à partir des centres `M`.

   Construire une fonction `clusters(data: list, M: list) -> list` qui prend en paramètre la liste de points de données, la liste des centres de clusters et renvoient la liste des clusters associés sous forme de liste de listes de points.

Commentaires : terminaison et correction

- On peut montrer que chaque itération fait diminuer la somme des moments d'inertie.
- Il se peut qu'un cluster devienne vide. Auquel cas le comportement de l'algorithme est indéfini et il faut réfléchir…
- Comme il n'y a qu'un nombre fini de partitions possibles, l'algorithme termine nécessairement.
- La solution obtenue dépend du tirage au sort fait à l'initialisation ! (Minimum local pas nécessairement global…)

  $\rightarrow$ On peut envisager de le lancer sur plusieurs points de départ aléatoires et de conserver la meilleure partition ainsi obtenue.

Commentaires : temps de calcul

- Si on attend que les centres soient vraiment immobiles, le temps de calcul risque d'être exponentiel en le nombre de points.
  $\rightarrow$ On peut envisager de s'arrêter quand les centres sont « suffisamment immobiles » pour un sens à choisir.

Commentaires : dépendance à la géométrie des données

- La méthode est satisfaisante si les données sont regroupées en paquets « ronds », mais elle l'est moins si, par exemple, les données sont groupées en paquets alignés.
- Les clusters obtenus ont tendance à être convexes, ce qui n'est pas forcément souhaitable.

Paramétrage de la méthode

Idées

Choisir k est un problème difficile. On peut envisager de tracer la somme des moments d'inertie en fonction de k et de conserver la plus grande valeur de k qui a apporté une diminution significative.

Exercice

Écrire une fonction `moment_tot(clusters: list) -> float` qui prend en paramètre une liste de clusters et renvoie la somme des moments d'inertie.

On pourra utiliser la fonction `barycentre`.