tp_kmeans

January 8, 2026

1 Base de données des Iris

On importe la célèbre base de sonnées `iris`. Elle contient des informations sur 3 variétés : Setosa, Versicolor et Virginica. Un ensemble de fleurs a été étudié. Pour chacune on a noté les informations suivantes : longueur et largueur des sépales, longueur et largeur des pétales.

[1]:
```
from sklearn import datasets
iris = datasets.load_iris()
```

On récupère la liste des informations sur les sépales et pétales :

[2]:
```
D = iris.data
D[:5]
```

[2]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

Chaque ligne de la matrice ci-dessus est l'enregistrement des données pour une fleur.

[3]:
```
Y = iris.target
Y, len(Y)
```

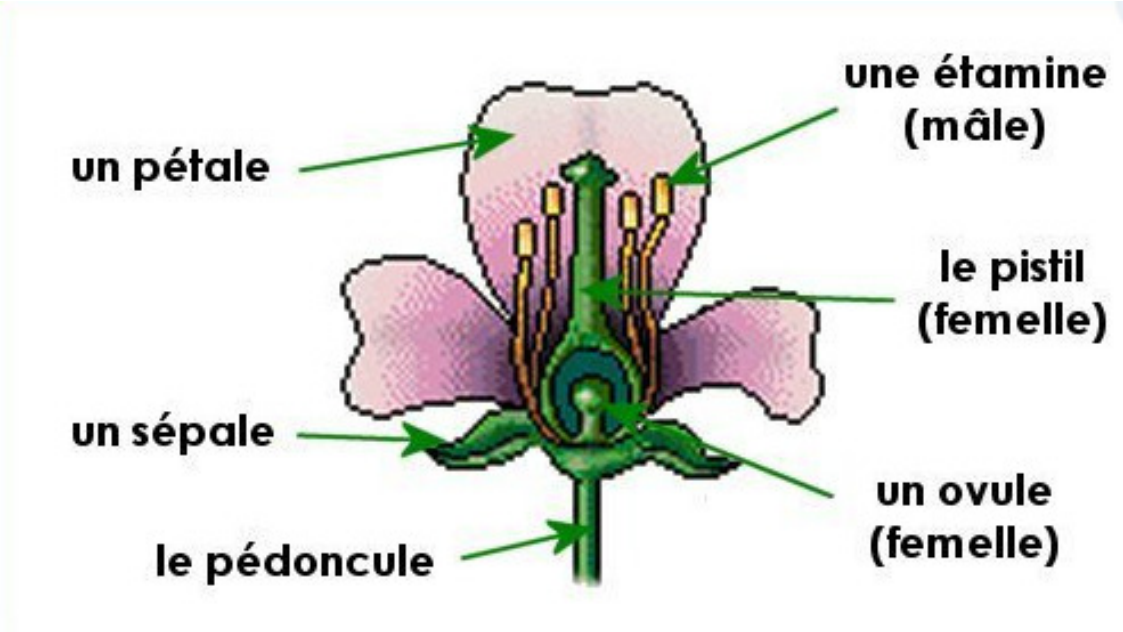
[3]:
```
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 150)
```

Le tableau `Y` indique, pour chaque numéro de ligne de `D` la catégorie à laquelle appartient la fleur correspondante (0 pour Setosa, 1 pour Versicolor et 3 pour Virginica). Dit autrement, le tableau `Y` représente une partition de l'ensemble des données. La donnée numéro `i` appartient à la classe d'équivalence numéro `Y[i]`. Ainsi, `D[i]` désigne les caractéristiques longueur et largeur des sépales de la fleur `i`, longueur et largeur des pétales; et `Y[i]` représente la variété à laquelle elle appartient.

Le tableau `D` est appelé *tableau des vecteurs caractéristiques*; `Y` est le *tableau des étiquettes de classe*.

Remarque

La base de données des Iris stocke les vecteurs caractéristiques sous forme de tableaux `numpy`. On accède aux items de ces objets comme à ceux d'une liste Python. Il y a des différences entre une liste Python et



Dans tout ce qui suit D désigne une liste de coordonnées de points à classifier et Y les classes connues de ces points. Pour fixer les idées on prendra D, Y égaux aux tableaux définis ci-dessus mais ce pourrait être tout autre chose.

Afin de récupérer la variété d'une fleur, on met toutes les données et leurs variétés dans un dictionnaire.

[4]:
```
donnees = {}
for i,data in enumerate(D):
    donnees[tuple(data)]=Y[i]
donnees[tuple(D[82])]
```

[4]: 1

On donne la fonction qui calcule le carré de la distance euclidienne :

[5]:
```
def d(x,y):
    s = 0
    for i in range(len(x)):
        s+=(x[i]-y[i])**2
    return s
```

[6]:
```
d([5.1, 3.5, 1.4, 0.2],[4.7, 3.2, 1.3, 0.2])
```

[6]: 0.25999999999999945

2 Algorithme kmeans

On implémente l'algorithme des k-moyennes avec une initialisation de Forgy. La distance utilisée dans toute cette section est la fonction `d` (voir à la fin de la section Base de données des Iris.

On construit une partition du nuage de sorte que la somme des distances de chaque point du nuage au centroïde de son cluster soit la plus petite possible.

2.0.1 Question 1

Ecrire la fonction `proches(x:[float],L:[[float]])->[int]` qui prend en paramètre un point x et une liste de points L et retourne une liste contenant l'indice d'un des points de L le plus proche de x. Dans les faits, la fonction retourne donc une liste d'un élément : l'indice d'un des centroïdes les plus proches de x.

On peut préférer une version qui retourne la liste de tous les indices des points les plus proches de x. Dans ce cas, le numéro du cluster de x peut-être n'importe lequel parmi les points de la liste renvoyée. Il faudra alors faire un choix.

Pour des raisons de compatibilité de type entre les deux versions, on renvoie donc donc soit une liste d'un seul indice (version 1) soit une liste de plusieurs indices (version 2).

[7]:
```
#***********
#version qui renvoie tous les indices des points les + proches
def proches(p: [float], L: [[float]]) -> [int]:
    dmin = d(p, L[0])
    indices = [0]
    for i in range(1, len(L)):
        dist = d(p, L[i])
        if dist < dmin:
            dmin = dist
            indices = [i]
        elif dist == dmin:
            indices.append(i)
    return indices

#version qui renvoie un unique indice de point le plus proche
def proches(x,L):#V2
    _,i = min([(d(x,L[i]),i) for i in range(len(L))])
    return [i]
```

[8]:
```
E = [[1, 5],[4, 1], [-1,  2], [5,1]]
x = [4.5,0.5] # x équidistant de deux points de L
proches(x,E)
```

[8]: [1]

2.0.2 Question 2

Ecrire la fonction `moyenne(E)` qui retourne l'isobarycentre des points de E; E étant une liste de points représentés par des tableaux de même dimension. Le comportement de la fonction lorsque E est vide n'est pas précisé.

Dans l'algorithme des k-moyennes, E désigne bien entendu un cluster.

[9]:
```
#***************
def moyenne(E):
    bar = [0]*len(E[0])
    for e in E:
        for j in range(len(e)):
            bar[j]+=e[j]
    return [b/len(E) for b in bar]
```

[10]:
```
E = [[1, 5], [4, 1], [-1,  2],[5,1]]
moyenne(E)
```

[10]:
```
[2.25, 2.25]
```

En pratique, le cluster E peut être vide : cela arrive même assez souvent. Voici comment traiter le problème sans bloquer votre programme :

[11]:
```
E = []
try:
    m = moyenne(E)
except IndexError:
    m = [3.,6.]#ou toute autre valeur
print(m)
```

```
[3.0, 6.0]
```

2.0.3 Question 3

Pour initialiser l'algorithme kmeans, on choisit la méthode de Forgy : choix aléatoire de k centroïdes provisoires. La fonction `sample` du module `random` est faite pour cela.

La fonction `forgy(N,k)` prend en paramètres un nuage de points N et un nombre de clusters k. Elle renvoie k points du nuage choisis aléatoirement.

[13]:
```
from random import sample
L = [10,20,30,40,50,60,70]
sample(L,2)# choix de deux items aléatoires de L
```

[13]:
```
[20, 50]
```

[14]:
```
sample(D,3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
Cell In[14], line 1
----> 1 sample(D,3)

File ~/anaconda3/lib/python3.12/random.py:413, in Random.sample(self,␣
 ↪population, k, counts)
    389 # Sampling without replacement entails tracking either potential
    390 # selections (the pool) in a list or previous selections in a set.
    391
   (…)
    409 # too many calls to _randbelow(), making them slower and
    410 # causing them to eat more entropy than necessary.
    412 if not isinstance(population, _Sequence):
--> 413     raise TypeError("Population must be a sequence.  "
    414                     "For dicts or sets, use sorted(d).")
    415 n = len(population)
    416 if counts is not None:

TypeError: Population must be a sequence.  For dicts or sets, use sorted(d).
```

[18]:
```
#***********
def forgy(N,k):
    indices = list(range(len(N)))
    choix = sample(indices,k)
    return [N[e] for e in choix]
```

[25]:
```
#************
#ne marche pas car N est un numpy.array : sample refise ce type
def forgy(N,k):
    choix = sample(N,k)
    return choix
```

[28]:
```
#*************
import numpy as np
def forgy(N, k):
    indices = np.random.choice(len(N), size=k, replace=False)
    return N[indices]
```

[29]:
```
starting_points = forgy(D,4)#choix de 4 données aléatoires dans D
starting_points
```

[29]:
```
array([[6.9, 3.1, 5.1, 2.3],
       [6. , 3.4, 4.5, 1.6],
       [5.4, 3.7, 1.5, 0.2],
       [6.4, 2.9, 4.3, 1.3]])
```

[15]:
```
from random import randint
randint(0,4)
```

[15]: 1

2.0.4 Question 4

Ecrire la fonction `attribution(N[[float]],C:[int],M:[[float]])->bool*[[[float]]]` qui prend en paramètres un nuage de points `N`, une liste `C` d'entiers telle que `C[i]` donne le numéro du cluster du point d'indice `i` à un instant t et une liste `M` contenant les centroïdes à cet instant t.

La fonction attribue à chaque point `x` du nuage `N` le meilleur cluster possible : celui dont le centroïde est le plus proche de `x`. Pour ce faire, la liste `C` est mise à jour. A la fin de l'algorithme, elle indique pour chaque point quel est son centroïde à l'instant $t+1$.

La valeur renvoyée est un tuple (bouléen, liste des clusters) :

- le bouléen indique si un point du nuage a changé de cluster durant le déroulement de l'algorithme (c.a.d qu'une valeur `C[x]` a changé pour au moins un indice de point `x`).
- la liste des clusters est une liste de listes de points. En position `i`, on trouve le contenu du cluster numéro `i`: une liste de coordonnées de points.

En résumé, `attribution` renvoie un tuple et modifie la liste `C`.

[16]:
```
#**********
def attribution(N,C:[int],M)->bool:
    change = False
    k = len(M)
    clusters = [[] for _ in range(k)]
    for x in range(len(N)):
        i = proches(N[x],M)[0]
        if C[x] != i:
            change = True
        C[x] = i
        clusters[i].append(N[x])
    return change, clusters
```

[17]:
```
N = [[1,0], [2,1], [1.5,0.5],\
    [10,9], [12,11], [10.5,9.5],\
    [5,4], [6,5], [5.5,6]] # nuage

M = [ [10.8,  9.8],[1.5, 0.5],\
     [5.5, 5. ]] # centroïdes initiaux

cluster_number_of_each_point = [0,1,2,0,1,2,2,1,0] # clusters initiaux : au pif␣
↪!
b, cluster_content = attribution(N,cluster_number_of_each_point,M)
print(b)
print(cluster_number_of_each_point)
print(cluster_content)
```

```
True
[1, 1, 1, 0, 0, 0, 2, 2, 2]
```

```
[[[10, 9], [12, 11], [10.5, 9.5]], [[1, 0], [2, 1], [1.5, 0.5]], [[5, 4], [6,
5], [5.5, 6]]]
```

2.0.5 Question 5

'Ecrire la fonction `kmeans(N,k)` qui prend en paramètres un nuage `N` et le nombre de clusters souhaité k et renvoie les clusters formés par la méthode des k-moyennes. La fonction renvoie les clusters (liste de listes de points et un tableau indiquant pour chaque point à quel cluster il est affecté).

L'initialisation se fait par la méthode de Forgy. La fonction lance une boucle qui s'arrête lorque les clusters n'évoluent plus. Si un cluster est vide au tour $i+1$, on lui attribue le centroïde qu'il avait au tour i.

[18]:
```
#**********
def kmeans(N,k):
    centroides = forgy(N,k)# centroides initiaux
    cluster_index = [None] * len(N) # index des clusters initiaux
    change, clusters = attribution(N,cluster_index,centroides) #maj clusters␣
 ↪initiaux
    #rqe : aucun cluster initial n'est vide !
    cpt = 0
    while change:
        #nouveaux centroïdes
        nv_centroides = []
        for i,C in enumerate(clusters):
            if len(C) == 0:
                nv_centroides.append(centroides[i])
            else :
                nv_centroides.append(moyenne(C))
        centroides = nv_centroides #maj centroides
        #maj clusters
        change, clusters = attribution(N,cluster_index,centroides)
        cpt+=1
    print("nb itérations = {}".format(cpt))
    return clusters, cluster_index
```

[19]:
```
clusters, cluster_index = kmeans(D,3)
```

```
nb itérations = 4
```

[20]:
```
[len(c) for c in clusters]# cardinaux des clusters
```

[20]:
```
[38, 62, 50]
```

[21]:
```
#**********
def afficher(clusters,donnees):
    res = []
    for c in clusters:#pour tout cluster
```

```
        dico = {i:0 for i in range(len(clusters))}
        for p in c:#pour tout
            dico[donnees[tuple(p)]]+=1
        res.append(dico)
    for i,d in enumerate(res):
        print("Dans le cluster {}, les effectifs des différentes variétés sont :
↪".format(i))
        for k in d:
            print("  pour la variété {} : {} éléments".format(k,d[k]), end =" ;␣
↪")
        print()
```

2.0.6 Question 6

Ecrire une fonction de tests `afficher(clusters,donnees)` qui prend en paramètres la partition `clusters` obtenue par l'appel `kmeans(N,k)` et un dictionnaire (point, catégories réelles) dont les clés sont les coordonnées des points du nuage et dont les valeurs possibles sont dans $[0, k-1]$.

Pour chaque cluster, on affiche le nombre de points dans les variétés 0 à $k-1$.

[22]:
```
afficher(clusters,donnees)
```

```
Dans le cluster 0, les effectifs des différentes variétés sont :
  pour la variété 0 : 0 éléments ;   pour la variété 1 : 2 éléments ;   pour la
variété 2 : 36 éléments ;
Dans le cluster 1, les effectifs des différentes variétés sont :
  pour la variété 0 : 0 éléments ;   pour la variété 1 : 48 éléments ;   pour la
variété 2 : 14 éléments ;
Dans le cluster 2, les effectifs des différentes variétés sont :
  pour la variété 0 : 50 éléments ;   pour la variété 1 : 0 éléments ;   pour la
variété 2 : 0 éléments ;
```