KMEANS

December 10, 2024

1 Base de données des Iris

On importe la célèbre base de données `iris` comme au TP précédent.

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 l'indice d'un des centroïdes les plus proches de x.

Erratum : pour des raisons de compatibilité, cet indice est encapsulé dans une liste.

[15]:
```
E = [[1, 5],[4, 1], [-1,  2], [5,1]]
x = [4.5,0.5] # x équidistant de deux points de L
proches(x,E)
```

[15]:
```
[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.

[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.

[1]:
```
from random import sample
L = [10,20,30,40,50,60,70]
sample(L,2)# choix de deux items aléatoires de L
```

[1]:
```
[30, 20]
```

[14]:
```
starting_points = forgy(D,4)#choix de 4 données aléatoires dans D
starting_points
```

[14]:
```
[array([5.7, 2.9, 4.2, 1.3]),
 array([6.3, 2.5, 5. , 1.9]),
 array([5.2, 3.5, 1.5, 0.2]),
 array([5.5, 2.3, 4. , 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`.

[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 ainsi qu'un tableau indiquant pour tout point du nuage quel est son cluster.

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.

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

```
nb itérations = 4
```

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

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

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 ;
```