TP5 : méthodes d'intelligence artificielle

1 Algorithme des k-moyennes

Cet algorithme a été décrit dans le cours
Programmer cet algorithme et le tester sur l'exemple du cours.
Les données ainsi qu'une trame suggérant certaines fonctions auxiliaires sont fournies dans le fichier `Kmoy-Acompleter.py`.

Données :

```
import numpy as np
import matplotlib.pyplot as plt
import random as rd

#données :
points = [(-0.14052840523526092, 3.483186147758519),
 (-0.3292743163116352, 3.51042565315023),
 (-0.24417343739852077, 2.3079735744098473),
 (-0.3294571104623631, 4.319863349316977),
 (-0.5756477777142489, 3.5496569813990995),
 ...,
 ...]
```

Fonctions utiles :

```
# Fonctions utiles
def dist(P1,P2):
    """renvoie la distance entre deux points,
    représentés sous forme d'une liste ou d'un couple de deux éléments"""
    ...



def pointLePlusProche(P,listePoints):
    """renvoie le point le plus proche de P dans la liste listePoints
    ainsi que son indice i"""
    ...









    ...
```

Suite des fonctions utiles

```
def regroupement(listePoints , pointsMoyens ):
    """renvoie  une liste L de listes de points.
    L[i]  est la liste des points de listePoints
    dont l'élément le plus proche parmi les points de pointsMoyens
    est pointsMoyens[i]"""
    # k est défini de manière globale
    L = [ [] for _ in range(k) ] # L[i] devra contenir les points
                                 # les plus  proches de pointsMoyens[i]
    #A compléter









def pointMoyen(listePoints):
    assert listePoints != []
    """ signale que l'on rencontre le problème suivant :
        l'une des classe est vide (le calcul de son barycentre
        n'a alors pas de sens).
        La conséquence avec l'algorithme tel qu'il est programmé
        ci-desous était qu'il ne terminait pas.
    """
    #A compléter









    ...
```

PROGRAMME PRINCIPAL

```
# rappel : la liste des points à classer est la liste points
# choix de k
k = 3

#initialisation des points moyens
pointsMoyens  = [ (-2,-1) , (0,3) , (-1,3)]

   #initialisation du premier regroupement
listeGroupes = regroupement(...       , ...              )



nouveauxPointsMoyens = []
for groupe in listeGroupes:
    nouveauxPointsMoyens.append(...                )



while nouveauxPointsMoyens != pointsMoyens :
    pointsMoyens = nouveauxPointsMoyens.copy()
    listeGroupes = regroupement(...          , ...             )

    nouveauxPointsMoyens = []
    for groupe in listeGroupes:
        nouveauxPointsMoyens.append(...                        )


# on fait apparaître maintenant les groupes en différentes couleurs

listeCouleurs = [ 'red' , 'blue' , 'green' , 'yellow']

for i in range(k):
    groupe = listeGroupes[i]
    X = [ point[0] for point in groupe]
    Y = [ point[1] for point in groupe]
    couleur = listeCouleurs[i]
    plt.scatter(X,Y,c=couleur)

plt.show()
```

2 Application de l'algorithme kNN à la reconnaissance de chiffres

Dans cette partie on applique l'algorithme kNN dans une situation plus proche d'une situation réelle.
On va pour cela se servir d'un jeu de données fourni par la bibliothèque `scikit-learn`. Cette bibliothèque est dédiée aux algorithmes d'apprentissage. Nous n'allons pas utiliser ses fonctionnalités, mais juste un jeu de données qu'elle fournit.
On importe tout d'abord ce jeu de données (voir le fichier `kNN_chiffres_Acompleter.py` pour les détails). On en tire un tableau numpy à 3 dimensions représentant un ensemble d'images en noir et blanc figurant des chiffres.
On va séparer cet ensemble d'images en une partie qui servira à l'apprentissage et une autre qui servira aux tests.
Il faut adapter à cette situation l'algorithme kNN vu précédemment. En particulier :

- la distance doit être modifiée (on prend la distance euclidienne dans un espace où chaque pixel est une coordonnée) ;
- il est préférable ici de ne pas utiliser un tri mais une fonction créée spécialement pour sélectionner les k images du jeu d'apprentissage qui sont les plus proches de l'image que l'on souhaite classiffier.