

Matrices et vertèbres

INFORMATIQUE COMMUNE - Devoir n° 3 - Olivier Reynet

A Parenthésage optimal

a Position du problème

Soit une matrice A de n lignes et m colonnes. On note a_{ij} un élément de la matrice A :

$$a_{ij} \text{ avec } \begin{cases} 0 \leq i \leq n-1 \\ 0 \leq j \leq m-1 \end{cases}$$

Dans cette section, on considère trois matrices :

- A de dimensions $n \times m$ et ses éléments a_{ij}
- B de dimensions $m \times p$ et ses éléments b_{ij}
- C de dimensions $p \times q$ et ses éléments c_{ij}

Le produit matriciel est une opération associative :

$$ABC = A(BC) = (AB)C \quad (1)$$

On peut donc décomposer ce calcul du produit de trois matrices de deux manières en fonction de la position des parenthèses.

En Python, on choisit de représenter une matrice comme une liste à n éléments, chaque élément de cette liste étant une liste de m nombres, c'est-à-dire qu'une matrice est une liste de liste.

A1. Écrire une fonction de signature `zeros_matrice(n,m) -> list[list[int]]` qui renvoie une matrice de taille $n \times m$ ne contenant que des zéros.

Solution :

```
def zeros_matrice(n,m):  
    return [[0 for _ in range(m)] for _ in range(n)]
```

A2. Écrire une fonction de signature `dimensions(M)` dont le paramètre est une matrice et qui renvoie sa dimension sous la forme d'un tuple d'entiers (nombre de lignes, nombre de colonnes).

Solution :

```
def dimensions(M):  
    assert len(M) > 0  
    n = len(M)  
    m = len(M[0])  
    return n,m
```

A3. On pose $P = AB$. Donner l'expression de l'élément p_{ij} de P en fonction de ceux de A et B .

Solution :

$$p_{ij} = \sum_{k=0}^{m-1} a_{ik} b_{kj} \quad \text{avec} \quad \begin{cases} 0 \leq i \leq n-1 \\ 0 \leq j \leq p-1 \end{cases} \quad (2)$$

A4. Écrire une fonction de signature `produit(A,B)` qui renvoie le produit des matrices A et B . On garantira par une assertion la faisabilité du calcul.

Solution :

```
def produit(A, B):
    n, m1 = dimensions(A)
    m2, p = dimensions(B)
    assert m1 == m2
    m = m1
    P = zeros_matrice(n, p)
    for i in range(n):
        for j in range(p):
            for k in range(m):
                P[i][j] += A[i][k] * B[k][j]
    return P
```

A5. Quel est le nombre de multiplications nécessaires pour calculer P en fonction de n, m et p ?

Solution : P possède $n \times p$ éléments. Chaque élément nécessite m produits. Donc il faut nmp multiplications.

A6. Combien de multiplications sont-elles nécessaires pour calculer $A(BC)$ avec $n = 10, m = 100, p = 5, q = 50$? Même question pour calculer $(AB)C$. Conclure.

Solution :

1. Pour calculer $A(BC)$:

- BC nécessite mpq multiplications et est une matrice $m \times q$.
- $A(BC)$ nécessite nmq multiplications.

Le total des multiplications est $mpq + nmq = (p + n)mq = 75000$.

2. Pour calculer $(AB)C$:

- AB nécessite nmp multiplications et est une matrice $n \times p$.
- $(AB)C$ nécessite npq multiplications.

Le total des multiplications est $nmp + npq = (m + q)np = 7500$.

L'écart est important, c'est pourquoi on cherche à exploiter l'associativité du produit matriciel pour optimiser le produit.

b Optimisation du produit de plusieurs matrices

Pour un produit de n matrices, on se propose de trouver le parenthésage qui minimise le nombre de multiplications d'éléments.

Soit une suite (A_i) de n matrices telles que la matrice A_i a pour dimensions $d_i \times d_{i+1}$ avec $0 \leq i < n$. Ainsi, le produit $A_0 A_1 \dots A_{n-1}$ est défini et a pour dimensions $d_0 \times d_n$. L'objectif est de déterminer le parenthésage optimal. On désigne ce nombre minimum de multiplications par *le coût optimal du produit*.

On définit les notations suivantes :

- $\Pi(i, j)$ désigne le produit matriciel $A_i A_{i+1} \dots A_j$ avec $0 \leq i \leq j < n$ et $\Pi(i, i) = A_i$.
- $S(i, j)$ désigne le coût optimal du produit $\Pi(i, j)$.

- A7. Si n est grand, le nombre de parenthésages possibles du produit de n matrices est de l'ordre du nombre de Catalan de rang n , c'est-à-dire $C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$. Conclure.

Solution : Le nombre donné permet de conclure que si on veut énumérer toutes les possibilités pour trouver l'optimale, l'algorithme possèdera une complexité exponentielle, ce qui n'est pas envisageable..

- A8. Justifiez l'existence d'un coût optimal pour tout produit $\Pi(i, j)$.

Solution : Le nombre de parenthésages possibles pour n matrice est un nombre fini. Le coût d'un parenthésage est un entier positif. Toute partie de \mathbb{N} possède un plus petit élément. Donc il existe nécessairement un coût minimal.

- A9. Montrez que, si un parenthésage optimal pour $\Pi(i, j)$ conduit à calculer le produit de $\Pi(i, k)$ par $\Pi(k+1, j)$ avec $i \leq k < j$, alors les parenthésages de $\Pi(i, k)$ et $\Pi(k+1, j)$ induits par celui de $\Pi(i, j)$ sont optimaux.

Solution : On dispose d'un parenthésage optimal pour $\Pi(i, j)$. Soit un entier k tel que $i \leq k < j$. Ce parenthésage optimal conduit à séparer le produit $A_i \dots A_j$ entre A_k et A_{k+1} pour un certain k :

$$\Pi(i, j) = A_i \dots A_k A_{k+1} \dots A_j = \Pi(i, k) \Pi(k+1, j)$$

Démonstration. Par l'absurde. Supposons que le parenthésage de $\Pi(i, k)$ induit par celui de $\Pi(i, j)$ n'est pas optimal. On peut donc le remplacer un parenthésage optimal pour réduire son coût. Mais cela réduit également le coût de $\Pi(i, j)$, ce qui est absurde d'après notre hypothèse. On raisonne de même pour $\Pi(k+1, j)$. Les parenthésages induits sont donc nécessairement optimaux. ■

C'est le principe de Bellman : «Tout sous-chemin d'un chemin optimal est optimal».

Le principe de Bellman permet de conclure que :

$$S(i, j) = \begin{cases} +\infty & \text{si } i > j \\ 0 & \text{si } i = j < n \\ \min_{i \leq k < j} (S(i, k) + S(k+1, j) + d_i d_{k+1} d_{j+1}) & \text{si } i < j < n \end{cases} \quad (3)$$

A10. Justifier l'équation 3 dans chaque cas.

Solution :

- Si $i > j$, le produit n'est pas calculable à cause des dimensions incompatibles.
- Si $i = j$, $S(i, j)$, $\Pi(i, j)$ est la matrice A_i . Donc $S(i, i) = 0$ pour tout i car il n'y a rien à calculer.
- Soit $i < j$ et k le point de découpe du produit : $\Pi(i, j) = \Pi(i, k)\Pi(k + 1, j)$. Pour calculer le coût de ce découpage, il faut faire la somme :
 1. du coût du produit $\Pi(i, k)$, dont la valeur optimale est $S(i, k)$
 2. du coût du produit $\Pi(k + 1, j)$, dont la valeur optimale est $S(k + 1, j)$
 3. du coût du produit de ces deux matrices : $\Pi(i, k)$ est une matrice $d_i \times d_{k+1}$ et $\Pi(k + 1, j)$ est une matrice $d_{k+1} \times d_{j+1}$. Ce produit génère $d_i d_{k+1} d_{j+1}$ multiplications.
 On cherche le k qui minimise la somme de ces coûts en prenant le minimum.

A11. Écrire une fonction de signature `pdyn_par(d)` qui renvoie, pour une liste `d` des dimensions des matrices, le coût optimal du produit des matrices de 0 à $n - 1$ ainsi que l'indice de découpage optimal sous la forme d'un tuple. Cette fonction procède de manière itérative en complétant un tableau dans un certain sens. Dessiner la relation de récurrence sur le tableau peut vous aider à programmer l'ordre de sa complétion. On pourra utiliser l'objet `inf` qui représente l'infini de la bibliothèque `math`, pourvu que l'importation soit correcte. L'utilisation de la fonction `min` n'est **pas** autorisée.

Solution :

```
def pdyn_par(d):
    n = len(d) - 1 # car la dernière dimension n'est pas l'indice d'une matrice
    S = [[math.inf for _ in range(n)] for _ in range(n)]
    kmin = 0
    for i in range(n - 1, -1, -1): # on a besoin des lignes suivantes pour calculer
        for j in range(i, n):
            if i == j:
                S[i][j] = 0
            elif i > j:
                S[i][j] = math.inf
            else:
                smin = math.inf
                kmin = i
                for k in range(i, j):
                    s = S[i][k] + S[k + 1][j] + d[i] * d[k + 1] * d[j + 1]
                    if s < smin:
                        smin = s
                        kmin = k
                S[i][j] = smin
    return S[0][n - 1], kmin
```

A12. Discuter de la complexité de la fonction `pdyn_par` en relation avec le calcul matriciel. Est-ce un calcul préalable pertinent et si oui sous quelles conditions?

Solution : La complexité de `pdyn_par` est en $O(N^3)$, où N est le nombre de matrices à multiplier, à cause des deux boucles à N itérations et de la boucle interne sur k qui est en $O(j - i)$.

Si les dimensions d_i des matrices sont grandes devant le nombre de matrices N , c'est-à-dire $\forall i \in \llbracket 0, N - 1 \rrbracket, d_i \gg N$, alors c'est un calcul préalable pertinent.

En effet, le nombre optimal de multiplications $S(0, n - 1)$ se décompose en trois parties : $(S(i, k) + S(k + 1, n - 1) + d_i d_{k+1} d_n)$. Même en faisant abstraction des deux premiers coûts, le nombre de multiplications reste grand devant N^2 : en prenant d le plus petit de d_i, d_{j+1} et d_{k+1} , on a $d^3 \gg N^2$. Le calcul du parenthésage optimal est alors négligeable devant la complexité de la multiplication.



Certains algorithmes permettent de calculer l'optimal en $O(N \log N)$.

A13. Écrire une fonction récursive de signature `rec_par(d, i, j, memo)` qui produit le même résultat que `pdyn_par`. Le paramètre `memo` permet d'effectuer la mémoïsation. Pour résoudre le problème, la fonction est invoquée comme suit : `c = rec_par(d, 0, len(d) - 2, {})`.

Solution :

```
def rec_par(d, i, j, memo):
    if (i, j) in memo:
        return memo[(i, j)]
    elif i == j:
        return 0, 0
    else:
        smin = math.inf
        kmin = i
        for k in range(i, j):
            s = rec_par(d, i, k, memo)[0] + rec_par(d, k + 1, j, memo)[0] + d[i] * d[k + 1]
                * d[j + 1]
            if s < smin:
                smin = s
                kmin = k
        memo[(i, j)] = smin, kmin
    return memo[(i, j)]
```

B Vertèbres et apprentissage

a Analyse des données

Un jeu de données sur les différentes configurations de vertèbres chez l'être humain a été constitué patiemment et mis à disposition des chercheurs. Pour un nombre N de patients, 6 attributs biomécaniques ont été mesurés :

1. l'angle d'incidence du bassin en degrés,
2. l'angle d'orientation du bassin en degrés,
3. l'angle de lordose lombaire en degrés,
4. la pente du sacrum en degrés,

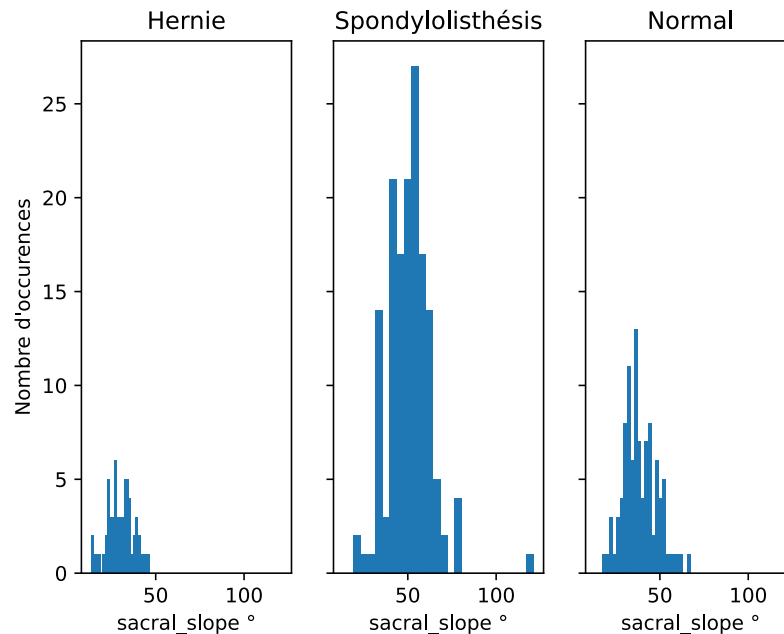


FIGURE 1 – Histogramme des données selon la pente du sacrum et les différentes classes

5. le rayon du bassin en mm,
6. la distance algébrique de glissement de spondylolisthésis en mm.

Le premières lignes du tableau des données sont représentées ci-dessous. La dernière colonne représente le diagnostic posé par le médecin :

- 0** hernie discale,
- 1** spondylolisthésis,
- 2** normal.

pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degre_1	class
57.28	15.14	63.99	42.13	116.73	30.34	1
63.02	22.55	39.60	40.47	98.67	-0.25	0
63.95	16.06	63.12	47.89	142.36	6.29	2
39.05	10.06	25.01	28.99	114.40	4.56	0
59.72	7.724	55.34	52.00	125.17	3.23	2
68.83	22.21	50.09	46.63	105.98	-3.53	0
63.40	14.11	48.13	49.28	111.91	31.78	1
41.18	5.792	42.86	35.39	103.34	27.66	1
89.83	22.63	90.56	67.19	100.50	3.040	2

TABLE 1 – Données médicales pour l'apprentissage automatique

B14. Dans l'optique d'un apprentissage automatique, on a tracé deux figures pour analyser les paramètres. Quelles conclusions simples sur le jeu de données peut-on tirer des figures 1 et 2?

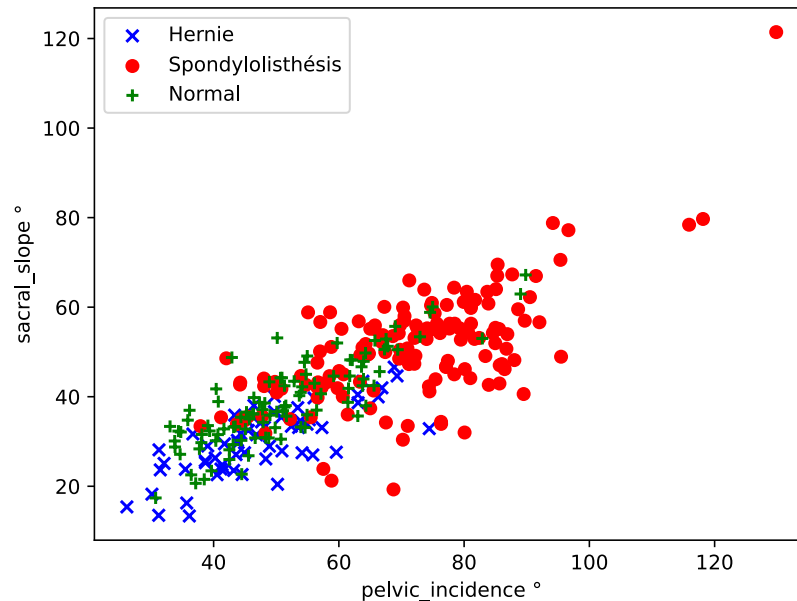


FIGURE 2 – Diagramme de dispersion selon l'orientation du bassin et la pente du sacrum

Solution : L'histogramme montre qu'il y a beaucoup plus d'échantillons de type 1 que de type 0 ou 2, ce qui peut entraîner un biais dans la classification : on va probablement mieux reconnaître les types 1 du fait de sa sur-représentation. Par ailleurs, la pente du sacrum semble pertinente pour distinguer les 0 des 1.

Le diagramme de dispersion montre que les deux paramètres permettent de distinguer 0 et 1 mais plus difficilement les 2.

Il est possible, si ce schéma se reproduit avec tous les paramètres, d'avoir de bonnes chances de classifier les 2 (échantillons plus nombreux et paramètres plus clivants) et plus de difficultés avec les 1 et les 0.

B15. Expliquer pourquoi on peut utiliser la méthode des K plus proches voisins sur ces données et quelle opération résulte de cette utilisation.

Solution : K plus proche voisins est un algorithme d'apprentissage supervisé, c'est-à-dire qu'il lui faut des données étiquetées par les classes en entrées. Or, c'est le cas de ce jeu de données. De plus, la dimension du problème n'est pas trop grande et le nombre d'échantillons non plus. C'est pourquoi on peut envisager d'utiliser KNN.

Une étude statistique des données est effectuée. On remarque que la différence entre l'élément maximal x_{max} et minimal x_{min} d'un attribut x peut varier fortement. Le tableau ci-dessous recense ces différences $x_{max} - x_{min}$ pour chaque attribut :

[103.68611919, 55.98681195, 111.7423855, 108.0626349, 92.98846564, 429.60126076]

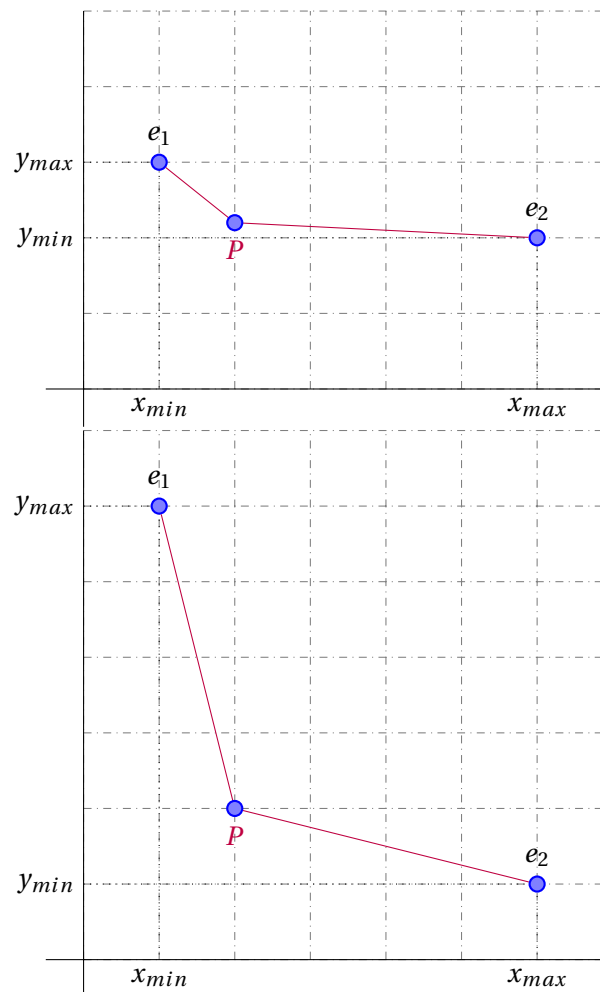
L'angle d'orientation du bassin s'étend sur une échelle de 56 unités, alors que la distance de glissement de spondylolisthésis sur une échelle de 430 unités.

B16. Cette observation peut-elle avoir une conséquence sur le résultat de l'algorithme des K plus proches voisins? Si oui, expliquer à l'aide d'un dessin sur un exemple simple en 2D quel est le problème.

Solution : Si les données ne sont pas mises à l'échelle, le paramètre dont la variation est la plus importante possède plus de poids proportionnellement qu'un autre dans la distance euclidienne et donc dans la classification opérée.

Ci-dessous, on a choisit un point P qui se situe proportionnellement à la même distance du minimum des deux paramètres x et y . Cependant, le premier graphique montre que la distance euclidienne accorde plus de poids à x qu'à y . Cela crée un biais en faveur du paramètre y qui sera plus déterminant, qui va plus peser sur la détermination de la classe : P paraît plus proche de e_1 car les variations de y sont négligeables devant celle de x .

Le second graphique montre qu'après mise à l'échelle des données, le poids de chaque paramètre est identique et P est à égale distance de e_1 et e_2 .



On dispose des données sous la forme **d'une liste de listes** E : chaque sous liste de E est une ligne du tableau 1. On souhaite procéder à un changement d'échelle des données afin quelles soient toutes

comprises entre 0 et 1. Dans ce but, on fait subir à toutes les données la transformation suivante :

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (4)$$

où x_i est la i ème valeur de l'attribut x , x_{min} et x_{max} le minimum et le maximum de l'attribut x .

- B17.** Expliquer en quoi la structure de données E est inadaptée pour effectuer la transformation 4 pour mettre à échelle les données. Un tableau Numpy est-il plus adapté? Pourquoi?

Solution : Le jeu de données est figé : on ne va pas en ajouter pendant le calcul. Donc un tableau dynamique (qui peut changer de taille) n'est pas nécessaire pour le calcul. Un tableau statique Numpy suffit.

Par ailleurs, une liste de liste n'est pas une matrice : en termes de manipulation, les colonnes ne sont pas accessibles comme les lignes.

On veut travailler sur les colonnes du tableau E . La syntaxe $E[0]$ permet bien d'accéder à toute une ligne de E . Mais la syntaxe $E[:,0]$ ne permet pas d'accéder à une colonne. Par contre, si E est un tableau Numpy, $E[0,:]$ renvoie une ligne et $E[:,0]$ une colonne. C'est pourquoi le calcul sur les colonnes sera plus facile avec Numpy. On pourra mieux sélectionner les éléments par colonne.

```
E = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(E[0][:]) # renvoie [1 2 3]
print(E[:,0]) # renvoie [1 2 3]

E = np.array(E)
print(E[0,:]) # renvoie [1 2 3]
print(E[:,0]) # renvoie [1 4 7]
```

On transforme donc la liste de listes E en tableau Numpy en utilisant la commande $E = np.array(E)$. Dans la suite, on manipulera donc ce tableau. On rappelle que la syntaxe pour accéder à l'élément d'indice i et j d'un tableau Numpy 2D T est $T[i,j]$.

- B18.** Écrire une fonction de prototype `scaler(E)` qui renvoie un nouveau tableau de données après application de la transformation 4. On fera attention à appliquer la transformation à tous les attributs mais **pas** à la classe des données. L'utilisation des fonctions Numpy `max` et `min` est autorisée. Leur manuel est donné ci-dessous. L'usage du calcul élément par élément est fortement conseillé mais pas obligatoire.

`numpy.max(a, axis)` : renvoie le maximum (ndarray ou scalar) d'un tableau ou les maxima selon un axe.
`numpy.min(a, axis)` : renvoie le minimum (ndarray ou scalar) d'un tableau ou les minima selon un axe.

Exemple :

```
import numpy as np
a = np.array([[10, 5.5], [43, 1.7]])
amin = np.min(a, 0) # amin vaut [10. 1.7]
```

Solution :

```
def scaler(E):
    Emin = np.min(E[:, :-1], 0)
```

```

E_max = np.max(E[:, :-1], 0)
E[:, :-1] = (E[:, :-1] - E_min) / (E_max - E_min)
return E

```

b Apprentissage

B19. Dans le cadre de l'algorithme des plus proches voisins, on utilise la fonction `trier` sur la liste `D` des tuples (indice du voisin, distance au voisin) pour l'ordonner du voisin le plus proche au plus éloigné. Compléter les lignes marquées # 1 à compléter afin que le tri soit opérationnel en conservant le numéro # pour faciliter la correction.

```

def f(t):
    n = len(t)
    t1, t2 = [], []
    for i in range(n // 2):
        t1.append(t[i])
    for i in range(n // 2, n):
        t2.append(t[i])
    return t1, t2

def g(t1, t2):
    n1, n2 = len(t1), len(t2)
    if n1 == 0:
        return ... # 1 à compléter
    elif n2 == 0:
        return ... # 2 à compléter
    elif t1[0][1] < t2[0][1]:
        return [t1[0]] + g(t1[1:], t2)
    else:
        return ... # 3 à compléter

def trier(t):
    if len(t) < 2:
        return t
    else:
        t1, t2 = f(t)
        return ... # 4 à compléter

```

B20. Écrire une fonction de signature `k_plus_proches(k, E, X)` qui renvoie les **indices** des k plus proches voisins de X dans le jeu des données E . L'utilisation de la fonction `numpy.linalg.norm` qui renvoie la norme euclidienne d'un vecteur est autorisée. La création d'une liste `D` des tuples (indice du voisin, distance au voisin) et l'utilisation de la fonction `trier` est obligatoire.

Exemple d'utilisation de la fonction `numpy.linalg.norm` :

```

a = np.array([1,2,3])
b = np.array([4,5,6])
np.linalg.norm(a-b)
# renvoie —> 5.196152422706632

```

Solution :

```
def k_plus_proches(k, E, X):
    assert len(E) > 0
    assert len(X) == len(E[0, :-1])
    n = len(E)
    D = [(i, np.linalg.norm(X - E[i, :-1])) for i in range(n)] # on enlève la classe
    T = trier(D)
    V = [v for v, d in T]
    return V[:k]
```

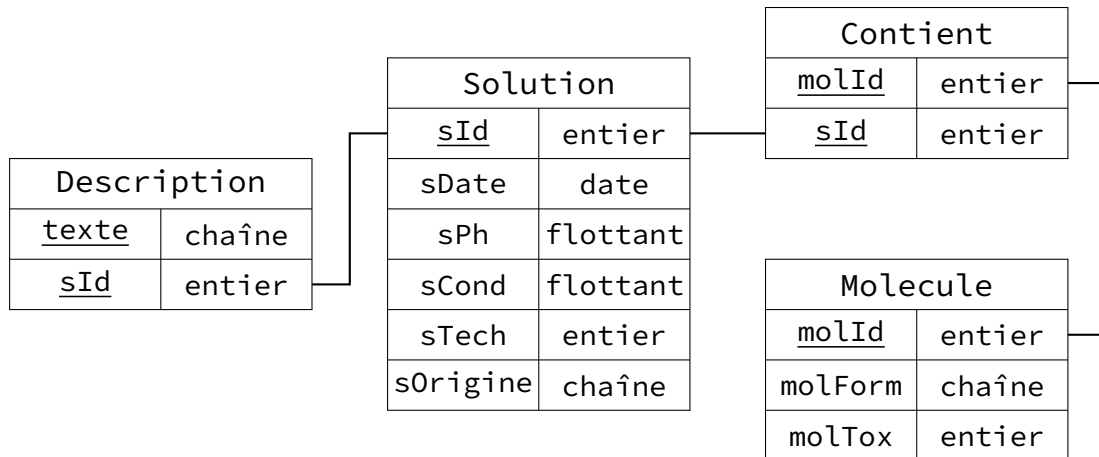
B21. Connaissant le résultat de la fonction précédente, proposer une solution pour déterminer la classe de x et l'implémenter. Cette fonction possède la signature `classe_majoritaire(V, E, N)`, où V est la liste des indices des k plus proches voisins dans E et N le nombre de classes.

Solution : On peut soit choisir la classe à la majorité relative des classes des voisins de x ou alors pondérer le nombre d'occurrences par la distance à x en utilisant $\frac{\sum_{i=0}^{k-1} d_i}{d_j}$ par exemple. On implémente la majorité relative.

```
def classe_majoritaire(V, E, N):
    h = [0 for _ in range(N)]
    for v in V:
        h[E[v,1]] += 1
    cmaj = 0
    vmax = h[0]
    for i in range(len(h)):
        if h[i] > vmax:
            vmax = h[i]
            cmaj = i
    return cmaj
```

C SQL forever

On dispose d'une base de données répertoriant des solutions pour lesquelles on a testé la présence de certaines molécules plus ou moins toxiques. Cette base de données est représentée par le schéma ci-dessous :



La clé primaire de chaque table est formée par les attributs soulignés.

On dispose pour chaque table de la description de ses attributs :

- **Solution** :
 - sId : identifiant de la solution,
 - sDate : date et heure de l'enregistrement de la solution (on supposera le type date compatible avec les opérations de comparaison, le minimum, le maximum),
 - sPh : pH de la solution,
 - sCond : conductivité de la solution,
 - sTech : identifiant du technicien ayant manipulé la solution,
 - sOrigine : origine de la solution.
- **Molécule** :
 - molId : identifiant de la molécule,
 - molForm : formule chimique de la molécule,
 - molTox : toxicité de la molécule. Si l'entier vaut 0, la molécule n'est pas toxique. Plus l'entier est grand, plus la toxicité est grande.

Les tables **Contient** et **Description** représentent des relations entre les entités. **Contient** indique que la présence d'une molécule a été détectée dans une solution et **Description** permet au technicien de décrire des éléments complémentaires associés à un solution sous la forme d'un texte.

C22. Dans la base de données décrite ci-dessus, une molécule peut-elle être présente dans plusieurs solutions différentes? Une molécule peut-elle être contenue plusieurs fois dans la même solution? Justifier la réponse.

Solution : C'est une relation de plusieurs à plusieurs « Une solution contient une ou plusieurs molécules » qui a été transformée dans le modèle relationnel en une table **Contient**. Grâce aux deux clés étrangères de **Contient** qui forme une seule clé primaire, on peut associer plusieurs solutions différentes avec une même molécule et plusieurs molécules différentes avec une même solution.

Par contre, la table Contient ne pourra contenir qu'une seule ligne pour une molécule et une solution donnée, car la paire constitue une clé primaire et est donc unique. Cela signifie qu'une solution ne contient qu'une seule fois une molécule et qu'une même molécule n'est comptée qu'une seule fois dans une solution.

- C23.** Écrire une requête SQL donnant la formule chimique des molécules présentes dans la base si leur toxicité est strictement supérieure à 10.

Solution :

```
SELECT mForm
FROM molecule
WHERE molTox > 10
```

- C24.** Écrire une requête SQL donnant l'identifiant et le pH des solutions dont la date d'enregistrement est **strictement** postérieure à "2023/04/01". Seuls les cinq premiers résultats sont présentés dans l'ordre décroissant de pH.

Solution :

```
SELECT sId, sPh
FROM Solution
WHERE sDate > "2023/04/01"
ORDER BY sPh DESC
LIMIT 5
```

- C25.** Écrire une requête affichant les formules des molécules contenues dans les solutions. Le résultat ne contient pas de doublons.

Solution :

```
SELECT DISTINCT(molForm)
FROM Molecule
JOIN Contient
ON Contient.molId = Molecule.molId
```

Le degré de toxicité d'une solution est sommairement évalué en faisant la somme des entiers `molTox` de chaque molécule contenue dans une solution.

- C26.** Écrire une requête donnant l'identifiant d'une solution et son degré de toxicité si celui-ci est strictement supérieur à 10.

Solution :

```
SELECT Contient.sId, SUM(molTox)
FROM Contient
```

```
JOIN Molecule ON Contient.molId = Molecule.molId  
GROUP BY sId  
HAVING SUM(molTox) > 10
```
