TD Informatique TSI2

Chap 5. Machine_Learning_Part1

Solution

EXERCICE 01

Trier des points. On dispose d'un ensemble de points dans l'espace muni d'un repère orthonormé d'origine O. Un point possède des coordonnées reprsentées par une liste du type `[x,y,z]`. L'objectif est de trier ces points en fonction de leur distance (au carré) à un point donné P, de la plus petite à la plus grande.

1. Écrivons une fonction `distance` qui prend en argument deux listes représentant les coordonnées de deux points quelconques et renvoie le carré de la distance euclidienne entre ces deux points.

```
def distance(p1,p2):
    return (p2[0]-p1[0])**2+(p2[1]-p1[1])**2+(p2[2]-p1[2])**2
distance([1,1,-1],[2,0,0])
    3
```

2. Écrivons une fonction `compare` qui prend en paramètres trois listes `p,p1,p2` représentant dans l'ordre le point P et deux points quelconques P_1, P_2 et qui renvoie -1 si P_1 est plus proche de P que P_2, 1 si P_2 est plus proche de P que P_1 et 0 si les deux points sont équidistants de P.

```
def compare(p,p1,p2):
       d1 = distance(p1,p)
       d2 = distance(p2,p)
       if d1 < d2 :
            return -1
       elif d2 < d1 :
            return 1
       else :
            return 0
```

On pose :

```
liste = [[1, 1, -1], [2, 0, 0], [1, -1, 1], [2, 1, -1]]
P = [0,0,0]
```

Calculons `compare(P,liste[i],liste[j])` pour tout i et j possibles avec $i < j$.

```
P = [0,0,0]
compare(P,liste[0],liste[1])
   -1
compare(P,liste[0],liste[2])
    0
compare(P,liste[0],liste[3])
    -1
compare(P,liste[1],liste[2])
    1
compare(P,liste[1],liste[3])
   -1
compare(P,liste[2],liste[3])
   -1
```

Donc si l'on pose : `d_i = distance(P,liste[i])` alors

$$d_0 < d_1,\ d_0 = d_2,\ d_0 < d_3,\ d_2 < d_1 \text{ et } d_1 < d_3.$$

3. On considère la procédure suivante qui a pour argument un point `p` et une liste nommée `new_liste` composée de listes de trois points, qui applique compare à `p` et à deux points quelconques de `new_liste` et qui renvoie la liste `new_liste` avec les points triés par ordre croissant des distances entre ces points et le point `p`.

```
def tri_points(p,new_liste):
    for i in range(len(new_liste)-1) :
        i_mini = i
        mini = new_liste[i]
        for j in range(i+1, len(new_liste)):
            if compare(p,mini, new_liste[j]) == 1:
                i_mini = j
                mini = new_liste[j]
        new_liste[i], new_liste[i_mini] = new_liste[i_mini], new_liste[i]
    return new_liste
```

On l'applique à `tri_points(P,liste)` de la question **2.**

```
tri_points(P,liste)
    [[1, 1, -1], [1, -1, 1], [2, 0, 0], [2, 1, -1]]
```

C'est logique car $d_0 < d_2 < d_1 < d_3$.

EXERCICE 02

Q1 Tapons les instructions suivantes.

```
In [1]: example_ens=[1,6,3,4,5,0,2,-2,-7,45]
In [2]: d={elt:False for elt in example_ens}
In [3]: d
Out[3]:
{1: False,
 6: False,
 3: False,
 4: False,
 5: False,
 0: False,
 2: False,
 -2: False,
 -7: False,
 45: False}
```

Au début, on a créé un dictionnaire `d` dont les clés sont les éléments de `example_ens` et les valeurs de toutes les clés sont `False`

```
In [4]: d[example_ens[3]]="BlaBla"
In [5]: d
Out[5]:
{1: False,
 6: False,
 3: False,
 4: 'BlaBla',
 5: False,
 0: False,
 2: False,
 -2: False,
 -7: False,
 45: False}
```

On a mis dans la clé `example_ens[3]` qui est 4 la valeur `BlaBla`

```
In [6]: from random import randrange;i=randrange(4,8)
In [7]: i
Out[7]: 4
In [8]: d[example_ens[i]]=2
In [9]: d
Out[9]:
{1: False,
 6: False,
 3: False,
 4: 'BlaBla',
 5: 2,
 0: False,
 2: False,
 -2: False,
 -7: False,
 45: False}
```

On a pris aléatoirement une valeur entre 4 et 7 compris, ici c'est 4.
Puis dans la clé `example_ens[i]` qui est 5, on a mis la valeur 2 et donc le nouveau dictionnaire est celui affiché.

2. On considère maintenant la procédure suivante (j'ai rajouté des `print`)

```
In [14]: def distribute_incomplete(ens,k):
    ...:     n = len(ens)
    ...:     d = {elt: False for elt in ens}
    ...:     d[ens[0]] = 1
    ...:     for p in range(1,k):
    ...:         print('p vaut',p);i = randrange(p,n)
    ...:         print('i vaut',i); d[ens[i]] = p+1
    ...:         print('d vaut a ce niveau',d)
    ...:         ens[i],ens[p] = ens[p],ens[i]
    ...:         print('ens vaut a ce niveau', ens)
    ...:     return d
    ...:
```

```
# On obtient alors :
In [16]: distribute_incomplete(example_ens,4)
p vaut 1
i vaut 4
d vaut a ce niveau {1: 1, 6: False, 3: False, 4: False, 5: 2, 0: False, 2:
    False, -2: False, -7: False, 45: False}
ens vaut a ce niveau [1, 5, 3, 4, 6, 0, 2, -2, -7, 45]
p vaut 2
i vaut 2
d vaut a ce niveau {1: 1, 6: False, 3: 3, 4: False, 5: 2, 0: False, 2:
    False, -2: False, -7: False, 45: False}
ens vaut a ce niveau [1, 5, 3, 4, 6, 0, 2, -2, -7, 45]
p vaut 3
i vaut 6
d vaut a ce niveau {1: 1, 6: False, 3: 3, 4: False, 5: 2, 0: False, 2: 4,
    -2: False, -7: False, 45: False}
ens vaut a ce niveau [1, 5, 3, 2, 6, 0, 4, -2, -7, 45]
Out[16]:
{1: 1,
 6: False,
 3: 3,
 4: False,
 5: 2,
 0: False,
 2: 4,
 -2: False,
 -7: False,
 45: False}
```

Expliquons alors à la main ce que la machine a fait presque spontanément !

Au départ, on a `example_ens=[1,6,3,4,5,0,2,-2,-7,45]`
Nommons `ens` cette liste pour simplifier.

Ici `n=10` pour commencer `d` est un ensemble composé de dix `False`

Alors `d[ens[0]]`=1 et on commence la boucle avec `k=4` et donc les `p` vont de 1 à 3.

On prend `p=1` et `i=randrange(1,10)` et `d[ens[i]]=2` On voit dans le `d` retourné à la fin que `ens[i]=5` et donc `i=4` a été choisi. Enfin, on permute `ens[4]` et `ens[1]` soit 5 et 6 dans `ens`

On prend ensuite `p=2` et `i=randrange(2,10)` et `d[ens[i]]=3` On voit dans le `d` retourné à la fin que `ens[i]=3` et donc `i=2` a été choisi. Enfin, on permute `ens[2]` et `ens[2]` dans `ens` ce qui ne fait rien.

On prend `p=3` et `i=randrange(3,10)` et `d[ens[i]]=4` On voit dans le `d` retourné à la fin que `ens[i]=2` et donc `i=6` a été choisi. Enfin, on permute `ens[6]` et `ens[3]` soit 2 et 4 dans `ens`

On obtient bien à la fin `[1, 5, 3, 2, 6, 0, 4, -2, -7, 45]`

```
In [12]: example_ens
Out[12]: [1, 5, 3, 2, 6, 0, 4, -2, -7, 45]
```

Warning : comme le résultat dépend de `randrange` vous avez dû obtenir autre chose si vous avez tapé le code.

3. Écrivons alors une fonction `distribute` qui prend en paramètres un ensemble donné `ens` sous la forme d'une liste et un nombre entier `k` non nul qui reprend `distribute_incomplete` en y ajoutant une boucle `for elt in ens[k:n]:` dans laquelle on mettra `d[elt] = randrange(1,k+1)` qui règle le problème des éments de `d` restés `False`

Tapons ensuite `distribute(example_ens,2)` et `distribute(example_ens,5)`.
Que fait donc `distribute` ?

```
In [17]: example_ens=[1,6,3,4,5,0,2,-2,-7,45]
In [18]: def distribute(ens,k):
    ...:     n = len(ens)
    ...:     d = {elt: False for elt in ens}
    ...:     d[ens[0]] = 1
    ...:     for p in range(1,k):
    ...:         i = randrange(p,n)
    ...:         d[ens[i]] = p+1
    ...:         ens[i],ens[p] = ens[p],ens[i]
    ...:     for elt in ens[k:n]:
    ...:         d[elt] = randrange(1,k+1)
    ...:     return d
    ...:
```

```
In [19]: distribute(example_ens,2)
Out[19]: {1: 1, 6: 2, 3: 1, 4: 1, 5: 2, 0: 2, 2: 2, -2: 2, -7: 1, 45: 1}
In [20]: example_ens=[1,6,3,4,5,0,2,-2,-7,45]
In [21]: distribute(example_ens,5)
Out[21]: {1: 1, 6: 4, 3: 3, 4: 2, 5: 5, 0: 4, 2: 3, -2: 3, -7: 3, 45: 5}
```

`distribute` génère aléatoirement une partition en k parties de l'ensemble contenant au moins k éléments.

4. Écrivons une fonction `unterteilen` qui prend en arguments un ensemble donné `ens` sous la forme d'une liste et un entier non nul `k` et qui renvoie la partition (nommée `parties` dans la procédure) en `k` parties de l'ensemble `ens` (contenant au moins `k` éléments) générée aléatoirement à l'aide de la fonction `distribute`.
Indication : on commencera par affecter à la variable `d` le dictionnaire `distribute(ens, k)`.
Puis on rentre dans `parties` une liste remplie de `k` fois la liste vide `[]`.
Puis on fera une boucle `for elt in d:` et dans cette boucle, on affectera chaque `elt` dans la liste de `parties` qui est en position `d[elt] - 1`.

```
In [23]: def unterteilen(ens,k):
    ...:     d = distribute(ens,k)
    ...:     parties = [[] for i in range(k)]
    ...:     for elt in d :
    ...:         parties[d[elt]-1].append(elt)
    ...:     return parties
    ...:
```

5. Appliquer `unterteilen` à `examples_ens` avec `k=3`, `k=6` et `k=1`.

```
In [24]:  example_ens=[1,6,3,4,5,0,2,-2,-7,45]

In [25]: unterteilen(example_ens,3)
Out[25]: [[1, 0, -7, 45], [6, 4, -2], [3, 5, 2]]

In [26]: example_ens=[1,6,3,4,5,0,2,-2,-7,45]
In [27]: unterteilen(example_ens,6)
Out[27]: [[1, 2], [4, -2], [3, -7], [0], [45], [6, 5]]

In [28]: example_ens=[1,6,3,4,5,0,2,-2,-7,45]
In [29]: unterteilen(example_ens,1)
Out[29]: [[1, 6, 3, 4, 5, 0, 2, -2, -7, 45]]
```