TD : RECONNAISSANCE DE CARACTÈRES AVEC KNN - CORRECTION

Les catégories et les symboles sont définis dans le fichier python de cette manière :

```
# Listes de symboles
categories = ["majuscules", "minuscules", "chiffres", "special"]
symboles = [
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
    "abcdefghijklmnopqrstuvwxyz",
    "0123456789",
    ".:,;'(!?)éèàçùêûâ",
]
```

On introduit la fonction suivante :

```
def lire_symbole_fichier(nomFichier: str) -> str:
    car = nomFichier.split('_')
    num = car[2].split('.')[0]
    var = car[1][:len(car[1])-2]
    ind = categories.index(var)
    return symboles[ind][int(num)]
```

1. Indiquer ce que valent les variables `car`, `num`, `var`, `ind` et ce qui est renvoyé par la fonction si `nomFichier="Zurich Light BT_majuscules18_10.png"`.

On obtient :

- **`car = ["Zurich Light BT", "majuscules18", "10.png"]` ;**
- **`num = "10"` (on prend "10.png", puis on coupe au point) ;**
- **`var = "majuscules"` (on retire les 2 derniers caractères de "majuscules18", donc on enlève "18") ;**
- **`ind = categories.index("majuscules")` donc `ind = 0` si `categories = ["majuscules", "minuscules", "chiffres", "special"]` ;**
- **`symboles[ind][int(num)] = symboles[0][10]` ;**
  Si `symboles[0] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"`, alors l'indice 10 (indexation à partir de 0) correspond à la lettre "K". Donc la fonction renvoie "K".

2. Écrire la fonction `lire_donnees_ref(dossier:str,fichiers_car_ref:list)->dict` qui prend en argument le dossier et une liste des noms de fichiers images contenus dans ce dossier `fichiers_car_ref` et qui renvoie le dictionnaire contenant tous les tableaux catégorisés.

Voir fichier « TD1_Correction.py » : Question 2

3. Écrire une fonction `distance(im1:array, im2:array)->float` qui calcule la distance entre les deux images `im1` et `im2` supposées de même dimension.

Voir fichier « TD1_Correction.py » : Question 3

4. Écrire la même fonction `distance_np(im1:array, im2:array)->float` mais en utilisant la fonction `np.linalg.norm()` de la librairie numpy.

Voir fichier « TD1_Correction.py » : Question 4

5. Écrire la fonction `calcul_distances(carac_ref:dict, carac_test:array)->dict` qui prend en argument le dictionnaire des tableaux catégorisés et un tableau associé au caractère à tester et qui renvoie le dictionnaire des distances.

Voir fichier « TD1_Correction.py » : Question 5

6. En se plaçant dans le pire des cas, indiquer le nom d'une méthode de tri performante envisageable, en précisant sa complexité temporelle en fonction de n.

Dans le pire des cas la méthode de tri fusion est performante et a une complexité en O(n ln(n)).

7. Compléter les 3 zones manquantes dans cet algorithme.

```
def Kvoisins(distances:dict, K:int ) -> list :
    voisins = [(float("inf"),"") for k in range (K)]
    for lettre in distances:
        d = distances[lettre]
        for j in range (len(d)):
            if d[j] < voisins[-1][0]:
                k = len(voisins)-1
                while k > 0 and d[j] < voisins[k-1][0]:
                    voisins[k] = voisins[k-1]
                    k=k-1
                voisins[k] = [d[j], lettre]
    return voisins
```

8. Préciser la complexité temporelle asymptotique dans le pire des cas de cet algorithme en fonction de `n` et de `K`. Comparer avec l'utilisation d'un tri classique sachant que `n` est grand et `K` ne dépassera pas 5.

Les deux boucles `lettre` et `j` permettent de balayer les `n` éléments. Le pire des cas est lorsque les distances sont par ordre décroissant « globalement » et pour chaque lettre. Dans ce cas on sera en O(K·n). Comme K est beaucoup plus petit que n, on a une complexité linéaire en n, avec une petite constante. On en déduit que cet algorithme est plus performant qu'un tri classique qui consisterait à trier toutes les distances en O(nlogn).

9. Écrire une fonction `symbole_majoritaire(voisins:list)->str` qui à partir de la liste voisins renvoyée par la fonction `Kvoisins` renvoie le symbole majoritaire.

Voir fichier « TD1_Correction.py » : Question 9

10. Commenter les résultats obtenus.

On remarque que le nombre de voisins ne semble pas influencer la reconnaissance des caractères. Par contre on constate que plus on utilise des images des caractères plus la reconnaissance est correcte.

11. Compléter la fonction Lire_test_mot().

Voir fichier « TD1_Correction.py » : Question 11

12. Écrire la fonction `KNN_test(symboles_numpy, k, base)` qui retrouve le mot codé dans le dictionnaire `symboles_numpy`

Voir fichier « TD1_Correction.py » : Question 12

13. Indiquer ce que voudrait dire M(1, 0) = 1.

Cela signifierait qu'un symbole A aurait été trouvé pour la recherche du symbole B.

14. Écrire la fonction `Matrice_confusion(k,base)` qui retourne la matrice de confusion sur les symboles de la liste `symboles_confusion` pour un KNN avec les k plus proches voisins et en utilisant la base `base` comme source d'apprentissage.

Voir fichier « TD1_Correction.py » : Question 14

15. Visuellement, que pouvez-vous conclure du KNN à partir de la matrice de confusion ?

La matrice n'est pas du tout diagonale pour la base de référence, le KNN est donc très médiocre. Par contre elle est quasiment diagonale avec l'utilisation de la base 11x79.

16. Écrire la fonction `Taux_de_reussite(matrice)` qui calcule le taux de réussite.

Voir fichier « TD1_Correction.py » : Question 16