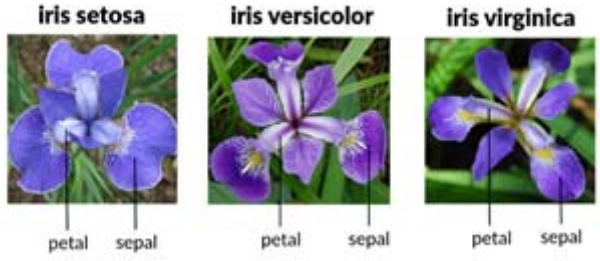
Exercice guidé sur la classification d'iris - Apprentissage automatique non supervisé

Algorithme des k moyennes (k-means)

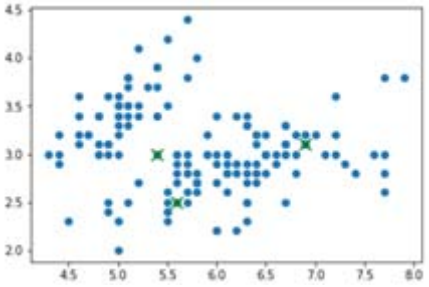
Pour mettre en place une classification à l'aide de l'algorithme des k-moyennes. Nous allons repartir du jeu de données des iris de Fischer. Mais cette fois-ci nous ne tiendrons pas compte de la classe définie dans la liste target...puisque nous cherchons à la définir.



Dans cet exercice, nous ne travaillerons que sur les 2 1ères caractéristiques (sur les 4).

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2

Le script suivant permet de tracer un nuage de points et de déterminer 3 centres initiaux aléatoires :



Analyser le code fourni puis définir et tester les fonctions suivantes :

Entrée[20]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_iris
import random

#Chargement du jeu de données
iris = load_iris()
X= iris.data
X = X[ : , :2 ] # extraction des 2 1eres colonnes de la matrice des iris => uniquement les 2 1eres caracteristiques

# détermination aléatoire des centres parmi l'ensemble des données fournies
k= 3
lesCentres = []
for i in range(k):
    ind = random.randint(0,len(X)-1)
    lesCentres.append(X[ind].tolist())


# affichage de la situation initiale
plt.scatter([x[0] for x in X], [x[1] for x in X]) # affichage des points pour les iris
plt.scatter([x[0] for x in lesCentres], [x[1] for x in lesCentres], marker='x', s=100, c='g') #affichage des centres x ver
plt.title("k-means iris de Fischer")
plt.show()
#plt.closeAll()

"""  Code à utiliser lorsque la fonction kmeans sera codée pour la tester
plt.close('all')
classes = kmeans(X, k, lesCentres)    # test de kmeans
cmap = ListedColormap(['g', 'r', 'b', 'purple'])
centres = calculer_centres( classes, k )

plt.scatter([x[0] for x in X], [x[1] for x in X], c=[plus_proche(x, centres) for x in X], cmap=cmap)
plt.scatter([x[0] for x in centres], [x[1] for x in centres], marker='x', s=100, c=range(k), cmap=cmap)
plt.title("Inertie = " + str(inertie( classes, centres,k )))
plt.show()
"""
```

Sortie[20]:

```
'  Code à utiliser lorsque la fonction kmeans sera codée pour la tester\nplt.close(\'all\')\nclasses = kmeans(X, k, lesCentre
s)    # test de kmeans\ncmap = ListedColormap([\'g\', \'r\', \'b\', \'purple\'])\ncentres = calculer_centres( classes, k )\n\
nplt.scatter([x[0] for x in X], [x[1] for x in X], c=[plus_proche(x, centres) for x in X], cmap=cmap)\nplt.scatter([x[0] for
 x in centres], [x[1] for x in centres], marker=\'x\', s=100, c=range(k), cmap=cmap)\nplt.title("Inertie = " + str(inertie( cl
asses, centres,k )))\nplt.show()\n'
```

1- Définir la fonction `fonction dist(x, y)` renvoyant la distance euclidienne de 2 vecteurs x, y passés en paramètres.

Entrée[21]:

```
def dist( x, y ):
    """ dist(x : list, y : list)-> float
        entrees : x, y, 2 listes (vecteurs) de flottants dont on veut calculer la distance euclidienne
        sortie : s, flottant, la distance entre les  vecteur x et y
    """
    d = 0.
    for i in range( len(x) ):
        d+= (x[i] - y[i])**2
    return d**.5
```

2- Écrire une fonction `plus_proche(x, lesCentres)` renvoyant le numéro i (indice) de la classe la plus proche de x parmi lesCentres, c'est-à-dire la classe telle que la distance de x à lesCentres[i] soit minimale.
Cette fonction fait appel à la fonction `dist` .

Entrée[22]:

```
#Q2
def plus_proche( x , lesCentres ):
    """ plus_proche( x : list , lesCentres : list )-> int
        entrees : x, liste, vecteur dont on veut connaitre le centre le plus proche
                : lesCentres, liste des centres
        sortie : imin, entier, indice du centre le plus proche
    """
    imin = 0
    for i in range(len(lesCentres)):
        if dist( x, lesCentres[i] ) < dist( x, lesCentres[imin] ):
            imin = i
    return imin
```

3- Écrire une fonction `calculer_classes( X ,  lesCentres , k )` renvoyant une liste de classes telle que classes[i] soit la liste des données de X dont le centre le plus proche est lesCentres[i].
Cette fonction fait appel à la fonction `plus_proche` .

Entrée[23]:

```python
#Q3
def calculer_classes( X, lesCentres, k ):
    """ calculer_classes( X:list, centres:list , k:int)->list
        entrees : X, liste, liste des entrées
                : lesCentres, liste des centres
                : k, entier, nombre de classes
        sortie : classes, liste de listes des points de chaque classe
    """
    classes = [ [] for i in range(k) ]
    for x in X:
        indPlusProche = plus_proche(x, lesCentres)  # indice du centre le plus proche
        classes[ indPlusProche ].append(x) # ajout de x à la classe possedant cet indice
    return classes
```

4- Le centre (ou isobarycentre ou centroide) d'un ensemble de vecteurs x1, . . . , xn est défini par le vecteur :

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} xi$$

Écrire une fonction `centre( X )` renvoyant le centre de la liste de vecteurs X, passée en paramètre

Entrée[24]:

```python
#Q4
def centre( X ):
    """ centre( X )
        determination du barycentre d'un ensemble X de vecteurs
        entree : X, liste, liste des donnees
        sortie : c, liste(vecteur) du barycentre de X
    """
    dim=0
    if len(X) != 0:
        dim = len(X[0])
    c = [0.]*dim
    if len(X) == 0 :
        return c
    for x in X:
        for i in range(len(x)): # pour chaque caracteristique
            c[i] = c[i] + x[i]

    for i in range(len(c)):
        c[i] = round( c[i] /len(X) , 3) #arrondi à 3 decimales
    return c

```

5- Écrire une fonction `calculer_centres( classes,k )` renvoyant la liste des centres de chaque classe. Cette fonction fait appel à la fonction

`centre` .

Entrée[25]:

```python
#Q5
def calculer_centres( classes, k ):
    """ calculer_centres(classes : lst , k : int) -> list
        entrees : classes, liste, représentant les classes
                :  k, entier, nombre de classes  à générer
        sortie : centres, liste (de listes) des centres générés
    """
    centres = []
    for classe in classes:
        centrei = centre( classe )
        centres.append( centrei )
    return centres
```

6- Écrire une fonction `kmeans(X, k, lesCentres)` appliquant l'algorithme des k-moyennes à X en partant des centres et renvoyant la liste des classes obtenues.
Cette fonction fait appel à `calculer_classes` et `calculer_centres` .

Entrée[1]:

```
def affiche( X, classes,k):
    cmap = ListedColormap(['g', 'r', 'b', 'purple'])
    centres = calculer_centres( classes, k )
    plt.close()
    plt.scatter([x[0] for x in X], [x[1] for x in X], c=[plus_proche(x, centres) for x in X], cmap=cmap)
    plt.scatter([x[0] for x in centres], [x[1] for x in centres], marker='x', s=100, c=range(k), cmap=cmap)
    plt.title("Inertie = " + str(inertie( classes, centres,k )))
    plt.show()

#Q6
def kmeans(X, k, lesCentres):
    """ kmeans(X, k, lesCentres)->list
        entrees : X, liste, liste des entrées
                : lesCentres, liste des centres
                : k, entier, nombre de classes
        sortie : classes, liste des points de chaque classe apres traitement du kmeans
    """
    centres2 =  []
    while  ( (sorted(lesCentres)) != (sorted(centres2)) ) : #tri sur les listes pour pouvoir utiliser ==
    # while not (  np.array( sorted(lesCentres)==sorted(centres2) ).all() ):
        centres2 = lesCentres
        classes = calculer_classes( X, centres2, k )
        # affiche(X,classes,k)
        # plt.show()
        lesCentres = calculer_centres( classes, k)

    return classes
```

7- Créer une fonction `inertie( classes, centres, k )` qui calcule l'inertie

Entrée[35]:

```
def inertie( classes, centres, k ):
    s = 0.
    for i in range(k):
        for x in classes[i]:
            s += dist(x, centres[i])**2
```

Entrée[36]:

```
#_________________ Test _________________________
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from sklearn.datasets import load_iris
import numpy as np
import random
#Chargement du jeu de données
iris = load_iris()

k= 3
X= iris.data
X = X[ : , :2 ] # extraction des 2 1eres colonnes de la matrice des iris => uniquement les 2 1eres caracteristiques

# détermination aléatoire des centres
lesCentres = []
for i in range(k):
  ind = random.randint(0,len(X)-1)
  lesCentres.append(X[ind].tolist())


# affichage
plt.scatter([x[0] for x in X], [x[1] for x in X]) # affichage des points pour les iris
plt.scatter([x[0] for x in lesCentres], [x[1] for x in lesCentres], marker='x', s=100, c='g') #affichage des centres
plt.show()
 #plt.closeAll()


classes = kmeans(X, k, lesCentres) # test de kmeans
cmap = ListedColormap(['g', 'r', 'b', 'purple'])
centres = calculer_centres( classes, k )
plt.close()
plt.scatter([x[0] for x in X], [x[1] for x in X], c=[plus_proche(x, centres) for x in X], cmap=cmap)
plt.scatter([x[0] for x in centres], [x[1] for x in centres], marker='x', s=100, c=range(k), cmap=cmap)
plt.title("Inertie = " + str(inertie( classes, centres,k )))
plt.show()

```