

LURI VÑÓ

Jorge

18/06/2020

MPSi 2

Informatique

2. Analyse de données

Q1)

```
SELECT idpatient  
FROM MEDICAL  
WHERE etat = "hernie discale"  
;
```

V

Q2) SELECT PATIENT.mon, PATIENT.prenom
FROM PATIENT, MEDICAL
WHERE PATIENT.id = MEDICAL.idpatient
AND etat = "spondylolisthésis"
;

Distinct

Q3) SELECT etat, COUNT(*) as nb_patients
FROM MEDICAL
GROUP BY etat
;

Q4) Comme les réels sont stockés en 32 bits et les entiers en 8 bits, l'intérêt principal de l'utilisation de numpy est de réduire le poids de la base de données, ce qui permet une plus rapide exécution et une réduction de l'espace occupé en mémoire, on les traite comme des matrices.

Q 5) Nous avons un tableau à N lignes (ici 100000) et m colonnes (ici 6) à valeurs de 32 bits.

Donc, nous avons $N \times m$ cases à 32 bits soit $100000 \times 6 \times 32 = 19200000$ bits.

Puis, nous avons un vecteur à N lignes soit N cases de 8 bits donc 800000 bits.

Donc, le tableau et le vecteur occupent au plus 20000000 bits.

Or, 8 bits = 1 octet.


Donc $20\text{ Mbits} = 2500000$ octets
 $= 2,5\text{ Mo}$

La mémoire totale nécessaire est de 2,5 Mo.



Q 6)

```
def separationParGroupe(data, etat):  
    normal = []  
    hernie = []  
    spody = []  
    for i in range(len(etat)):  
        if etat[i] == 0 :  
            normal.append(data[i])  
        elif etat[i] == 1 :  
            hernie.append(data[i])  
        elif etat[i] == 2 :  
            spody.append(data[i])  
    return normal, hernie, spody
```



Q 7

• TEST : $i \neq j$



• ARCS1 : $n, n, i * m + j + 1$

• ARCS2 : $\text{data}[:, i], \text{data}[:, j],$
 $\text{marks} = \text{mark} \quad [\text{stat}]$



• ARCS3 : $\text{data}[:, i]$



Q10) def min-max(X):

min = X[0]

max = X[0]

for i in range(len(X)):

if max < X[i]:

max = X[i]

elif min > X[i]:

min = X[i]

return max, min



Complexité ?

Q11) def distance(z, data):

dist = []

for i in range(len(data)):

S = 0

for j in range(len(data[0])):

S += (z[j] - data[i][j])²

S = sqrt(S)

dist.append(S)

return S



Q12) La partie 1 crée un tableau à n colonnes et 2 lignes où dans la première ligne elle associe les distances euclidiennes et dans la deuxième ligne associe le chiffre correspondant au vecteur z . Ainsi, le tableau est un récapitulatif des distances euclidiennes et puis elle tire ce tableau.

La partie 2 crée un tableau comptant le nombre de personnes par état tenant compte des attributs

La partie 3 vérifie les informations et la compare avec le tableau créé en partie 2 et renvoie si le patient est malade.

T est le tableau des distances ordonnées par attribut dont la première colonne est la ligne de l'âge (index de l'attribut)

select : nombre de patients par maladie dépendant des attributs.

ind : état du patient

Q13) Sur la diagonale, on a le nombre de personnes par état dont la maladie réelle et celle dont l'IA a fait prédiction coïncident. Ici, il y a 23 patients normaux, 11 avec un ulcère et 40 avec la 2^e maladie dont le résultat trouvé et attendu coïncident.

De plus, dans la première ligne, le 4 coïncide avec 4 patients normaux dont ils ont eu une prédiction d'ulcère et 7 normaux avec la 2^e maladie.

Pour la première colonne, nous avons pour le 7 le nombre de personnes avec une prédiction de normalité mais avec un ulcère. De même pour le 5 et la 2^e maladie.

Donc, ce tableau sert à trouver le nombre de fois de coïncidence et non-coïncidence de la prédiction avec la réalité, en étant le nombre de la ligne l'état réel du patient et la colonne l'état de prédiction de l'IA.

On pourra évaluer la taxe de réussite ✓

Q4) la taxe de réussite est toujours supérieure à 68% mais toujours inférieure à 75% avec très peu de variation et un apogée pour $k = 8$. ✓

Ainsi, le programme a une efficacité sensible pour tout k , mais pas suffisante car elle n'excède pas 75% (en médecine, c'est peu)

✓

Q15)

```
def moyenne(x):
```

```
    S = 0
```

```
    for i in range(len(x)):
```

```
        S += x[i]
```

```
    return S / len(x)
```



```
def variance(x):
```

```
    m = moyenne(x)
```

```
    S = 0
```

```
    for i in range(len(x)):
```

```
        S += x[i] - m**2
```

```
    return S / len(x)
```



```

Q16) def synthese (data, etat) :
    t = np.zeros (3, len (data[0]))
    groups = Separation.Groups (data, etat)
    for i in range (1, 4):
        for j in range (len (data[0])):
            t[i-1; j] = [moyenne (groups[i-1]
                                [:, j], variance (groups[i-1] [:, j]

return t

```