Résumé des diapos de « Algorithmes sur les données massives » Session 2018

Michel Beigbeder

18 mai 2018

Tri externe

- Pourquoi « externe » ?
- Hiérarchie de mémoire
- Tas
- Tri par tas
- Tri par fusion

Trouver des sous-ensembles similaires (1/5)

- Similarité de Jaccard entre deux sous-ensembles A et B de U : $Sim(A, B) = |A \cap B| / |A \cup B|$
- Données : D ensemble de sous-ensembles de U
- But : Soit $s \in [0, 1]$, trouver (presque) toutes les paires $\{A, B\}$ d'éléments de D telles que $Sim(A, B) \geq s$
- Idée :
  1. Hashage par minimum (*Minhashing*) : conversion de grandes **fonctions caractéristiques** en signatures, en préservant la similarité
  2. Hashage sensible à la localité (*Locality Sensitive Hashing, LSH*) : regroupe dans des mêmes alvéoles les signatures similaires
- application à des textes : on construit pour chaque texte l'ensemble de ses k-grammes, ou l'ensemble des *hash* de ses k-grammes (ou k est un petit entier)

Sous-ensembles similaires (2/5) : hashage par minimum

1. Soit σ une permutation des éléments de U, la fonction de hashage par minimum h_σ associe à une colonne le numéro de la première ligne dans laquelle la colonne C contient un 1
2. La signature d'une colonne est le résultat de l'application de plusieurs (disons : 100) fonctions de hashage par minimum indépendantes à une colonne.
3. On représente les signatures de la collection par une matrice (pleine ou non-creuse) de signatures
4. **Propriété :** La probabilité (sur toutes les permutations des lignes) que deux colonnes aient la même valeur de hashage par minimum est la similarité de Jaccard de ces colonnes

$$P(h_\sigma(C_1) = h_\sigma(C_2)) = Sim(C_1, C_2)$$

Sous-ensembles similaires (3/5) : implémentation du hashage par minimum

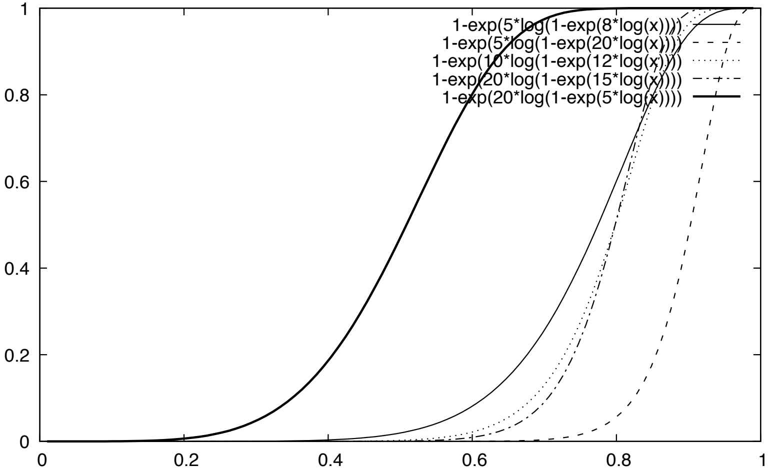
- Approximation à la permutation : hashage
  $h_i : \mathbb{N} \rightarrow \mathbb{N}$ et $h_i(x)$ est la nouvelle position de la ligne x
- Garder un tableau à deux dimensions indexé par les colonnes et les fonctions de hashage h_i : $M[i, c]$
- ... pour stocker le minimum des $h_i(r)$ pour laquelle la colonne C a un 1 dans la ligne r.

```
For each row r do
    For each hash function h_i (i.e. each permutation) do
        compute and store h_i(r)
    end For
    For each column C do
        If (C has 1 in row r) then
            For each hash function h_i // each permutation do
                If (h_i(r) is smaller than M(i, c)) then
                    M(i, c) = h_i(r)
                end If
            end For
        end If
    end For
end For
```

Sous-ensembles similaires (4/5) : hashage sensible à la localité

- But : éviter de calculer les similarités de Jaccard de toutes les paires, ce qui serait $O(n^2)$
- Comment :
  - Partionner les lignes de la matrice de signatures en b bandes
  - Chaque bande contient r lignes
    il y a donc $b \times r$ lignes au total
  - Pour chaque bande, on construit une table de hachage et on y insère les (identifiants de) colonnes.
    Nota bene : ces tables doivent être les plus grandes possibles
  - Les paires de candidats sont toutes les paires construites avec deux colonnes qui se retrouvent dans la même alvéole pour au moins une bande
  - Question : choisir b et r
- *Savoir évaluer la probabilité que les deux colonnes soient dans la même alvéole pour au moins une bande en fonction de leur similarité de Jaccard*

Sous-ensembles similaires (5/5) : choix du nombre de bandes et du nombre de lignes par bande



Filter un flot (1/1) : filtre de Bloom

- But : représenter (approximativement) un ensemble X
- Moyen : un tableau de bits et un ensemble de fonctions *hashage*, tous initialisés à zéro
- L'argument des fonctions de *hashage* est un élément, et la valeur de *hashage* est la position dans le tableau de bits
- Pour chaque élément x de X on positionne à 1 tous les bits $h(x)$ pour chaque fonction h
- Quand un élément y arrive dans le flot, pour savoir s'il appartient à X, on calcule $h(y)$ pour toutes les fonctions h
- Si tous les bits sont positionnés à 1, on déclare que y appartient à X
- Si un seul bit est positionné à 0, on déclare que y n'appartient pas à X
- *Savoir calculer la densité des 1 dans le tableau de bits en fonction de $|X|$, la taille du tableau, etc.*

Compter des 1 dans un flot de bits (1/3)

- But : étant donné un flot de bits, pouvoir répondre approximativement à des requêtes : Combien de 1 dans les k derniers bits, avec $k \leq N$ (N une constante) avec des complexités en temps et en espace sous-linéaires.
- Moyen : on résume les N derniers bits par une liste d'alvéoles.
- Une alvéole est un segment de la fenêtre qui contient un nombre de bits à 1 qui est une puissance de 2 ; elle est représentée par un enregistrement
  - l'estampille de sa fin (la date la plus récente) ($O(\log N)$ bits)
  - la taille de l'alvéole, le nombre de 1 ($O(\log \log N)$ bits)
- Il y a dans la liste soit une, soit deux alvéoles de la même taille
- Les alvéoles ne se recouvrent pas
- Les alvéoles sont triées par taille

  Les alvéoles plus récentes ne sont pas plus petites que les plus anciennes.
- Les alvéoles disparaissent lorsque leur date est $> N$.

Compter des 1 (2/3) : mise à jour de la liste d'alvéoles

- Si le nouveau bit est à 0... rien à faire
- Si le nouveau bit est à 1
  1. Créer une nouvelle alvéole de taille 1 pour ce bit
     Estampille de fin $\leftarrow$ valeur courante
  2. S'il y a maintenant trois alvéoles de taille 1, combiner les deux plus anciennes en une alvéole de taille 2
  3. S'il y a maintenant trois alvéoles de taille 2, combiner les deux plus anciennes en une alvéole de taille 4
  4. Et ainsi de suite...
- *Savoir le faire « à la main », voire préciser l'algorithme*

Compter des 1 (3/3) : interrogation

- Pour estimer le nombre de 1 dans les k bits les plus récents ($k \leq N$)
  1. On ne considère que alvéoles dont la date de fin est au plus k bits dans le passé
  2. Ajouter la taille de toutes les alvéoles sauf la plus ancienne
  3. Ajouter la moitié de la taille de l'alvéole la plus ancienne
- Rappel : on ne connaît pas combien de 1 de la dernière alvéole sont encore dans la fenêtre
- *Savoir majorer l'erreur commise par cette approximation*

Compter le nombre d'éléments distincts dans un flot (1/1)

- Une approximation : *hasher* chaque élément a dans 2^n (par exemple, $n = 64$)
- cette valeur de hash, $h(a)$, se termine par $r(a)$ bits à zéro
- une approximation du nombre d'objets différents est $2^{max_a r(a)}$
- en effet, $P(\max_{a \in A} r(a) \geq r) = P(\exists a \in A, r(a) \geq r) = 1 - (1 - 2^{-r})^m \approx 1 - e^{-m2^{-r}}$
  - si $m \gg 2^r$, $P(\max_{a \in A} \geq r) \approx 1$
  - si $m \ll 2^r$, $P(\max_{a \in A} \geq r) \approx 0$
- puis moyenne des médianes de ces valeurs pour plein de fonction *hashage*

Calculer des moments d'ordre 2 ou plus (1/1)

- Le moment d'ordre k est $\sum_{x \in E} m_x^k$
- on choisit uniformément une position i dans le flot, entre 1 et n
- on note : $e(i)$ l'élément à la position i.
- on note : $c(i)$ nombre d'occurrences de $e(i)$ aux positions $i, i+1, \ldots, n$
- chaque valeur de i fournit une estimation du moment d'ordre 2 :

$$n \times (2 \times c(i) - 1)$$

- *Savoir prouver que l'espérance de $n \times (2 \times c(i) - 1)$ est le moment d'ordre 2*
- *Savoir généraliser à des moments d'ordre supérieur...*
- *...et savoir ce que sont les moments d'ordre 0 et 1*