MAILLARD
Gilles
MPSI 2

13/06/2020

DS d'info n°3

1) SELECT idpatient from MEDICAL where etat='Hernie discale';

✓

2) SELECT nom, prenom from PATIENT join MEDICAL on PATIENT.id = MEDICAL.idpatient where etat = 'spondylolisthésis';

✓

3) SELECT etat, count(idpatient) from PATIENT group by etat;

Distinct

4) Quand les tableaux sont de grande taille, la bibliothèque de calcul numérique Numpy permet d'aller plus vite.

✓

5) Le tableau a N lignes et n colonnes.

Ici N = 100 000 et n = 6. Il y a donc 600 000 cases.

Chaque case est codée sur 32 bits = 4 octets.

On a donc 4(600 000) = 2 400 000 octets

Il faut donc 2,4 Mo pour le tableau data.

✓

6) def separationParGroupe(data, etat):

```
    N = len(data)
    O = []
    X = []
    E = []
    for k in range (N):
        if etat[k] = 0 :
            O.append ( data[k,:])
        elif etat[k] = 1 :
            X.append ( data[k,:])
        else :
            E.append ( data[k,:])
    return [O,X,E]
```

7) ARGS1 = n, n, i*n - ~~n~~ + j

ARGS2 = groupes[i], groupes[j], masken = mask[n*i + j]

ARGS3 = groupes[j]

TEST = i != j

Doit dépendre de k

8) Les diagrammes diagonaux permettent d'avoir un récapitulatif du nombre de patients en fonction des valeurs d'un attributs. Cela permet de voir où se situe la majorité des personnes. V

Les diagrammes non diagonaux permettent de regarder le lien entre deux attributs. Cela peut permettre d'exprimer la cause d'un attribut. V

9) $x_{norm\,j} = \frac{x_j - \min(x)}{\max(x) - \min(x)}$ V

10) def min_max(x) :

return min(x), max(x)

Sans les fonctions min et max:
def min_max(x):

n = len(x)

max = x[0]
min = x[0]
for i in range (1, n):

if x[i] > max:

max = x[i]

if x[i] < min:

min = x[i]

return min, max

OK

Complexité ?

11) def distance (z, data):

min, max = min_max (data [:, z])

d_z =

12) Partie 1 : On crée une liste T avec la distance entre le n-uplet à classer et le connu et sa valeur d'état. Puis on la trie par ordre croissant des distances.

Partie 2 : On crée une matrice ligne nulle de taille nb On lui rajoute 1 et la valeur de etat pour tous les entiers entre 0 et k-1.

Partie 3 : on cherche la plus grande valeur d'indice pour lequel sa valeur correspondante dans select pour l'indice est la plus grande.

T : liste avec la distance entre le n-uplet à classer et celle avec le n-uplet connu ainsi que sa valeur d'état.

dist : les distances euclidiennes entre le n-uplet z et chaque n-uplet de data

select : matrice ligne qui contient la valeur de etat T[i][1]

plus 1.

ind: indice pour lequel la valeur avec cet indice dans select est la plus grande.

15)
```
def moyenne(x):
    S=0
    for k in range(len(x)):
        S=S+x[k]
    return S/len(x)

def variance(x):
    S=0
    u=moyenne(x)
    for k in range(len(x)):
        S=S+x[k]-u**2
    return S/len(x)
```

16)
```
def synthese(data, etat):
    X0=[]=X1=X2
    for k in range(6):
        for i in range(len(data)):
            if etat[i]=0:
                X0.append(data[i])
            if etat[i]=1:
                X1.append(data[i])
            elif:
                X2.append(data[i])
    A=[moyenne(X0), variance(X0)]
    B=[moyenne(X1), variance(X1)]
    C=[moyenne(X2), variance(X2)]
    X0=X1=X2=[]
    L.append[A,B,C]
    return L
```

17)
```
import numpy as np
def gaussienne(a, moy, v):
    e = np.exp(-(a - moy)**2 / 2 / v)
    return e / (np.sqrt(2 * np.pi * v))
```

18)
```
def probabiliteGroupe(z, data, etat):
    L = synthese(data, etat)
    P = 1 = Q = R
    for i in range(6):
        a, b = L[i, 0]
        P = gaussienne(data[i:], a, b) * P
        c, d = L[i, 1]
        Q = Q * gaussienne(data[i, :], c, d)
        e, f = L[i, 2]
        R = R * gaussienne(data[i, :], e, f)
    return [P, Q, R]
```

19)
```
def prediction(z, data, etat):
    P0, P1, P2 = probabiliteGroupe(z, data, etat)
    if P1 > P0 :
        if P2 > P1 :
            return 2
        else :
            return 1
    if P2 > P0 :
        return 2
    else :
        return 0
```

20) Les valeurs de probabilité trouvées sont très petites. L'ordinateur peut avoir du mal à les différencier. En prenant le logarithme décimal on se ramène à des valeurs que peut calculer l'ordinateur sans erreur. Par exemple,

$P0 = 1,1 \times 10^{-15}$ $\log P0 \simeq -15$

$P1 = 7,4 \times 10^{-16}$ $\log P1 \simeq -16$