I. Programmer soi-même une IA avec apprentissage

Connectez-vous sur

`https://capytale2.ac-paris.fr/web/c/186f-1380759`

et téléchargez le code à compléter `TP7etud1.py`

1. Programmer une fonction `distance` prenant en paramètres deux éléments de la liste lecture et renvoyant leur distance euclidienne mutuelle.

```
def distance(elt1, elt2):
    """
    elt1 -- élément de la liste lecture, forme [float, float, type]
    elt2 -- même nature qu'iris1

    renvoie la distance euclidienne entre les deux éléments.
    """
```

2. Tester votre programme avec :

```
print('distance\n',distance([7, 2],[8, 3]))
```

3. Si `L` est une liste de valeurs numériques ; expliquer le fonctionnement du code ci-dessous :

```
M = [(L[i], i) for i in range(len(L))]
M.sort(key= lambda x: x[0])
```

4. On crée ensuite, pour k entier fixé la liste `I` suivante :

```
I = [ M[i][1] for i in range(k)]
```

Cette liste `I` une liste d'indices de k éléments. Les valeurs correspondantes de `L[i][0]` sont-elles croissantes ou décroissantes ?

5. Programmer une fonction `kPlusProchesVoisins` prenant en paramètres un élément `elt` à tester et la valeur de k nombre de plus proches voisins, et renvoie les k plus proches voisins de elt présents dans la liste lecture (variable globale).

```
def kPlusProchesVoisins(elt,k,Listee):
    """
    elt -- liste de points [xi1, xi2] de type [float, float]
    k -- entier naturel, compris entre 1 et le nombre de données dans la liste
    Listee -- liste d'entrainement
    formée de points [xi1, xi2] de type [float, float]

    renvoie la liste des k plus proches voisins
    (pris dans les données du fichier iris.csv) de iris.
    """
    # on crée la liste des distances entre elt et les données du fichier:
    L =.............................................
    # on ajoute la donnée "indice":
    M = [(L[i], i) for i in range(len(L))]
    # on trie M suivant les distances, ordre croissant:
    M.sort(.....................................)
    # On récupère uniquement les indices des k premiers éléments:
    I = [ .................. for i in range(k)]
    # on renvoie la liste des points correspondantes:
    return [Listee[k] for k in I]
```

On pourra utiliser le principe suivant :

– création de la liste des distances de la nouvelle fleur à chacune des fleurs de la base.

– Ajout de l'information "indice" (numéro de ligne dans le fichier base) afin que cette information ne soit pas perdue par un tri.

– Tri de la liste en ordre croissant des distances.

– Récupération des k premiers indices dans la liste ainsi triée.

–récupération des données correspondantes dans le fichier.

6. Tester votre programme avec :

```
L=[[2,3],[1,4],[5,2],[3,3],[6,5]]

print('kPlusProchesVoisins([0, 2],3,L) \n'
      ,kPlusProchesVoisins([0, 2],3,L),'\n')
```

7. Après avoir exécuté la commande

```
L=[[2,3,'type1'],[1,4,'type1'],[5,2,'type1'],[3,3,'type2'],[6,5,'type2']]

print('etiquette_maj(L)=',etiquette_maj(L),'\n')
```

expliquer le comportement de la fonction

```
etiquette_maj(V)
```

8. A l'aide des fonctions etiquette_maj et kPlusProcheVoisins, programmer une fonction `prediction` prenant en paramètres un élément `elt` à tester et la valeur de k nombre de plus proches voisins, et renvoie une prévision de type au vu de ses plus proches voisins.

```
def prediction(elt, k):
    """
    elmt -- liste [x,y] de type [float, float]
    k -- nombre de voisins entier naturel

    renvoie le type prévu au vu de ses k voisins
    """
```

9. Tester votre programme avec :

```
print("prédiction pour knn\n",prediction([0, 2], 3,L))
```

10. Utiliser le code fourni en ligne pour représenter le graphe correspondant.

```
print(affiche([5.9,2.9],5,lecture))
```

11. Exécutez les commandes suivantes :

```
k=3
print('k=',k)
print("\nprédiction pour knn :",prevision([5.9,2.9],k,lecture))
k=5
print('k=',k)
print("\nprédiction pour knn :",prevision([5.9,2.9],k,lecture))
```

Expliquez le résultat observé.

12. On va maintenant créer la matrice de confusion, en utilisant la moitié de la liste pour la prévision et l'autre moitié pour la tester ! Utilisez le code mis à disposition pour le compléter.

```
def confusion(Ltest, Ltrain, k, m):
    """
    confusion renvoie ..........................................................
    """
    total_erreur = 0
    # total_erreur représente ....................................
    matrice_confusion = [[0 for _ in range(m)]for _ in range(m)]
    # la matrice est de taille ...................................
    for i in range(len(Ltest)):
        e_predit=prediction([Ltest[i][0],Ltest[i][1]], k, Ltrain)
        e_correct = Ltest[i][2]
        # on met à jour la valeur compte-tenu des valeurs correcte et prédite pour Ltest[i]
        matrice_confusion[...........][...........] += 1
        if e_predit != e_correct:
            # dans ce cas ......................
            total_erreur += 1
    r=total_erreur/len(Ltest)
    C=matrice_confusion
    return (r,C)
```

```
# exemple:
print('confusion(L2test, L2train, 5, 3) \n'
      ,confusion(L2test, L2train, 5, 3),'\n')
```

13. Que représentent les coefficients diagonaux de la matrice de confusion ?
    Que représentent les coefficients non-diagonaux de la matrice de confusion ?
14. Fermez votre fichier`TP7etud3.py` puis ouvrez le second `TP7fin.py`.
    Expliquez le graphe observé.