

DS d'informatique

Analyse des données

Q₁: Patients ayant une hernie discale



```
SELECT id FROM MEDICAL WHERE etat = "hernie discale"
```

Q₂: Nom, prénom ayant le syndrome de...

```
SELECT nom, prenom FROM PATIENT WHERE MEDICAL.etat = "syndrome de..."
```

Q₃: États et nombres de patients pour chaque état

Il faut faire une jointure

```
SELECT count(id), état FROM MEDICAL GROUP BY état
```

Distinct

Q₄: intérêt d'écriture manuscrite

~~avec ce module, on dispose d'outils permettant d'extraire des données structurées de données de données...~~

La structure array prend même l'aspect d'une liste python traditionnelle.



Q₅: Taille du tableau

Q₆: fonction separation Par Groupe

def separation Par Groupe (data, etat):

norm = []

hern = []

Spond = []

for i in range(N):

if etat[i] == 0:

norm.append(data[i])

elif etat[i] == 1:

hern.append(data[i])

else:

Spond.append(data[i])

return [norm, hern, Spond]

Q₇: ARG1, ARG2, ARG3 et TEST

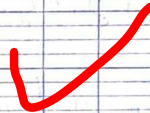
ARG1: (3, 1, i)

ARG2:

Apprentissage et production

Q₉ : espérance de $x_{\text{max}-j}$

$$x_{\text{max}-j} = \frac{x_j - \min(x)}{\max(x) - \min(x)}$$



Q₁₀ : fonction min-max (X)

def min-max(X):

min = X[0]

max = X[0]

for i in range(1, len(X)):

if X[i] < min:

min = X[i]

elif X[i] > max:

max = X[i]



return min, max

un seul passage de liste donc complexité en $O(n)$



Q₁₁: fonction distance

def distance(z, data):

distances = []

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

d = 0

for j in range(len(z)):

d = d + (data[i][j] - z[j])**2

distances.append(sqrt(d))

return distances

Q₁₂: Explications

La partie 1 crée une liste de liste contenant les distances entre les vecteurs de data et z ainsi que le numéro du vecteur de data correspondant, puis trie cette liste

~~La partie 2 crée une liste de nb éléments, puis on récupère les résultats de la partie 1, on récupère ensuite les index de la liste état initial par ces distances, puis on remplace les éléments de select indicés par les indices précédents de 1.~~

La partie 2 crée une liste de nb éléments contenant le minimum de la partie 1. Dans chaque état

La partie 3

Serie - DS d'implémentation

Validation de l'algorithme

Q_{13} : Matrice de confusion

La diagonale de la matrice donne le nombre de fois où l'algorithme a correctement la bonne valeur pour chaque état ✓

1^{ère} ligne : l'algorithme affiche 23 fois 0 quand il doit afficher 0, 4 fois 1 quand il doit afficher 0 et 7 fois 2 quand il doit afficher 0. ✓

1^{ère} colonne : quand on prévoit l'état 0, l'algorithme a réussi 23 fois, et a raté 17 fois : 7 fois il devrait afficher 1 et 5 fois, il devrait afficher 2.

Donc cette matrice sert à évaluer l'efficacité de l'algorithme

Q_{14} : Coût d'efficacité ✓

Le taux de réussite est instable : il est plus bas aux bas et hautes valeurs de k .
de plus, il est moyenné par 75%

donc l'efficacité n'est pas bonne ✓

Méthode de classification naïve

Q15: Moyenne et variance

def moyenne (X)

m = 0

for i in range (len(X)):

m = m + X[i]

return m / len(X)

def variance (X):

m = moyenne(X)

n = 0

for i in range (len(X)):

n = n + (X[i] - m)**2

return n / len(X)

Q16: Fonction Synthèse

def Synthèse (data, etat):

L = Separation Par Groupe (data, etat)

L = []

for i in range (len(L)):

L[i] = []

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

l₁.append(moyenne(L[i][j]), variance(L[i][j]))

l.append(l₁)

[i,j]

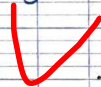


return l

Q11: fonction gaussienne

def gaussienne(a, moy, v):

return (1/sqrt(2*pi*v)) * exp(-(a-moy)**2/(2*v))



Q12: fonction probabilité Gauss (2, data, etat)

def probabilité Gauss (2, data, etat):

a = synthese (data, etat)

for i in synthese:

A = synthese