PYTHON. **Apprentissages. Exemple**

On utilise les données disponibles de la bibliothèque *Scikit-Learn* sur les fleurs iris, qu'on importe par :

```
01    from sklearn.datasets import load_iris
02    iris = load_iris()
03    print(iris.target_names)  # renvoie :
array ([ 'setosa' , 'versicolor' , 'virginica'] , dtype = '<U10' )
```

Remarque : U10 désigne les "10-character unicode string", les chaînes de caractère de longueur < 10.

```
04    print(iris.feature_names)   # liste des 4 caractéristiques prises en compte :
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
05    print(iris.data)   # caractéristiques des différents iris ; on obtient :
array ([ [ 5.1 , 3.5 , 1.4 , 0.2 ] , [ 4.9 , 3.  , 1.4 , 0.2 ] , ...  ])
```

Important : `iris.data[i]` est un vecteur de $\mathbb{R}^4$ qui contient les 4 caractéristiques de l'iris d'indice i. Les caractéristiques des iris de la base de données sont donc contenues dans le tableau à deux dimensions `iris.data`.

```
06    print(iris.target) # renvoie array ([ 0 , 0 , ...  , 2 , 2 ])
```

Remarque : `iris.target` donne la variété de l'iris (plus exactement son rang $r \in \{0, 1, 2\}$ dans la liste `iris.target_names`) dans le même ordre que `iris.data`

1) Écrire la fonction `distance(x:[float],y:[float]) -> [float]` qui renvoie la distance entre les deux vecteurs x et y de $\mathbb{R}^n$.

2) *Algorithme des k-plus proches voisins*

Écrire la fonction `kppvoisin(C:[[float]],V:[int],X:[float],k:int)` qui prend en arguments
- C la liste des caractéristiques des iris des données d'apprentissage
- V la liste des variétés des données d'apprentissage (ce sont ainsi les valeurs de référence)
- X les données de l'iris dont on souhaite déterminer la caractéristique
- k le nombre de plus proches voisins à considérer,

et qui renvoie la variété de l'iris (son rang dans la liste `iris.target_names`) en utilisant l'algorithme des k plus proches voisins, c'est-à-dire attribuer la variété (ou l'une des variétés) la plus représentée parmi les k iris les plus proches de X. On suppose connue la méthode `L.sort()` qui trie les éléments de L (pour l'ordre lexicographique).

3) Proposer des instructions permettant de séparer les données disponibles (dans `iris.data` et `iris.target`) en deux parties : 2/3 pour l'apprentissage et 1/3 pour le test.

Afin de pouvoir utiliser la fonction `kppvoisin`, on créera quatre listes :
- une liste `CA` des caractéristiques des données d'apprentissage,

- une liste `VA` des variétés de ces données d'apprentissage,

- et de même deux listes `CT` et `VT` pour les données de test.

On pourra utiliser la procédure `random.shuffle(L)` qui mélange aléatoirement la liste L.

4) Écrire la fonction `matrice_confusion(CA,VA,CT,VT,k)` qui prend en arguments les listes `CA` et `VA` des caractéristiques et variétés des données d'apprentissage, les listes `CT` et `VT` des caractéristiques et variétés des données de test, et k le nombre de plus proches voisins considérés, et qui renvoie la matrice de confusion. La matrice de confusion est une matrice carrée qui mesure la qualité d'un système de classification : le coefficient situé sur la ligne i et la colonne j contient le nombre d'éléments de la classe réelle i qui ont été estimés comme appartenant à la classe j. Plus les termes sur la diagonale sont grands, mieux le système de classification est efficace.

5) On obtient la matrice suivante : $\begin{pmatrix} 20 & 0 & 0 \\ 0 & 12 & 7 \\ 0 & 2 & 9 \end{pmatrix}$. Commenter.

6) Les données numériques sont d'ordres de grandeur différents selon les caractéristiques. Par conséquent, la longueur des sépales a une plus grande importance dans le classement que la largeur des pétales. Pour éviter cela, il faut normaliser les données en les ramenant dans l'intervalle $[0, 1]$.

a) Comment peut-on normaliser les données ? Proposer une expression de la donnée numérique $d'_i \in [0, 1]$ en fonction de la donnée initiale d_i, et des valeurs minimale et maximale $d_{\min}$ et $d_{\max}$ de cette donnée.

b) Proposer une fonction `normaliser` qui prend en argument la liste des données (du formet de `iris.data)`, et renvoie une nouvelle liste (de listes) et dans le même ordre qui contient les données normalisées.

On rappelle que :

- pour récupérer toute la colonne de rang i dans un tableau `T` de type `numpy` à deux dimensions, on peut écrire simplement `T[:,i]`

- pour récupérer toute la ligne de rang i, on peut écrire simplement `T[i,:]` ou même `T[i]`.