Épreuve d'informatique MPI

Concours commun INP

2024-04-24

1 Algorithme Single Pass

Cette partie comporte des questions nécessitant un code en langage C.

On cherche à classer $n = 5$ documents textuels doc_i, $i \in [\![1, n]\!]$ dans lesquels les termes T_1, T_2 et T_3 apparaissent un certain nombre de fois, ces occurrences étant décrites dans le tableau suivant :

	doc_1	doc_2	doc_3	doc_4	doc_5
T_1	1	2	0	1	1
T_2	3	1	1	3	0
T_3	3	0	0	1	1

Chaque document doc_i est donc représenté par un ensemble de $p = 3$ valeurs. On notera d_i, $i \in [\![1, n]\!]$ un vecteur de $\mathbb{N}^3$, de composantes d_{ij}, $j \in [\![1, p]\!]$, d_{ij} indiquant le nombre d'occurrences du terme T_j dans le document doc_i.

On cherche à voir si des textes traitent des mêmes thématiques, en faisant l'hypothèse que des textes sont sémantiquement proches si des termes communs apparaissent.

Question 1. Recopier et remplir le tableau suivant en appliquant l'algorithme des k-moyennes avec $k = 2$ et d_2 et d_5 comme centres de classe initiaux. On utilisera la distance $\delta(d_i, d_j) = \sum_{\ell=1}^{p} |d_{i\ell} - d_{j\ell}|$. On notera de plus c_i les centres de classe et $\mathcal{A}_i$ les classes correspondantes.

Itération	c_1	c_2	$\mathcal{A}_1$	$\mathcal{A}_2$
1	d_2	d_5		
2				
3				

On souhaite maintenant traiter ce problème par une méthode dite « single pass » décrite dans l'algorithme 1.

Question 2. Appliquer l'algorithme 1 avec $\theta = 5.0$. Détailler les résultats des étapes de l'algorithme.

On propose d'implémenter cet algorithme en langage C. À cet effet, on définit un type structuré `vecteur` permettant d'encoder les centres de classe et les textes.

```c
struct vecteur_s {
        double *v;       // pointeur vers les coordonnées
        int taille;      // taille du vecteur
        int num_classe; // classe du vecteur
};
typedef struct vecteur_s vecteur;
```

Puisque le nombre de centres de classe varie au cours de l'algorithme, on utilise une liste chaînée de vecteurs pour représenter l'ensemble des centres de classe.

Algorithme 1 Algorithme Single Pass

Entrées : $(d_1, \ldots, d_n)$ les vecteurs des documents, θ un seuil qui appartient à $\mathbb{R}$.
Sorties : $(\mathcal{A}_1, \ldots, \mathcal{A}_j)$ les classes de centres $(c_1, \ldots, c_j)$.

$c_1 \leftarrow d_1$ `// Initialisation`
$\mathcal{A}_1 \leftarrow \{d_1\}$
$j \leftarrow 1$
pour i de 2 à n **faire**
  pour k de 1 à j **faire**
    Étape (i) Calculer $\delta(d_i, c_k)$
  fin pour
  si Étape (ii) $\delta(d_i, c_k) > \theta \forall c_k$ **alors**
    $j \leftarrow j + 1$ `// Création d'une nouvelle classe`
    $\mathcal{A}_j \leftarrow \{d_i\}$
    $c_j \leftarrow d_i$
  sinon
    `// Indice du centre de classe le plus proche de` d_i `au sens de` δ
    Étape (iii) $l \leftarrow \arg \min_{1 \leq k < j} (\delta(d_i, c_k))$
    $\mathcal{A}_l \leftarrow \mathcal{A}_l \cup \{d_i\}$ `// Affectation de` d_i `à la classe` ℓ
    $c_l \leftarrow \frac{1}{|\mathcal{A}_l|} \sum_{d_i \in \mathcal{A}_l} d_i$ `// Recalcul de` c_l
  fin si
fin pour

```
struct noeud_s {
        vecteur *c;
        struct noeud_s *suivant;
};
typedef struct noeud_s noeud;
```

Question 3. Écrire une fonction de prototype `void ajoutVecteur(vecteur *vec, noeud **tete)` permettant d'ajouter un vecteur `vec` en tête de la liste des centres de classe pointée par `tete`. On prendra soin de vérifier que l'allocation mémoire s'est bien passée.

On suppose dans la suite disposer :
— de la variable `vecteur *documents[nb_documents]` qui contient l'ensemble des documents, où pour tout $i \in [\![0, \texttt{nb\_documents} - 1]\!]$, `documents[i]` est égal à d_{i+1}.
— d'une fonction `void recalculCentre(vecteur *documents[], int nb_documents, noeud *tete, int l)` qui effectue le recalcul du centre c_ℓ et met à jour le nœud correspondant dans la liste chaînée des centres de classe.

Question 4. Écrire une fonction de prototype `double delta(vecteur *di, vecteur *c)` qui calcule la distance entre le document `di` et le centre de classe `c` (étape (i) de l'algorithme 1). On suppose que d_i et c ont la même taille.

Question 5. Écrire une fonction de prototype `bool distmax(double dists[], int j, double theta)` qui réalise l'étape (ii) de l'algorithme 1. Le tableau `dists` contient les distances de d_i à tous les c_k : pour tout $k \in [\![0, j-1]\!]$ l'élément `dists[k]` du tableau `dists` contient la distance de d_i à c_{k+1}. La fonction renvoie `true` si $\delta(d_i, c_k) > \theta \; \forall c_k$ et `false` sinon.

Question 6. Écrire une fonction de prototype `int distmin(double dists[], int j)` qui réalise l'étape (iii) de l'algorithme 1. Le tableau `dists` contient les distances de d_i à tous les c_k comme dans la question précédente. La fonction renvoie l'entier l décrit dans l'algorithme.

Question 7. À l'aide des questions précédentes et de la fonction `recalculCentre`, proposer une implémentation de l'algorithme 1 sous la forme d'une fonction de prototype `noeud *algorithme1(vecteur *documents[], int nb_documents, double theta)`. Évaluer la complexité de l'algorithme 1.