Algorithme des *k* plus proches voisins

L'algorithme k-nn se base sur un jeu de données comportant plusieurs paramètres en général quantitatifs (qui se mesurent à l'aide d'un nombre), et un paramètre qualitatif : la classe de l'objet.

L'objectif de l'algorithme est de classer un nouvel individu (au sens statistique) dans une des classes de la population existante. Déroulement de l'algorithme :

- Calcul de toutes les distances euclidiennes entre le jeu de données connues et le nouvel individu.
- Tri des éléments du jeu de données par les distances dans l'ordre croissant
- Sélection des k voisins les plus proches : ce sont les k premiers éléments triés à l'étape précédente
- Le nouvel individu appartient à la classe majoritaire dans les k plus proches voisins

Mise en forme des données

Nous allons travailler sur de la reconnaissance de fruits, commençons par explorer les données. Vous les trouverez dans le fichier `donnees_fruits.csv`, à décompresser dans donnees_k_nn
Le programme suivant importe un fichier csv, donne les intitulés des champs, et représente les données. Vérifiez que le chemin d'accès au fichier csv est correct, modifiez-le si nécessaire.

[ ]
```
import csv
from collections import OrderedDict   # structure de données
utilisée

# On importe le fichier d'exemple de base de données de films
donnees_fruits_csv = open('donnees_fruits.csv','r', encoding
='utf-8')
lecteur = csv.DictReader(donnees_fruits_csv, delimiter=';')
fruits = list(lecteur)

print("Identifiants des champs de données et type du champ")
items_fruits =list(fruits[0].items())
data = list(fruits[0].values())
for champ,donnee in items_fruits:
    print("L'identifiant '",champ,"' comporte des données de type
",type(donnee),", de valeur pour le 1er fruit :",
          donnee)

donnees_fruits_csv.close()
```

On retrouve le problème déjà constaté pendant le chapitre sur les données : les champs numériques sont considérés comme du texte.
Reprendre et modifier le programme précédent pour que les champs aient le bon type de données.

```
[ ]
```

Pour la suite, il est nécessaire de savoir quels sont les types de fruits (ananas, kiwis, etc.) présents dans la base de données. Ecrire un programme qui répond à cette question.

```
[ ]
```

Présentation des données

Le programme suivant importe la bibliothèque matplotlib, qui permet de faire assez facilement tout type de graphique scientifique. Il donne ensuite tous les schémas possibles en deux dimensions, avec les 4 champs numériques. Les pommes sont en vert, les citrons en bleu (pour des raisons de lisibilité), les oranges en orange et les mandarines en rouge.
Eventuellement cliquez sur les schémas, pour les voir tous

```
[ ]
```

```python
import matplotlib.pyplot as plt
# import matplotlib.patches as mpatches
from matplotlib.lines import Line2D

#Données
nom = [fruit['nom_fruit'] for fruit in fruits]
code_couleur = []
code_marqueur = []
for i in range(len(nom)):
    if nom[i] == "pomme":
        code_couleur.append('green')
        code_marqueur.append('.')
    elif nom[i] == 'mandarine':
        code_couleur.append('red')
        code_marqueur.append('v')
    elif nom[i] == 'orange':
        code_couleur.append('orange')
        code_marqueur.append('s')
    else:
        code_couleur.append('blue')
        code_marqueur.append('*')
masses =[fruit['masse'] for fruit in fruits]
```

```
largeurs = [fruit['largeur'] for fruit in fruits]
hauteurs = [fruit['hauteur'] for fruit in fruits]
couleurs = [fruit['score_couleur'] for fruit in fruits]

#graphiques
#subplot permet de faire plusieurs graphiques (ici 3 lignes 2
colonnes, on précise la taille du schéma total)
fig, axes= plt.subplots(nrows=3, ncols=2,figsize=(18,18))

#pour la légende, compliquée à faire
legend_elements = [Line2D([0], [0], marker='o', color='w',
label='Pommes', markerfacecolor='g', markersize=10),
                   Line2D([0], [0], marker='o', color='w',
label='Mandarines', markerfacecolor='r', markersize=10),
                   Line2D([0], [0], marker='o', color='w',
label='Oranges', markerfacecolor='orange', markersize=10),
                   Line2D([0], [0], marker='o', color='w',
label='Citrons', markerfacecolor='blue', markersize=10),
                  ]

plt.subplot(321)    # sous-schéma 1 dans les 3 colonnes et 2
lignes (3 - 2 - 1)
plt.scatter(masses, largeurs, c = code_couleur, alpha = 0.5, s =
70)
plt.xlabel('masse')
plt.ylabel('largeur')
plt.legend(handles=legend_elements,loc = 'lower right')

"""
# Autre méthode pour la légende, avec des rectangles
classes = ['pomme', 'mandarine', 'orange', 'citron']
class_colours = ['green','red','orange','yellow']
ronds = []
for i in range(0,len(class_colours)):
    ronds.append(mpatches.Circle((0,0),5, color =
class_colours[i])) # Circle fait des rectangles O_o
plt.legend(ronds,classes)
"""

plt.subplot(322)
plt.scatter(masses, hauteurs, c = code_couleur, alpha = 0.5, s =
70)
plt.xlabel('masse')
plt.ylabel('hauteur')
plt.legend(handles=legend_elements,loc = 'lower right')

plt.subplot(323)
plt.scatter(masses, couleurs, c = code_couleur, alpha = 0.5, s =
70)
plt.xlabel('masse')
plt.ylabel('score couleur')
plt.legend(handles=legend_elements,loc = 'lower right')
```

```
plt.subplot(324)
plt.scatter(largeurs, hauteurs, c = code_couleur, alpha = 0.5, s
= 70)
plt.xlabel('largeur')
plt.ylabel('hauteur')
plt.legend(handles=legend_elements,loc = 'lower right')

plt.subplot(325)
plt.scatter(largeurs, couleurs, c = code_couleur, alpha = 0.5, s
= 70)
plt.xlabel('largeur')
plt.ylabel('score couleur')
plt.legend(handles=legend_elements,loc = 'lower left')

plt.subplot(326)
plt.scatter(hauteurs, couleurs, c = code_couleur, alpha = 0.5, s
= 70)
plt.xlabel('hauteur')
plt.ylabel('score couleur')
plt.legend(handles=legend_elements,loc = 'lower left')


plt.show()
```

Conclusion graphique:

Les schémas précédents montrent, suivant les projections, une classification nette pour les mandarines, assez nette pour les citrons, mais bien moins claire pour les pommes et oranges.

Classification de nouveaux individus

On donne deux individus de classe inconnue, dont on a mesuré les caractéristiques suivantes (masse, largeur, hauteur) :

```
banane  = [100,6.3,8]
```

```
pasteque = [162,7.1,7.3]
```

On voudrait savoir à quelle classe appartiennent ces fruits. Pour cela on va programmer l'algorithme *k*-nn, suivant la méthode exposée en préambule:

- Calcul de toutes les distances euclidiennes entre le jeu de données connues et le nouvel individu.
- Tri des éléments du jeu de données par les distances dans l'ordre croissant
- Sélection des k voisins les plus proches : ce sont les k premiers éléments triés à l'étape précédente. On pourra prendre k = 3 ou k = 5 comme proposé coi-dessous
- Le nouvel individu appartient à la classe majoritaire dans les k plus proches voisins Compléter le programme suivant.

```
[ ]   # Programme k-nn

      from math import sqrt

      def distance(pointA,pointB):
          """
          Renvoie la distance euclidienne entre deux points
          @param pointA :   tuple de flottants
          @param pointB :   tuple de flottants, de même dimension que
      pointA
          @return distAB :  flottant positif ou nul, distance
      euclidienne entre A et B
          """

          return distAB

      def kPlusProches(point,liste,k):
          """
          Renvoie les k plus proches voisins d'un point dans une liste
      de données
          @param point :      tuple de flottants
          @param liste :      liste de tuples, les tuples ont comme
      dimension celle de point + 1. La première
                              valeur de chaque tuple est la classe de
      l'individu, les autres sont les
                              coordonnées (valeurs des paramètres)
          @param k :          entier strictement positif, inférieur ou
      égal au nombre d'éléments de liste
          @return k_voisins : liste des voisins les plus proches de
      "point" dans "liste"
                              , au sens de la distance appelée dans
      cette fonction
          """

          return k_voisins

      def appartient_a(point,liste,k):
          """
          Renvoie la classe d'un individu déterminée par celle de ses k
      plus proches voisins
          @param point :      tuple de flottants
          @param liste :      liste de tuples, les tuples ont comme
      dimension celle de point + 1. La première
                              valeur de chaque tuple est la classe de
      l'individu, les autres sont les
                              coordonnées (valeurs des paramètres)
          @param k :          entier strictement positif, inférieur ou
      égal au nombre d'éléments de liste
          @return classe :    chaine ou code numérique donnant la
      classe de "point". C'est la classe majoritaire dans les
```

```
                            k-voisins les plus proches
    """

    return classe

# quelques petits test avec des assert : s'il y a un message
d'erreur c'est que la fonction est fausse
assert distance((0,0),(1,1)) == sqrt(2)
assert distance((1,1),(1,1)) == 0
assert(distance((1,1),(4,5))) == 5
assert distance ((1,1,1),(2,2,2)) == sqrt(5)   # celui-là est
volontairement faux. Corrigez-le !


banane  = [100,6.3,8]
pasteque = [162,7.1,7.3]
liste_fruits = []
# Extraction des données pertinentes dans la base de données des
fruits.
for fruit in fruits:

 liste_fruits.append((fruit['nom_fruit'],int(fruit['masse']),int(
fruit['largeur']),int(fruit['hauteur'])))


type_de_fruit = appartient_a(banane,liste_fruits,3)
print("le fruit de caractéristiques ",banane," est probablement
un.e ",type_de_fruit)
type_de_fruit = appartient_a(banane,liste_fruits,5)
print("le fruit de caractéristiques ",banane," est probablement
un.e ",type_de_fruit)

type_de_fruit = appartient_a(pasteque,liste_fruits,3)
print("le fruit de caractéristiques ",pasteque," est probablement
un.e ",type_de_fruit)
type_de_fruit = appartient_a(pasteque,liste_fruits,5)
print("le fruit de caractéristiques ",pasteque," est probablement
un.e ",type_de_fruit)
```

Que concluez-vous ?

Interlude : encore des jolis graphes

Ci-dessous un graphe 3D un peu plus sophistiqué, avec les nouveaux individus

```
[ ]  import matplotlib.pyplot as plt
     from mpl_toolkits.mplot3d import Axes3D
```

```
#graphiques
#subplot permet de faire plusieurs graphiques (ici 3 lignes 2
colonnes, on précise la taille du schéma total)
fig = plt.figure(figsize=(9,9))
ax = Axes3D(fig)

#pour la légende, compliquée à faire
legend_elements = [Line2D([0], [0], marker='o', color='w',
label='Pommes', markerfacecolor='g', markersize=10),
                   Line2D([0], [0], marker='*', color='w',
label='Mandarines', markerfacecolor='r', markersize=10),
                   Line2D([0], [0], marker='s', color='w',
label='Oranges', markerfacecolor='orange', markersize=10),
                   Line2D([0], [0], marker='p', color='w',
label='Citrons', markerfacecolor='blue', markersize=10),
                   Line2D([0], [0], marker='v', color='w',
label='Inconnu', markerfacecolor='black', markersize=10),
                  ]

#fig.add_subplot(111, projection = '3d')
for i in range(len(nom)):
    if nom[i] == "pomme":
        ax.scatter(masses[i], largeurs[i], hauteurs[i], c =
"green", s= 70, alpha = 0.5)
    elif nom[i] == 'mandarine':
        ax.scatter(masses[i], largeurs[i], hauteurs[i], c =
"red", s= 70, marker='*', alpha = 0.5)
    elif nom[i] == 'orange':
        ax.scatter(masses[i], largeurs[i], hauteurs[i], c =
"orange", s= 70, marker='s', alpha = 0.5)
    else:
        ax.scatter(masses[i], largeurs[i], hauteurs[i], c =
"blue", s= 70, marker='p', alpha = 0.5)
#ax.scatter(masses, largeurs, hauteurs, c = code_couleur, s= 70)
banane  = [100,6.3,8]
pasteque = [162,7.1,7.3]
ax.scatter(100,6.3,8,c= 'black', marker ="v", s= 110)
ax.scatter(162,7.1,7.3,c = 'black', marker ="v", s = 110)
ax.set_xlabel('masse')
ax.set_ylabel('largeur')
ax.set_zlabel('hauteur')
fig.legend(handles=legend_elements,loc = 'lower right')

plt.show()
```

Améliorations et compléments.

Pondération

Une amélioration classique de l'algorithme k-nn est de pondérer l'importance des voisins. On utilise en général le coefficient 1/d, où d est est la distance entre l'individu et son voisin. Cette amélioration est simple à mettre en oeuvre lorsqu'on ne fait pas de la classification mais de la régression ; c'est-à-dire qu'on essaie non pas de prévoir une classe, mais une valeur numérique (exemple : prévision d'une probabilité de pluie). Avec les classes, c'est un peu plus compliqué : une chaîne de caractères est difficilement divisible par un nombre. Mais vous allez imaginer une solution, ce n'est pas si compliqué que ça. Recopiez et modifiez votre code dans la cellule ci-dessous.

[ ]

```
# Programme k-nn avec pondération

from math import sqrt

def distance(pointA,pointB):
    """
    Renvoie la distance euclidienne entre deux points
    @param pointA :   tuple de flottants
    @param pointB :   tuple de flottants, de même dimension que
pointA
    @return distAB :  flottant positif ou nul, distance
euclidienne entre A et B
    """

    return distAB

def kPlusProches(point,liste,k):
    """
    Renvoie les k plus proches voisins d'un point dans une liste
de données
    @param point :      tuple de flottants
    @param liste :      liste de tuples, les tuples ont comme
dimension celle de point + 1. La première
                        valeur de chaque tuple est la classe de
l'individu, les autres sont les
                        coordonnées (valeurs des paramètres)
    @param k :          entier strictement positif, inférieur ou
égal au nombre d'éléments de liste
    @return k_voisins : liste des voisins les plus proches de
"point" dans "liste"
                        , au sens de la distance appelée dans
cette fonction
    """

    return k_voisins

def appartient_a(point,liste,k):
    """
```

```
    Renvoie la classe d'un individu déterminée par celle de ses k
plus proches voisins
    @param point :      tuple de flottants
    @param liste :      liste de tuples, les tuples ont comme
dimension celle de point + 1. La première
                        valeur de chaque tuple est la classe de
l'individu, les autres sont les
                        coordonnées (valeurs des paramètres)
    @param k :          entier strictement positif, inférieur ou
égal au nombre d'éléments de liste
    @return classe :    chaine ou code numérique donnant la
classe de "point". C'est la classe majoritaire dans les
                        k-voisins les plus proches
    """

    return classe


banane  = [100,6.3,8]
pasteque = [162,7.1,7.3]
liste_fruits = []
# Extraction des données pertinentes dans la base de données des
fruits.
for fruit in fruits:

 liste_fruits.append((fruit['nom_fruit'],int(fruit['masse']),int(
fruit['largeur']),int(fruit['hauteur'])))


type_de_fruit = appartient_a(banane,liste_fruits,3)
print("le fruit de caractéristiques ",banane," est probablement
un.e ",type_de_fruit)
type_de_fruit = appartient_a(banane,liste_fruits,5)
print("le fruit de caractéristiques ",banane," est probablement
un.e ",type_de_fruit)

type_de_fruit = appartient_a(pasteque,liste_fruits,3)
print("le fruit de caractéristiques ",pasteque," est probablement
un.e ",type_de_fruit)
type_de_fruit = appartient_a(pasteque,liste_fruits,5)
print("le fruit de caractéristiques ",pasteque," est probablement
un.e ",type_de_fruit)
```

Distance de Hamming.

L'objectif dans cette partie est de tenter de deviner si un champignon est comestible ou pas.
Le problème est que la base de données "champignons" ne comporte aucune donnée numérique !
Pour calculer les distances, on va utiliser la distance de Hamming. Pour trouver sa valeur, on

compare les "coordonnées" des points une par une. Ici, ce sont des caractères. S'ils sont identiques, on ajoute 0, s'ils sont différents on ajoute 1.
Cette distance peut être utilisée pour comparer deux chaines de caractères, par exemple Hamming("bateau","gateau") = 1

Le fichier comportant les données sur les champignons s'apelle `champignons.csv`. Le fichier `champs_donnes_champis.txt`, (ainsi que ce document "champignons") comporte la codification et les noms des champs. Le premier champ comporte soit "p" pour toxique/poison, soit e pour comestible/edible.

Le programme ci-dessous affiche le premier enregistrement.

[ ]
```
import csv
from collections import OrderedDict   # structure de données utilisée

# On importe le fichier d'exemple de base de données de films
champignons_csv = open('champignons.csv','r', encoding ='utf-8')
lecteur = csv.DictReader(champignons_csv, delimiter=',')
champis = list(lecteur)

print("Identifiants des champs de données et type du champ")
items_champis =list(champis[0].items())
data = list(champis[0].values())
print("Les données du 1er champignon sont :")
for champ,donnee in items_champis:
    print("L'identifiant '",champ,"' comporte la valeur :",donnee)

print()
print("Il y a au total ",len(data)," champs dont un seul pour la classe")
champignons_csv.close()
```

Pour sélectionner des variables pertinents afin d'appliquer la méthode des *k* plus proches voisins, on va se contenter d'une approche graphique.

Consultez ce document ; déduisez-en les variables les plus pertinentes pour discriminer les champignons comestibles de ceux qui sont toxiques. On utilisera entre 4 et 8 variables. Puis, votre voisin vous demandant votre avis, appliquez la méthode des *k* plus proches voisins, avec la distance de Hamming, pour donner un avis éclairé sur les champignons de caratèristiques suivantes:

```
c1 =
['f','s','y','f','a','f','c','n','y','t','b','s','s','y','y','u','y','o'
,'z','w','v','d']
c1_variante =
['f','s','e','f','a','f','c','n','y','t','u','s','s','y','y','u','y','o'
,'z','b','v','d']
c2 =
['x','y','n','f','f','f','c','n','w','t','b','y','s','w','w','u','w','o'
```

```
,'p','w','c','d']
c3 =
['s','s','y','f','s','d','c','n','y','e','e','s','s','y','y','u','y','n'
,'n','y','c','d']
```

Attention : il y a deux champignons délicieux et un très toxique !

Remarques :

- j'ai eu des erreurs étranges avec une autre bibliothèque que csv sur le champ surfSupAnneau et les suivants (les clés ne sont pas reconnues). Si vous avez ce souci, il faudra peut-être créer un fichier csv avec uniquement les colonnes de données que vous avez choisies (sans oublier la classe !).
- cette méthode est fortement déconseillée IRL... consultez un mycologue ou un pharmacien avant de vous faire une poêlée de champignons 🍄🍄☠️☠️☠️🍄🍄

[ ]

```
# Programme k-nn avec distance de Hamming

from math import sqrt

def distance(pointA,pointB):
    """
    Renvoie la distance euclidienne entre deux points
    @param pointA :   tuple de flottants
    @param pointB :   tuple de flottants, de même dimension que
pointA
    @return distAB :  flottant positif ou nul, distance
euclidienne entre A et B
    """

    return distAB

def kPlusProches(point,liste,k):
    """
    Renvoie les k plus proches voisins d'un point dans une liste
de données
    @param point :      tuple de flottants
    @param liste :      liste de tuples, les tuples ont comme
dimension celle de point + 1. La première
                        valeur de chaque tuple est la classe de
l'individu, les autres sont les
                        coordonnées (valeurs des paramètres)
    @param k :          entier strictement positif, inférieur ou
égal au nombre d'éléments de liste
    @return k_voisins : liste des voisins les plus proches de
"point" dans "liste"
                        , au sens de la distance appelée dans
cette fonction
    """

    return k_voisins
```

```
def appartient_a(point,liste,k):
    """
    Renvoie la classe d'un individu déterminée par celle de ses k
plus proches voisins
    @param point :      tuple de flottants
    @param liste :      liste de tuples, les tuples ont comme
dimension celle de point + 1. La première
                        valeur de chaque tuple est la classe de
l'individu, les autres sont les
                        coordonnées (valeurs des paramètres)
    @param k :          entier strictement positif, inférieur ou
égal au nombre d'éléments de liste
    @return classe :    chaine ou code numérique donnant la
classe de "point". C'est la classe majoritaire dans les
                        k-voisins les plus proches
    """

    return classe
```

[**Frederic Mandon**](mailto:frederic.mandon@ac-montpellier.fr), Lycée Jean Jaurès - Saint Clément de Rivière - France (2015-2020)