# TD : Algorithme des K plus proches voisins

Spé NSI - Lycée du parc

Année 2020-2021

**Introduction**

On se propose ici d'écrire un algorithme dérivé de l'algorithme de tri par insertion dont le but est d'obtenir les k plus petits éléments d'une liste. On verra ensuite comment adapter et utiliser cet algorithme pour résoudre un problème de classification.

## I Les k plus petits éléments d'une liste

### Question 1

Compléter la fonction suivante après avoir bien lu sa spécification dans la docstring.

Si vous n'y arrivez pas, vous pouvez ouvrir votre cours au chapitre qui traite des tris et consulter le paragraphe sur le tri par insertion.

```
def insertion_croissante(L, x):
    """
    L est une liste de flottants qui est triée par ordre croissant
    et x est un flottant.
    Ajoute l'élément x à la liste L en l'insérant à sa place.
    """
```

### Question 2

Pour déterminer les k plus petits éléments de la liste L, on procède en deux temps :

On range un à un les k premiers éléments de L dans une nouvelle liste `plus_petits` (ce qui revient à faire un tri par insertion de cette partie de L).

On parcourt le reste de la liste et pour chaque élément x, on le compare avec le plus grand de la liste `plus_petits`. Si x est plus petit, on retire le dernier élément de `plus_petits` et on y insère x à l'aide de `insertion_croissante`.

Compléter le code de la fonction suivante en conséquence.

```
def k_plus_petits(L, k):
    """
    Renvoie la liste des k plus petits éléments de L
    """

    assert type(k) == int and k > 0, "k doit être un entier > 0"
    assert k <= len(L), "La liste ne contient pas assez d'éléments"


    plus_petits = []

    #  Partie à compléter
    #  Indication : il y a deux boucles à écrire

    return plus_petits
```

# II Les k plus proches voisins

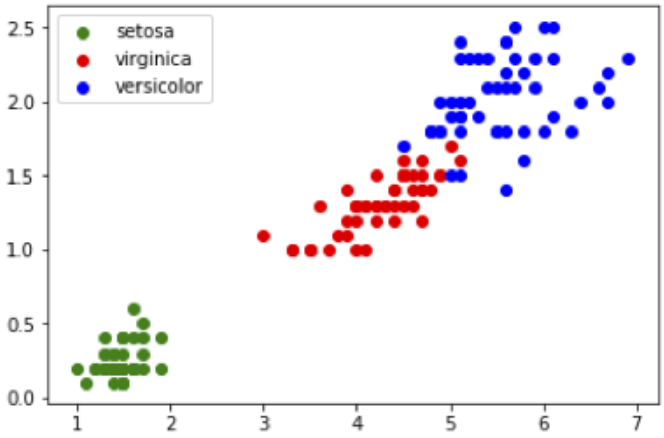
L'algorithme des k plus proches voisins est essentiellement le même que celui qu'on vient d'écrire. Son intérêt réside dans son utilisation pour classifier des données en se basant sur la connaissance des classes pour un grand nombre d'autres données de même type. En cela il se range dans la famille des algorithmes d'apprentissage automatique (machine learning).

Examinons un exemple : Un botaniste a mesuré quatre éléments caractéristiques de 150 fleurs d'Iris dont 50 de chacune de trois espèces différentes (ce sont les classes qui nous intéressent ici). Il a ainsi obtenu un ensemble de 150 enregistrements comportant pour chaque fleur :

— la longueur des sépales (en cm)
— la largeur des sépales (en cm)
— la longueur des pétales (en cm)

— la largeur des pétales (en cm)
— l'espèce d'iris : Iris setosa, Iris virginica ou Iris versicolor

On ne peut pas représenter simmultanément quatre données numériques dans le plan mais, si on se restreint par exemple à la longueur des pétales en abscisse et la largeur en ordonnée, on obtiendrait l'ensemble des points suivants :




Les données sont incluses dans le fichier présent sur le réseau : C'est la définition de la liste `BDIris`

(base de données) comportant 150 éléments qui sont des tuples de 5 éléments sous la forme : `(long_sep, larg_sep, long_pet, larg_pet, esp)`

où les quatre premiers éléments sont des flottants qui correspondent aux mesures et le cinquième est un entier entre 0 et 2 qui code l'espèce (0 pour Iris setosa, 1 pour Iris virginica et 2 pour Iris versicolor)

On va utiliser ces données pour essayer de déterminer à quelle espèce appartient une fleur connaissant les quatre mesures (longueur et largeur des sépales et pétales). On considère une fleur dont on a les mesures mais pas l'espèce (code -1 pour l'espèce) :

```
nouv_fleur = (long_sep, larg_sep, long_pet, larg_pet, -1)
```

### Question 3

Pour mesurer à quel point cette fleur ressemble à chacune des 150 de la base de données, on va utiliser une distance. Parmi les k plus proches de cette fleurs (c'est à dire les k plus petites distances à `nouv_fleur`), on va ensuite déterminer quelle est l'espèce la plus fréquente. La fonction suivante calcule la distance entre deux fleurs.

Compléter la fonction suivante qui calcule la distance entre deux fleurs. n utilisera la distance euclidienne :

$$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2+(t_2-t_1)^2}$$

```
1 from math import sqrt
2 def distance(fleur_1, fleur_2):
3     """Renvoie la distance entre les deux fleurs."""
4
5
6
7
8
```

**Question 4**

Adapter la fonction `insertion_croissante` pour en déduire la fonction `insertion_fleur` en utilisant la fonction `distance` pour comparer les distances de deux fleurs avec `nouv_fleur`.

```
def insertion_fleur(nouv_fleur, L, fleur):
    """
    nouv_fleur est la fleur de référence. L est une liste de fleurs
    qui est triée par ordre croissant des distances à nouv_fleur.
    Ajoute fleur à la liste L en l'insérant à sa place pour la
    distance à nouv_fleur.
    """


```

---
---
---
---
---
---
---
---
---
---
---
---

**Question 5**

Adapter la fonction `k_plus_petits` pour en déduire la fonction `k_NN` (de l'anglais k-nearest neighbors) à l'aide de la fonction `insertion_fleur`. On utilisera la liste `BDIris` comme une variable globale.

```
def k_NN(nouv_fleur, k):
    """
    Renvoie la liste de k fleurs les plus proches de nouv_fleur.
    """

    assert type(k) == int and 1 <= k <= 150,
    "k n'est pas entier entre 1 et 150"

    plus_proches = []

    #
    #  Partie à compléter
    #

    return plus_proches

```

**Question 6**

Lire et compléter les codes des fonctions pour en déduire une fonction qui renvoie le code de l'espèce que la méthode des k plus proches voisins attribue à `nouv_fleur`.

```
def indice_max(L):
    """
    L est une liste non vide d'entiers. Renvoie l'indice de la
    première occurence du maximum de la liste.
    """

    #
    #  Partie à compléter.
    #


def prediction_espece(nouv_fleur, k):
    """
    Renvoie le code de l'espèce que la méthode des k plus proches
    voisins attribue à nouv_fleur d'après les données de BDIris.
    Dans le cas d'égalité, n'importe laquelle des réponses possibles
    est bonne.
    """

    plus_proches = k_NN(nouv_fleur, k)
    compteurs = [0,0,0]  # compteurs[i] va compter les
    # fleurs d'espèce codée par i.

    #
    #  Partie à compléter.
    #


    return indice_max(compteurs)
```