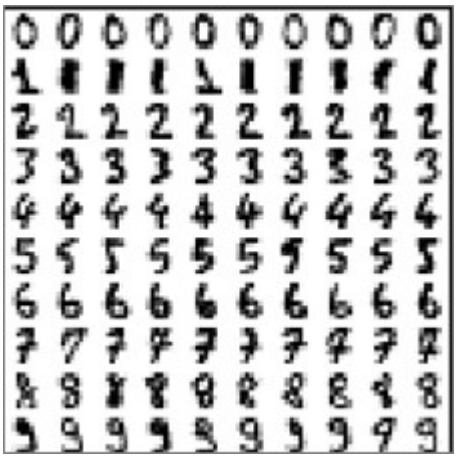
Exercice : Reconnaissance des chiffres

Dans cet exercice, nous allons à nouveau utiliser Scikit-learn qui est une bibliothèque libre Python destinée à l'apprentissage automatique, et notamment le jeu de données `digits` :
nous disposons de 1 797 images de 8 × 8 pixels en niveaux de gris représentant les chiffres de 0 à 9.



Ces données nous sont fournies par l'intermédiaire de 2 tableaux :
• X est une matrice de :
o 1797 lignes pour les images,
o 64 colonnes pour les caractéristiques des images.
Pour tout k ∈ [0, 1796], X[k] est un vecteur de R^64 qui représente l'image digitalisée d'un chiffre.
• Y est un vecteur de taille 1 797.
Pour tout k ∈ [0, 1796], Y[k] correspond à l'étiquette de X[k], ie l'entier compris entre 0 et 9 représenté par X[k].

Le fichier IA_Chiffres_Eleves.py est à votre disposition : il comporte des fonctions permettant d'afficher les chiffres, ainsi que le squelette des fonctions que vous aurez à développer

Entrée[5]:

```
from math import *
from random import *
import  matplotlib.pyplot as plt
import numpy as np
from copy import deepcopy

from sklearn.datasets import load_digits

# Chargement du jeu de données des chiffres
chiffres=load_digits()

X=chiffres.data
Y=chiffres.target # recuperation des classes

# Affichage des chiffres
def afficherChiffres( X, Y, limit_max=10 ):
    """   afficherChiffres( X: list, Y: list, limit_max=10 : int ):
          entrees : X, liste du jeu de données
                  : Y, liste des etiquettes associees
                  : limit_max, entier, nombre de caractères affichés de chaque t
    """
    classes, nombres = np.unique( Y, return_counts = True )
    nombre_max = min( np.max(nombres), limit_max )
    img = np.zeros( ( 100, nombre_max*10 ) )
    for i in range( 10 ) :
        index_classe = np.where(Y == i)[0][:limit_max]
        for j, echantillon in enumerate( index_classe ):
            img[i*10+1:i*10+9,j*10+1:j*10+9] = X[echantillon].reshape((8, 8))
    plt.imshow( img, cmap='binary' )
    plt.xticks([])
    plt.yticks( 5 + 10*np.arange(10), np.arange(10) )
    plt.show()
# tests affichage
# afficherChiffres( X, Y)
# afficherChiffres( X, Y, 14)

def obtenir_echantillons( X , Y, nbEchantillons = 10 ):
    '''Donne une sélection aléatoire d indices d'échantillons pour chaque classe
    index = []
    for classe in np.unique(Y):
        index_classe = np.where(Y == classe)[0]
        replace = len(index_classe) > nbEchantillons
        index += list( np.random.choice(index_classe, size=nbEchantillons, repla
    return index

index = obtenir_echantillons( X, Y )
afficherChiffres( X[index], Y[index] )
plt.show()
```

Apprentissage supervisé

On envisage l'algorithme des k plus proches voisins.
1. Écrire la fonction `distance( im1, im2 )` qui renvoie la distance euclidienne entre les images im1 et im2.

Entrée[6]:

```
def distance( im1 , im2 ):
    d=0
    for i in range( len( im1 ) ):
        d = d + ( im2[i] - im1[i] )**2
    d = np.sqrt( d )
    return d


```

2. Écrire la fonction `PlusProchesVoisins( X, Y, im, k )` qui renvoie la ou les étiquettes les plus présentes parmi les k voisins de im (image de 8 × 8 pixels représentée sous la forme d'une liste).

Entrée[7]:

```
def PlusProchesVoisins( X , Y , im , k ):
    # calcul des distances entre chaque image du jeu de données et l'image im
    lesDistances=[]
    nbClasses = 10
    for i in range( len(X) ):
        d = distance( X[i] , im )
        lesDistances.append( [d,Y[i]] )

    # tri par ordre croissant de distance
    lesDistances.sort()

    # on cherche la classe majoritaire parmi les voisins les plus présents :
    # on compte le nombre de voisins plus proches dans chaque catégorie
    voisins=[0]*nbClasses
    for i in range( k ): #on regarde les k plus proches voisins de im
        voisins[ lesDistances[i][1] ]+=1
    # on cherche la ou les catégories plus représentées
    rep = [] # liste des chiffres les plus représentés
    maxi = 0 # compte le maximum
    for i in range( nbClasses ):
        if voisins[ i ] > maxi:
            maxi = voisins[ i ]
            rep = [ i ]
        elif voisins[ i ]== maxi :
            maxi = voisins[ i ]
            rep.append( i )
    return rep
```

3. Proposer des instructions pour partager les données en 2 groupes : un groupe pour l'apprentissage et l'autre pour le test.

Entrée[8]:

```
# Séparation des données d'apprentissage et de tests
AX,AY=[],[]
TX,TY=[],[]
dA={} # dictionnaire des clés déjà choisies comme donnée d'apprentissage
dT={}
i=0 # compteur
while i<= len( X )-1:
    j=randint( 0, len( X )-1 )
    if j not in dA:
        AX.append( X[j] )
        AY.append( Y[j] )
        dA[j]=1
        i=i+1
    elif j not in dT:
        TX.append( X[j] )
        TY.append( Y[j] )
        dT[ j ]=1
        i=i+1
# apreès separation puis echantillonnage
index = obtenir_echantillons(AX, AY)
#afficherChiffres(AX, AY)
afficherChiffres( X[index], Y[index] )
```

4. Écrire des instructions pour calculer le taux de bonnes prédictions sur les données de tests.

Entrée[9]:

```
# Précision
nb = 0 # compteur des bonnes prédictions
for i in range(len(TX)):
    prediction=PlusProchesVoisins( AX, AY, TX[i], 3)
    if len( prediction ) == 1: # pas d'indécision
        if prediction[0] == TY[i]:
            nb = nb + 1
```

Apprentissage non supervisé

On s'intéresse à nouveau à l'ensemble des images de chiffres de 0 à 9, mais cette fois-ci sans prendre en compte leur étiquette.
Nous allons appliquer l'algorithme des k moyennes avec k = 10, de manière à obtenir un regroupement de ces images en 10 classes, que l'on espère correspondre aux différents chiffres représentés. Pour rappel, ces images sont regroupées dans une liste X. Chaque image X[i] est représentée par un vecteur de R64.
5. Expliquer l'algorithme des k-moyennes.

Entrée[ ]:

```

```

6. Écrire une fonction `barycentre( s )` qui calcule le barycentre d'un ensemble d'images représenté par la liste s. Ce dernier est un vecteur de R64 (il représente lui aussi une image 8 × 8)

Entrée[10]:

```
def barycentre(s):
    bary=[0]*64
    for k in range(len(s)):
        for i in range(64):
            bary[i]=bary[i]+s[k][i]#/len(s)
        bary[i] = bary[i] / len( s )
    return bary

def dist( X1, X2 ):
    d=0
    for i in range(len(X1)):
        d=d+(X1[i]-X2[i])**2
    d= np.sqrt(d)
    return d

```

7. Écrire la fonction `kmoyennes(X,k)` qui prend en argument une liste X d'images et un entier k et renvoie une liste de k classes où sont regroupées les images. Pour cela on choisit aléatoirement k centres initiaux lesCentres. Pour chaque image Xi, on calcule la valeur de lesCentresj la plus proche de Xi pour ranger ce dernier dans la classe. On calcule les barycentres de ces classes. On s'arrête quand ces derniers ne sont pas modifiés.

Entrée[11]:

```
def kmoyennes( X, k ):
    # choix au hasard de k images
    changement = True
    lesCentres=np.array([ X[randint(0,len(X)-1)] for i in range(k)] )

    while changement: # on boucle tant que les centres bougent
        lesClasses=[ [] for i in range(k) ] # k classes
        for i in range(len(X)):
            # recherche du centre le plus proche de Xi
            dmin = np.inf
            jmin = -1
            for j in range(k):
                d = dist(X[i],lesCentres[j])
                if d < dmin:
                    dmin = d
                    jmin = j

            # on ajoute Xi à la classe lesClasses[jmin]
            lesClasses[ jmin ].append( X[i] )

      # calcul des nouveaux barycentres pour chaque classe
        nouveauxCentres=np.array([barycentre(lesClasses[i]) for i in range(k)])
        if nouveauxCentres.all() == lesCentres.all():
            changement = False
        else :
            lesCentres=deepcopy( nouveauxCentres )
    return lesClasses , lesCentres

lesClasses , lesCentres = kmoyennes(X,10)
index=[i for i in range(10)]

afficherChiffres( lesCentres, Y[index] )
plt.show()
```

Entrée[ ]:

```

```