Jules Hecquet

1PSI1

DS d'info n°9

1) Select id from MEDICAL
where etat = 'hernie discale'

2) Select nom, prenom from PATIENT, MEDICAL
where patient.id = medical.id
and etat = 'spondylolisthésis'

3) Select etat, count(*) as nombre_de_patient from MEDICAL
group by etat.

4) Le type array permet un accès plus facile et rapide aux données du tableau et prends moins de place en mémoire.

5) Le tableau contient N×n valeurs, codées sur 32 bits
Le vecteur contient N valeurs, codées sur 8 bits
Donc la mémoire nécessaire pour stocker le tableau est de :
$100\,000 \times 6 \times 32 + 100\,000 \times 8$ bits
$= 100\,000 \times 6 \times 4 + 100\,000$ octets
$= 0{,}1 \times 6 \times 4 + 0{,}1$ Mo
$= 2{,}$

6) def repartition_par_Groupe(data, etat):
g0, g1, g2 = [], [], []
n = len(etat)
for i in range(n):

```
if etat[i] = 0 :
    g0.append ( data [i, :])
if etat[i] = 1 :
    g1.append ( data [i, :])
if etat [i] = 2 :
    g2.append ( data [i, :])

return ( g0, g1, g2 )
```

7) ARGS 1 = n, n, (xi + j + 1)

ARGS 2 = groupe[h, :, i], groupes [h, :, j],
number = nombre [h]

ARGS 3 = data [:, i]

TEST = « i <> j

8) Les diagrammes de la diagonale permettent d'étudier la répartition des différentes valeurs des attributs dans l'échantillon de patient.

Les diagrammes en dehors de la diagonale permettent d'observer la présence ou l'absence de corrélation entre deux attributs en fonction de l'état de santé.

9) $x_{norm\,j} = \dfrac{x_j - r_{in}(X)}{r_{ex}(X) - r_{in}(X)}$

10)
```
def min_max(X):
    min, max = X[0], X[0]
    n = len(X)
    for i in range(1, n):
        if X[i] < min:
            min = X[i]
        if X[i] > max:
            max = X[i]
    return (min, max)
```

11)
```
def distance(z, data):
    n = len(z)
    l = len(data[:, 0])
    distances = []
    for i in range(l):
        s = 0
        for k in range(n):
            s += (data[i, k] - z[k]) ** 2
        distances.append(s ** 0,5)
    return(distances)
```

12) La partie 1 crée la liste T, qui contient les couples (distance z, ligne i ; i) et la trie en place ~~selon~~ par distance croissante. dist est la liste des distances entre z et les lignes de data.

La partie 2 crée la liste select, qui ~~conte~~ contient le nombre de patient pour chaque état, et la remplis

il ~~ne les état~~ en prenant en compte les états des K patients de plus proches de z.

La troisième partie cherche ~~et renvoi~~ dans select et renvoie l'état le plus représenté (parmis les K patients les plus proches de z)

res est ~~l'é~~ le nombre de patients ~~le plus~~ de l'état le plus représenté, et ind est cet état.

13) La somme des valeurs sur la diagonale de la matrice est le nombre de bon diagnostics faits par l'IA.

En lisant la 1° ligne, on voit que sur les 34 patients en état 0, 23 ont été diagnostiqués en état 0 par l'IA, 4 en état 1 et 7 en état 2

En lisant la 1ère colonne, on voit que sur les 35 patients que l'IA a dignostiqué avec l'état 0, 23 avaient effectivement l'état 0, 7 avaient l'état 1, et 5 l'état 2

Cette matrice sert donc à voir si l'IA est juste, ~~et si deux états sont souvent c~~
et si les mauvais diagnostics sont aléatoires, ou plus fréquents avec certains états.

14) On voit que l'incidence du nombre de voisins est de 5% en plus, que le nombre optimal est environ 10 et que, pour cette valeur, l'algorithme fait environ $\frac{3}{4}$ de bons diagnostiques.

15)
```
def moyenne (sc):
    n = len (sc)
    s = 0
    for i in sc :
        s += i / n
    return (s)

def variance (sc)
    n = len ( sc )
    v = 0
    mu = moyenne (sc)
    for i in sc :
        v += (( i - mu)**2 )/ n
    return (v)
```

16)
```
def synthèse ( data , etat):
    groupes = separationEnGroupe ( data, etat)
    res = []
    n = len ( data[0])
    for e in range (3) :
        t = []
        for i in range (n):
            v = variance ( groupes [e, : , i])
            mu = moyenne ( groupes [e, : , i])

            t . append ( [mu, v])
        res . append ( t)
    return ( res )
```

```
from math import exp, pi

17) def gaussienne(a, moy, v):
        return ((exp**(-(a-moy)**2/(2*v)))/((2*pi*v)**0,5))

18) def probabilite_Groupe(z, data, etat):
    synt = synthese(data, etat)
    groupes = separation_Groupe(data, etat)
    n = len(data[0])
    l = len(data[:,0])
    p = [(len(groupes[i,:,0])/l) for i in range(3)]

    ons = []
    for e in range(3):
        s = 0
        for k in range(n):
            mu, v = synt[e,k,0], synt[e,k,1]
            s = gaussienne(z.[k], mu, v) x s

        ons.append(p.[e] x s)

    return(ons)

19) def prediction(z, data, etat):
    prob = probabilite_Groupes(z, data, etat)
    c = 0
    v = prob.[0]
    for i in range(1:3):
        if prob.[i] > v:
```

```
V = Prob.[i]
c = i

return (c)
```

2°) L'utilisation du logarithme permet de calculer la probabilité finale avec ~~des~~ une somme plutôt qu'un produit, ce qui permet de réduire les erreurs de calcul dues aux arrondis, et de faciliter les expressions.