MOSTEFAOUI Mayan
IPT - cycles 3-4

Q1) La requête SQL demandée est :

```
SELECT PATIENT.id FROM PATIENT JOIN MEDICAL ON
(PATIENT.id = MEDICAL.id AND etat='hernie discale');
```

Q2) La requête est :

```
SELECT PATIENT.nom, PATIENT.prenom FROM PATIENT JOIN MEDICAL
ON (PATIENT.id = MEDICAL.id AND etat='spondylolisthésis');
```

Q3) La requête est :

```
SELECT MEDICAL.etat, MAX(etat_groupe) FROM
(SELECT etat, count(*) as etat_groupe FROM MEDICAL GROUP BY etat);
```

Q4) L'intérêt d'utiliser numpy est de pouvoir disposer de fonctions optimisées en complexité temporelle pour des tableaux de (très) grande taille, facilitant de fait le traitement des données.

Q5) Pour $N = 10^5$:

- le tableau data contient 6×10^5 cases stockant des réels codés sur 4 octets.
- le vecteur etat contient 10^5 cases stockant des entiers sur 1 octet.

Donc le coût de stockage est de : $6 \times 10^5 \times 4 = 2{,}4 \times 10^6$ octets, soit 2,4 Mo pour le tableau data et de $10^5 \times 1 = 10^5$ octets, soit 0,1 Mo pour le tableau etat, d'où un total de 2,5 Mo.

Q6) La fonction demandée est :

```
def separationParGroupe(data_etat):
    """Sépare le tableau data suivant les valeurs du
    vecteur etat."""
    septab = [[], [], []]
    for i in range(len(data)):
        septab[etat[i]].append(data[i])

    return septab
```

Q7) Les arguments à utiliser sont :

- pour ARGS1 : (i, j, i+j)
- pour ARGS2 : (groupes[i], groupes[k], marker = mark[k])
- pour ARGS3 : groupes[i]
- pour TEST : i != j

Q8) Les diagrammes de la diagonale permettent d'avoir une distribution du nombre de patients pour un attribut donné.
Les diagrammes hors diagonale peuvent permettre en lumière certaines distributions de valeurs pour une pathologie donnée.

Q9) Une expression de $x_{norm,j}$ serait :

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

Q10) def min_max(X):
    """Retourne le maximum et le minimum de X."""
    miniX, maxiX = 0,0
    for x in X: #invariant : (miniX, maxiX) = (min(X), max(X)).
        if x > maxiX:
            maxiX = x
        elif x < miniX:
            miniX = x

    return miniX, maxiX

La fonction est bien de complexité linéaire

Q11) def distance(z, data):

    dist = []
    minZ, maxZ = min_max(Z)   Znorm = [(k-minZ)/(maxZ-minZ) for k in Z]
    for i in range(len(data)):
        minX, maxX = min_max(data[i])
        Xnorm = [(x-minX)/(maxX-minX) for x in data[i]]
        d = 0
        for j in range(len(Xnorm)):
            d += (Znorm[j] - Xnorm[j])**2
        d = sqrt(d)
        dist.append(d)
    return dist

Q12) Explication de la fonction KNN.

Partie 1 : → on initialise T (liste de liste contenant dist[i] et i) avec une liste vide.
→ on calcule dist, correspondant aux distances entre z et les données de data.
→ on ajoute à T la liste [dist(i), i] de manière à ce que T soit au format de l'énoncé, avec une boucle for de N tours.
→ on trie en place T.

Partie 2 : → on initialise une liste de nb zéros appelé select.
→ une boucle for de K tours incrémentant de 1 la case portée par l'indice de l'état de la ligne i de T:

```
select[ etat[T[i][1]] ] += 1
```

T[i][1] : récupère le numéro de la ligne
etat[...] : récupère l'état correspondant dans le vecteur etat (entre 0 et 2)
select[...] : case de select à incrémenter.

Partie 3 : → on initialise ind à 0
→ on initialise res avec la première case de select.
→ boucle for de nb-1 tours avec:
un test de comparaison et
éventuellement deux affectations.

variable res : cardinal du groupe majoritaire, ind : numéro du groupe majoritaire.

(Q13) L'information apportée par la matrice diagonale est de savoir si l'état réel i ($i \in [\![0,2]\!]$) est conforme à l'état prédit par l'algorithme KNN.

Première ligne : état_test = 0 → 23 uplets avec état_test = état_prédit
état_prédit $\in [\![0,2]\!]$ → 4 uplets avec état_prédit = 1
7 uplets avec état_prédit = 2

Première colonne : état_prédit = 0 → 23 uplets avec état_prédit = état_test
état_test $\in [\![0,2]\!]$ → 7 uplets avec état_test = 1
5 uplets avec état_test = 2

Cette matrice sert à mesurer la corrélation entre la prévision de l'algorithme KNN et les données de test, L'idéal étant de la rendre idéalement diagonale, ou au moins à coefficients diagonaux dominants.

(Q14) La courbe obtenue affiche des valeurs comprises entre 70% et 75% de réussite, avec le maximum atteints pour $K=8$ et $K=11$. Cela prouve que l'algorithme KNN n'est fonctionnel qu'avec des valeurs "modérées".
On peut aussi critiquer le pourcentage de réussite, insuffisamment élevé pour une application comme le pré-diagnostic médical.

Q15)
```
def moyenne(X):
    """Calcule la moyenne de X."""
    moy = 0
    n = len(X)
    for i in range(n):
        moy += X[i]

    return (moy/n)
```

Ces deux fonctions sont bien de complexité linéaire.

```
def variance(X):

    varian = 0
    moyX = moyenne(X)
    n = len(X)
    for i in range(n):
        varian += (X[i] - moyX**2)

    return (varian/n)
```

Q16)
```
def synthese(data, etat):

    synth = [[]]*len(etat)

    for i in range(len(etat)):
        for j in range(len(data)):
            mu = moyenne(data[j][i])
            vari = variance(data[j][i])
```

Q17)
```
def gaussienne(a, moy, v):
    return (1/sqrt(2*pi*v))* exp((-(a-moy)**2)/2*v)
```

Q18)
```
def probabiliteGroupe(z, data, etat):
    probas = [0,0,0]
    groupes = separationParGroupe(data, etat).
    synth = synthese(data, etat)
    for j in range(3):
        proba = 1
        for i in range(len(z)):
            proba *= gaussienne(z[i], synth[i][j][0], synth[i][j][1])
        probas[j] = proba*(1/3)
    return probas
```

Q19)
```
def prediction(z, data, etat):
    prob = probabiliteGroupe(z, data, etat)
    maxp, maxindex = 0,0
    for i in range(len(prob)):
        if prob[i] > maxp:
            maxp = prob[i]
            maxindex = i

    return i
```

Q20) L'utilisation du logarithme s'explique par les probabilités très faibles calculées : il est plus pratique de manipuler des quantités n'étant pas petites afin d'éviter les erreurs d'arrondi.

Q 21)