Pesée des fraises

INFORMATIQUE COMMUNE - Devoir n° 3 - Olivier Reynet

Consignes :

1. utiliser une copie différente pour chaque partie du sujet,
2. écrire son nom sur chaque copie,
3. écrire de manière lisible et intelligible,
4. préparer une réponse au brouillon avant de la reporter sur la feuille.

A Pesée

On souhaite modéliser l'usage d'une balance de Roberval et d'une boîte de poids $B = (b_1, b_2, \ldots, b_n)$ composée de n poids. L'unité des poids est le gramme. Chaque poids de la boîte est unique.

On dispose d'une liste de 13 poids représentée par une liste Python `B = [1000, 500, 200, 100, 100, 50, 20, 10, 10, 5, 2, 1, 1]`.

A1. Existe-t-il une limite au poids des objets que l'on peut peser? Pourquoi? Si elle existe, donner des instructions en Python qui permettent de calculer cette limite.

A2. Proposer une approche gloutonne pour trouver une liste de poids permettant de peser un objet de P g.

A3. Écrire une fonction de prototype `glouton(B : list[int],P : int)` qui implémente cette approche gloutonne. La fonction renvoie la liste des poids utilisés pour atteindre le poids `P` et `None` si ce n'est pas possible.

L'approche gloutonne est optimale, c'est-à-dire que lorsqu'elle fournit une solution, le nombre de poids utilisé est minimum. Mais on propose de le vérifier en pratique en calculant l'optimal avec une approche en programmation dynamique. Soit $S(i, j)$ le nombre de pièces minimum qu'il est nécessaire d'utiliser pour atteindre le poids j avec i poids de la boîte $B = (b_1, b_2, \ldots, b_n)$. Le principe de Bellman permet d'écrire que :

$$S(i,j) = \begin{cases} 0 & \text{si } j = 0 \\ +\infty & \text{si } j > 0 \text{ et } i = 0 \\ S(i-1,j) & \text{si } b_i > j \\ \min\left(1 + S(i-1, j-b_i), S(i-1,j)\right) & \text{sinon} \end{cases} \tag{1}$$

A4. Justifier la valeur de $S(i, j)$ pour chaque cas.

A5. Écrire une fonction de signature `nb_poids(B,P)` qui renvoie le nombre minimal de poids qu'il faut utiliser pour atteindre le poids `P`. Cette fonction procède de manière itérative, en complétant un tableau. On pourra utiliser l'objet `inf` de la bibliothèque `math`, pourvu que l'importation soit correcte.

A6. Quelle est la complexité de la fonction `nb_poids` ? Est-ce plus intéressant que l'approche gloutonne ?

A7. Expliquer pourquoi la mémoïsation est nécessaire pour écrire la fonction précédente récursivement.

A8. Écrire une fonction de signature `rec_nb_poids(B,P, memo)` qui renvoie le nombre minimal de poids qu'il faut utilisé pour atteindre le poids `P`. Cette fonction procède de manière récursive en utilisant la mémoïsation. Vous choisirez une structure de données adaptée à la mémoïsation pour le paramètre `memo` en expliquant pourquoi celle-ci est adaptée.
On dit qu'un système de poids est couvrant s'il permet d'atteindre tous les poids possibles de 0 à un entier maximum propre au système.

A9. Écrire une fonction de signature `couvrant(B)` qui renvoie `True` si le système de poids `B` est couvrant et `False sinon`.

A10. Quelle est la complexité de la fonction `couvrant` dans le pire et le meilleur des cas ?

A11. On dispose d'une boîte de poids B_2 constituée comme suit :

$$B_2 = (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024)$$

Ce système de poids est-il couvrant ? Y-a-t-il un avantage à utiliser cette boîte plutôt que B ?

B Usinage, usure et algorithmes

Les circuits électroniques nécessitent une réalisation précise. On peut créer des pistes de cuivre en détourant avec une fraiseuse numérique. Cependant, il est impossible de réaliser une largeur de piste absolument conforme à son plan et le concepteur de la pièce doit préciser la tolérance, c'est-à-dire un écart aux dimensions nominales qui permet un fonctionnement correct du circuit. La tolérance nécessaire peut être de l'ordre du micron dans le domaine des circuits microondes. Une fraise qui présente une certaine usure ne sera plus capable de garantir une telle tolérance.

On dispose d'une machine outil capable de mesurer la perte de masse des fraises et de mesurer ainsi leur usure : plus une fraise est légère, plus elle est usée. Dans le but de faire de la maintenance prédictive, on souhaite classer les fraises en cours d'utilisation en trois groupes repérés par 0, 1 ou 2, selon la perte de masse.

B1. Expliquer, en détaillant son fonctionnement, en quoi l'algorithme des K-moyennes peut-être utile dans l'objectif de créer ces trois groupes ?
On dispose d'une liste des pertes de masse de 11 fraises :
`P = [3,50,7,17,21,72,43,64,35,9,20]`
Les fraises sont identifiées par leur position dans la liste. L'unité u est une unité de masse.

B2. En opérant à la main à partir de la partition initiale `[[6, 4, 8], [10, 5, 7, 3, 9], [0, 1, 2]]`, et en s'aidant de schémas, calculer la partition qui résulte de l'algorithme des 3-moyennes sur cette liste. La partition initiale est constituée par les indices des éléments dans la liste `P`.

B3. Proposer un adjectif qui caractérise la principale différence entre l'approche de l'algorithme des K-moyennes et celle de l'algorithme des K plus proches voisins.

B4. En appliquant l'algorithme des 3 plus proches voisins sur les données étiquetées :
`E=[([3],0),([50], 2),([7],0),([57],2),([21],0),([72],2),([43],1),([64],2),([35],1),([9],0),([20],0)`
`,([13],0)]`
déterminer le groupe auquel appartient la fraise dont la perte de masse est $29u$. On choisit un nombre de voisins $k = \sqrt{\frac{n}{N}}$, si n est le nombre d'échantillons étiquetés et N le nombre de classes. En cas d'égalité sur la classe majoritaire, on choisit de prendre classe dont l'indice est le plus faible.

C Changement de dimension

Les fournisseurs du tour numérique parviennent à produire d'autres indicateurs du degré d'usure de la pièce. On dispose maintenant :

- d'une estimation de la largeur maximale de la fraise,
- du nombre de tours qu'elle a déjà effectués.

Au fur et à mesure, la production a réussi à produire un ensemble de données étiquetées en demandant aux techniciens d'évaluer eux-mêmes le groupe dans lequel devrait se trouver une fraise à l'aide d'autres contrôles. On dispose donc d'une liste `E` de n tuples `(p_1, p_2, p_3, c)` dans laquelle chaque tuple représente la perte de masse, la largeur maximale, le nombre de tours et le groupe de l'échantillon. On cherche à appliquer l'algorithme des K plus proches voisins à ces données.

C5. Écrire une fonction de signature `de(x: list[int],y: list[int])->float` qui renvoie la distance euclidienne entre deux points x et y d'un espace de dimension d. On s'assurera que les listes `x` et `y`, qui représentent les points de l'espace x et y ont bien la même dimension.

On dispose du code à trous suivant qui implémente le tri fusion. Ce tri permet de trier des listes du type `E=[(i0,d0), (i1,d1),..., (in,dn)]` d'après le second élément du tuple `di`.

```
def partager_en_deux(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 fusion(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]] + fusion(t1[1:], t2)
    else:
        return ...  # 3 à compléter

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

C6. En faisant attention à utiliser une formulation récursive, compléter les instructions `return ...` de manière idoine.

C7. Donner, sans la justifier, la complexité du tri fusion.

C8. Tel qu'il est implémenté, ce tri fusion est-il en place ?

C9. Écrire une fonction de signature `k_plus_proches(k : int, E : list[list[int]], X : list[int],)->list[int]` qui renvoie la liste des indices dans `E` des k plus proches voisins de `X` au sens de la distance euclidienne.

C10. Écrire une fonction de signature `classe_maj(V, E, N)` qui renvoie la classe majoritaire des éléments indexés par `V`, les voisins de `X`. Cette classe sera choisie à la majorité relative. En cas d'égalité on choisit la classe la plus petite. Le paramètre `N` est le nombre de classes.

La méthode de la validation croisée permet déterminer le k optimal de l'algorithme des k plus proches voisins, c'est-à-dire le nombre de voisins qui maximise le taux prédiction de l'algorithme. Elle consiste à tester tous les k sur un jeu de données divisé en b blocs que l'on considère alternativement comme des données de tests ou des données de vérification. Par exemple, pour $b = 4$, on a :

Valeur de k	Entrainement	Validation croisée	Précision
1	D1, D2, D3	D4	p14
1	D1, D2, D4	D3	p13
1	D1, D3, D4	D2	p12
1	D2, D3, D4	D1	p11
2	D1, D2, D3	D4	p24
2	D1, D2, D4	D3	p23
2	D1, D3, D4	D2	p22
2	D2, D3, D4	D1	p21
...	...	...	...

La moyenne des précisions obtenues pour chaque k permet par la suite de choisir la valeur optimale. Par exemple, dans le cas suivant, on choisira $k = 3$:

k	Précision moyenne
1	0,78
2	0,82
3	0,9
4	0,85
5	0,8
...	...

C11. Écrire une fonction de signature `blocks(E, b)` qui divise les échantillons en b blocs. Cette fonction renvoie donc une liste comportant b sous-listes. Les $b-1$ premières sous-listes sont de taille égale à $\lfloor n/b \rfloor$. La dernière sous liste comporte le reste des éléments de `E`.

On dispose d'une fonction de signature `knn(train,test)` qui renvoie la précision de la prédiction des classes de `test` d'après les données d'entrainement `train` et les classes vraies de `test`.

C12. Écrire une fonction de signature `validation_croisee(E, b, k_max)` qui renvoie le nombre de voisins optimal inférieur ou égal à k_{max} pour les données d'entrées.

D SQL forever

On dispose de la base de données décrite par la figure 1. Cette base de données comporte les éléments suivants :

1. la table fraise qui représente les fraises utilisées :
   - fraise_id : l'identifiant de la fraise, clef primaire
   - dur : la dureté de la fraise en N/m^2

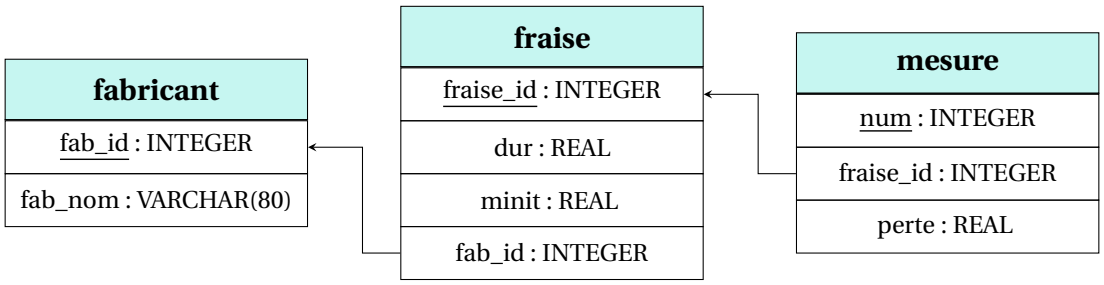

FIGURE 1 – Modèle physique de la base de données

- minit : la masse initiale de la fraise en u
- fab_id : l'identifiant du fabricant de la fraise, clé étrangère

2. la table fabricant :
   - fab_id : l'identifiant du fabricant, clef primaire,
   - fab_nom : le nom du fabricant
3. la table mesure :
   - num : le numéro de la mesure de perte de masse
   - fraise_id : l'identifiant de la fraise mesurée, clé étrangère
   - perte : la perte de masse de la fraise en %

D1. En utilisant des requêtes SQL, trouver les informations suivantes :

(a) Le nombre de fraises du fabricant dont l'identifiant est 42.

(b) Les noms des fabricants dont la dureté des fraises dépasse strictement les 250 N/m^2 dans l'ordre alphabétique. Il n'y aura pas de résultats redondants.

(c) Les identifiants, la masse initiale des fraises et leur perte de masse si les fraises ont perdu strictement plus de 10% de leur masse. Les résultats sont présentés par ordre décroissant de perte de masse et on limitera le nombre de résultat à 5.

(d) Le nombre de fraises, la moyenne de leur perte de masse et le nom du fabricant de ces fraises si la moyenne de la perte de masse est strictement inférieure à 5%.