DS d'informatique

Analyse des données

Q_1 : Patient ayant une hernie discale ✓

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

Q_2 : Noms, prénoms ayant la spondylarthrite

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

Il faut faire une jointure

Q_3 : États et nombre de patient pour chaque état

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

Distinct

Q_4 : intérêt d'utiliser numpy

~~avec le module on dispose d'outils permettant d'extraire des lignes, colonnes, dimensions de matrice ...~~

La structure array prend moins d'espace d'espace qu'une liste python traditionnelle. ✓

Q_5 : Taille du tableau

Q6 : fonction SeparationParGroupe

```
def SeparationParGroupe(data, etat):
    mourm = []
    lhum = []
    Spond = []
    for i in range(N):
        if etat[i] == 0:
            mourm.append(data[i])
        elif etat[i] == 1:
            lhum.append(data[i])
        else:
            Spond.append(data[i])
    return [mourm, lhum, Spond]
```

Q7 : ARGS1, ARGS2, ARGS3 et TEST

ARGS1 : (3, 1, i)

ARGS2 :

Apprentissage et prédiction

Q9 : expression de x_{norm_j}

$$x_{norm\_j} = \frac{x_j - min(X)}{max(X) - min(X)}$$

✓

Q10 : fonction min_max(X)

```
def min_max(X):
    min = X[0]
    max = X[0]

    for i in range(0, len(X)):
        if X[i] < min:
            min = X[i]
        elif X[i] > max:
            max = X[i]

    return min, max
```

✓

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

✓

Q_{11} : 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
```

Q_{12} : explications

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

La partie 2 crée une liste de nb éléments, puis on récupère les premiers éléments créés en partie 1 ; on récupère ensuite les indices de la liste triée par ces distances, puis on remplit tous les éléments de label indicés par les indices précédents de 1.

La partie 2 crée une liste de nb éléments contenant les fréquences des partis dans chaque état

La partie 3

Suite - DS d'informatique

Validation de l'algorithme

Q_{13} : Matrice de confusion

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

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

1ère colonne : quand on prédit l'état 0, l'algorithme a réussi 23 fois, et a raté 12 fois : 7 fois il devait afficher 1 et 5 fois, il devait afficher 2.

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

Q_{14} : Courbe d'efficacité ✓

Le taux de réussite est instable ; il est plus bas aux bons et hauts valeurs de k

De plus, il est majoré par 75 %

Donc l'efficacité n'est pas bonne ✓

Méthode de classification naive

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)
    v = 0
    for i in range(len(X)):
        v = v + (X[i] - m)**2
    return v / len(X)
```

Q16 : Fonction Synthèse

```
def Synthese(data, etat):
    L = SeparationParGroupe(data, etat)
    l = []
    for i in range(len(L)):
        l1 = []
        for j in range(len(L[i])):
            l1.append((moyenne(L[i][j]), variance(L[i][j])))
        l.append(l1)
    return l
```

```
    for j in range(len(L[0])):
        l1.append(moyenne(L[i][j]), variance(L[i], [j]))
    l.append(l1)
```
[:,j]

```
return l
```

<u>Q_{17} : fonction gaussienne</u>

```
def    gaussienne(a, moy, v):
    return (1/sqrt(2*pi*v)) * exp(-((a-moy)**2)/(2*v))
```

<u>Q_{18} : fonction probabiliteGroupe(2, data, etat)</u>

```
def probabiliteGroupe(2, data, etat):
    a = synthese(data, etat)
    for i in synthese :
        A = synthese
```