kppv Exo BAC ecrit

Thème 5 : Algorithmique

17 Algorihtme des k-plus proches voisins : Application : classification des Iris.

On souhaite rechercher dans un tableau les k plus proches voisins d'un objet donné.

On dispose pour cela d'un tableau t non vide contenant des objets d'un même type et d'une fonction `distance` qui renvoie la distance entre deux objets quelconques de ce type.
Etant donné in objet `cible` du même type que ceux du tableau t, on cherche à déterminer les indices des k éléments du tableau t qui sont les plus proches de cet objet (c'est-à-dire ceux dont la distance à l'objet `cible` est la plus petite).

0.0.1. Question 1 :

On suppose dans cette question que k=1.
La fonction `plus_proche_voisin(t,cible)` ci-dessous prend en argument le tableau `t` et l'objet `cible`. Ecrire sur votre copie uniquement le bloc d'instructions manquant pour que la fonction renvoie l'indice d'un plus proche voisin de `cible`.

```
def plus_proche_voisins(t,cible):
    dmin=distance(t[0],cible)
    idx_ppv=0
    n=len(t)
    for idx in range(1,n):
        ...
```

Processing math: 100%

```
        ...
        ...
    return idx_ppv
```

0.0.1. Question 2 :

On considère le coût en temps du bloc manquant est constant. Quelle est la complexité de la fonction `plus_proche_voisin` quand k = 1 ?

Dans la suite, on suppose que $k \geq 1$.

0.0.1. Question 3 :

Une approche naïve consiste à parcourir le tableau t pour trouver l'indice de l'élément le plus proche de `cible`, puis à recommencer pour trouver l'indice du deuxième élément le plus proche de `cible`, et ainsi de suite. Cela implique de parcourir k fois tout le tableau.
Afin de réduire le nombre d'appels à la fonction `distance`, la stratégie suivante permet de ne parcourir le tableau t qu'une seule fois. Lors de ce parcours, on stocke dans une liste `kppv`, initialement vide, les tupples `(idx,d)` où `idx` est l'indice d'un k plus proche voisin de `cible` déja rencontré et `d` la distance correspondante, triès dans l'ordre décroissant de leur distance à `cible`.
La fonction `recherche_kppv(t,k,cible)` ci-aprés renvoie ainsi la liste des tupples `(idx,d)` où `idx` est l'indice d'un k plus proche voisin de `cible` dans le tableau `t` et `d` la distance correspondante.

On admet que la fonction `insertion(kppv,idx,d)` insére le tupple `(idx,d)` dans la liste `kppv` de sorte que celle-ci demeure triée dans l'ordre décroissant des distances.

```
def recherche_kppv(t,k,cible):
    kppv=[]
    n=len(t)
    for idx in range(n):
        obj=t[idx]
        if len(kppv) < k:
            insertion(kppv,idx,distance(obj,cible))
        else:
            i0,d0=kppv[0]
            if distance(obj,cible)< d0:
                kppv.pop(0)  #supprime le 1er élèment de kppv
                insertion(kppv,idx,distance(obj,cicle))
    return kppv
```

Processing math: 100%

a On remarque qu'il y a plusisuers appels identiques à la fonction `distance(obj,cible)`. Comment ne faire qu'un seul appel à cette fonction ?
b Expliquer l'intérêt de maintenir la liste `kppv` triée.
c Ecrire une fonction `insertion(kppv,idx,d)` qui insére le tupple `(idx,d)` dans la liste `kppv` préalablement triée en préservant l'ordre décroissant selon l'élèment `d`.

On pourra éventuellement utiliser la méthode `insert` dont la documentation, fournie par la commande `help(list.insert)`, est la suivante :

```
insert(self, index, object, /)
    Insère l'object avant la position index dans l'objet appelant référencé par
self.
```

Exemple d'utilisation :

```
>>>liste=[4,2,8,9]
>>>liste.insert(1,3)
>>>liste
[4,3,2,8,9]
```

```
In [1]: help(list.insert)
```

```
Help on method_descriptor:

insert(self, index, object, /)
    Insert object before index.
```

Processing math: 100%