

Ingénierie de la fouille et de la
visualisation de données massives
(RCP216)
Réduction de l'ordre de complexité

Michel Crucianu

(prenom.nom@cnam.fr)

<http://cedric.cnam.fr/vertigo/Cours/RCP216/>

Département Informatique
Conservatoire National des Arts & Métiers, Paris, France

16 septembre 2025

Plan du cours

2 Réduction de l'ordre de complexité : besoins et problèmes

- Principe et typologie des besoins

3 Typologie des méthodes

4 *Locality-Sensitive Hashing* (LSH)

- Hachage et similarité
- *LSH* pour métriques courantes
- *LSH* et la similarité entre ensembles
- Amplification de fonctions *LSH*

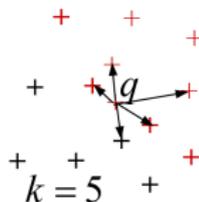
5 IVFFlat et HNSW

Méthodes de réduction de la complexité

A. Réduction du volume de données (sans changer l'ordre de complexité)

B. Réduction de l'ordre de complexité (sans diminuer le volume de données)

- Principe : les modèles sont souvent locaux, la décision est locale → les calculs impliquant des données éloignées peuvent être évités



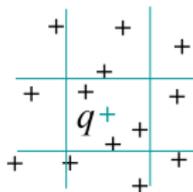
- Solutions basées sur l'utilisation d'index multidimensionnels ou métriques
- Méthodes exactes ou approximatives

Réduction de l'ordre de complexité : typologie des besoins

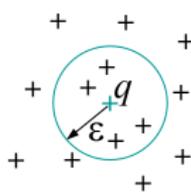
- Recherche par similarité (recommandation, *Retrieval Augmented Generation*, etc.)
- Décision : k plus proches voisins, SVM
- Estimation de densité (description, décision)
- Classification automatique, quantification
- Jointure par similarité (détection quasi-doublons, etc.)
- Apprentissage supervisé, apprentissage semi-supervisé

Recherche par similarité

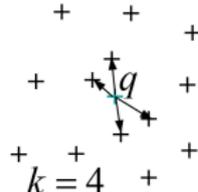
- Recherche par intervalle (*range query*) : $Range_r(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} \mid \forall i, |x_i - q_i| \leq r_i\}$
- Recherche dans un rayon (*sphere query*) : $Sphere_\epsilon(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} \mid d(\mathbf{x}, \mathbf{q}) \leq \epsilon\}$
- Recherche des k plus proches voisins (*kppv, kNN*) :
 $kNN(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} \mid |kNN(\mathbf{q})| = k \wedge \forall \mathbf{y} \in \mathcal{D} - kNN(\mathbf{q}), d(\mathbf{y}, \mathbf{q}) > d(\mathbf{x}, \mathbf{q})\}$



range query



sphere query

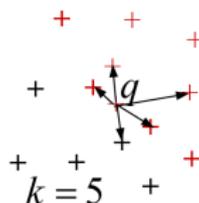


kNN query

N données \Rightarrow complexité $O(N)$?

Décision : k plus proches voisins

- 1 Trouver les k ppv de x
- 2 Vote majoritaire des k ppv (avec ou sans rejet d'ambiguïté)

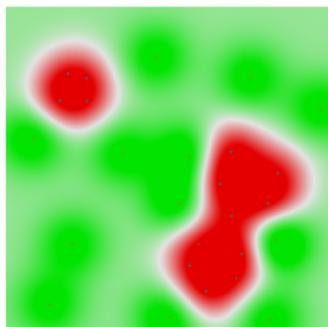


k plus proches voisins

N données \Rightarrow complexité $O(N)$?

Décision : SVM

$$f^*(\mathbf{x}) = \sum_{i=1}^N \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i) + b^*$$



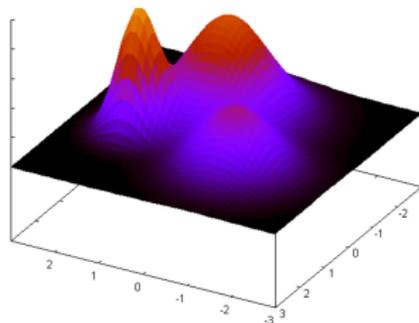
machine à noyaux

N vecteurs \mathbf{x}_i (vecteurs support) \Rightarrow complexité $O(N)$?

Noyau local ($K(\mathbf{x}, \mathbf{x}_i) \approx 0$ pour $d(\mathbf{x}, \mathbf{x}_i) > \epsilon$) \Rightarrow seuls comptent les voisins *dans un rayon*

Estimation de densité

- 1 Noyaux de Parzen : noyaux K identiques centrés sur les N observations initiales, la densité en \mathbf{x} est $f(\mathbf{x}) \propto \sum_{i=1}^N K(\mathbf{x}, \mathbf{x}_i)$
- 2 Modèle paramétrique (par ex. mélange additif de lois normales) : la densité est une somme pondérée de lois, on note par N le nombre de lois dans le mélange

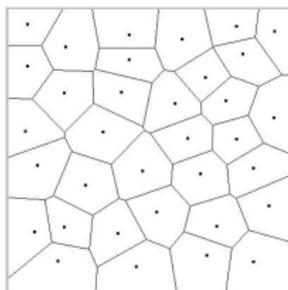


N noyaux \Rightarrow complexité $O(N)$?

Noyau local ($K(\mathbf{x}, \mathbf{x}_i) \approx 0$ pour $d(\mathbf{x}, \mathbf{x}_i) > \epsilon$) ou loi à décroissance rapide \Rightarrow seuls comptent les voisins *dans un rayon*

Classification automatique, quantification vectorielle

- *k-means* : groupes représentés par leurs centres de gravité, nombre de centres fixé (mais on considère que pour N données il faudrait utiliser $O(N^{1/3})$ centres)

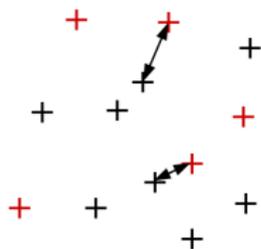


- *k-medoids* : extension aux espaces métriques en général, chaque groupe est représenté par son élément le plus central (*medoid*)
- Classification ascendante hiérarchique : arbre de regroupements, pouvant donner de multiples partitionnements

N données \Rightarrow complexité $O(N)$ (ou $O(N^{4/3})$), resp. $O(N^2)$ et $O(N^2 \log N)$?

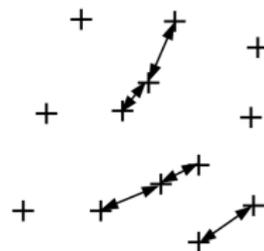
Jointure par similarité

- Jointure avec seuil de distance θ , ensembles \mathcal{D}_1 , \mathcal{D}_2 , avec N_1 et respectivement N_2 éléments : $K_\theta = \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in \mathcal{D}_1, \mathbf{y} \in \mathcal{D}_2, d(\mathbf{x}, \mathbf{y}) \leq \theta\}$
- Auto-jointure : $\mathcal{D}_1 \equiv \mathcal{D}_2$



jointure par similarité

⇒ complexité $O(N_1 \times N_2)$?



auto-jointure par similarité

⇒ complexité $O(N^2)$?

Réduction de l'ordre de complexité : objectifs

- Recherche par l'exemple, décision, estimation de densité : requête de même nature que les données
 - Objectif : $O(N) \rightarrow O(\log N)$ ou $O(1)$
- Apprentissage semi-supervisé, apprentissage actif, recherche par détecteur : requête = frontière de décision
 - Objectif : $O(N) \rightarrow O(\log N)$ ou $O(1)$
- Jointure par similarité, apprentissage supervisé à noyaux, *N-body problems*, certaines méthodes de classification automatique
 - Objectif : $O(N^2) \rightarrow O(N \log N)$ ou $O(N)$

- Remarque générale : complexité \geq taille du résultat !
 - Ex. : rayon de recherche trop grand \Rightarrow taille du résultat $O(N) \Rightarrow$ complexité $\geq O(N)$

Plan du cours

- 2 Réduction de l'ordre de complexité : besoins et problèmes
 - Principe et typologie des besoins

- 3 Typologie des méthodes

- 4 *Locality-Sensitive Hashing* (LSH)
 - Hachage et similarité
 - *LSH* pour métriques courantes
 - *LSH* et la similarité entre ensembles
 - Amplification de fonctions *LSH*

- 5 IVFFlat et HNSW

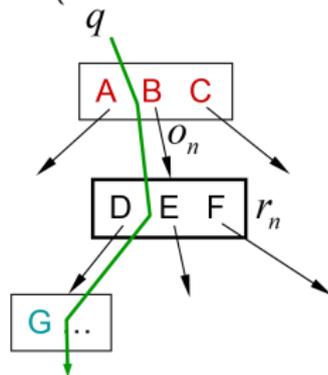
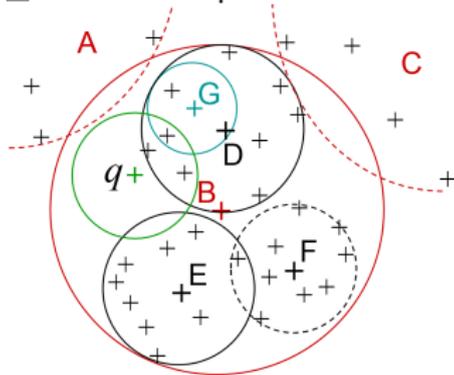
Réduction de complexité de la recherche par similarité : principe

■ Exemple des arbres de recherche :

- Regroupement hiérarchique des données
- Parcours de l'arbre en partant de la racine, idéalement $O(\log N)$ nœuds traversés \Rightarrow $O(\log N)$ calculs de distance, $O(\log N)$ échanges avec le disque

■ Hypothèses :

- 1 Seules les données proches de la requête sont utiles (sinon approche inutile)
- 2 Très peu de données sont proches (sinon réduction espérée faible)
- 3 Les données proches sont bien plus proches que les autres (sinon méthodes inefficaces)



Typologie des structures d'index

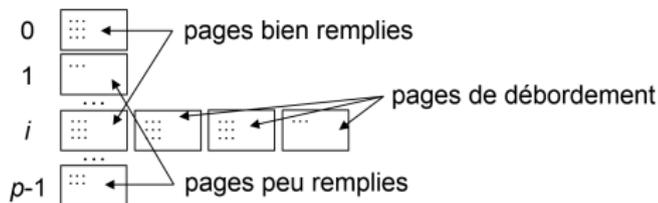
- Distinctions induites par la nature des données :
 - Espace vectoriel \Rightarrow il y a des coordonnées, on peut définir des centres de gravité, des densités de probabilité, etc.
 - Espace métrique \Rightarrow on ne peut parler que de distances
- Méthode de réduction de l'espace de recherche :
 - Partitionnement de l'espace de description : k-d-B-tree, LSD-tree, etc.
 - Partitionnement des données : R-tree, SS-tree, SR-tree, M-tree, etc.
 - Filtrage des données : VA-file, LPC-file, etc.
- Rapport avec la recherche approximative :
 - Définies pour la recherche exacte, adaptées à la recherche approximative : la plupart (dont M-tree, VA-file)
 - Conçues pour la recherche approximative : LSH, IVFFlat, HNSW, etc.
- Attention à la complexité de la **construction** de l'index !
- Pour une étude plus large du sujet, voir la monographie [5]

Plan du cours

- 2 Réduction de l'ordre de complexité : besoins et problèmes
 - Principe et typologie des besoins
- 3 Typologie des méthodes
- 4 *Locality-Sensitive Hashing* (LSH)
 - Hachage et similarité
 - *LSH* pour métriques courantes
 - *LSH* et la similarité entre ensembles
 - Amplification de fonctions *LSH*
- 5 IVFFlat et HNSW

Hachage

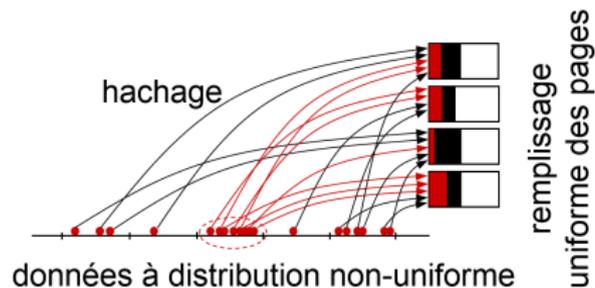
- Objectif dans ce contexte : trouver directement où est stockée une donnée
- Principe (BD relationnelle) : p pages disque, attribut qui prend des valeurs v dans un domaine \mathcal{D} , fonction de hachage $h : \mathcal{D} \rightarrow \{0, 1, 2, \dots, p - 1\}$
- Exemple (BD relationnelle) :
 - Table `film(titre, realisateur, annee)`, hachage sur l'attribut `titre`
 - Fonction de hachage (de préférence, p nombre premier) : $h(\text{titre}) = (\text{somme_codes_ascii_caractères}(\text{titre})) \text{ modulo } p$



- Propriété recherchée : « disperser » uniformément des données dont la distribution peut être très non-uniforme au départ

Hachage (2)

- Recherche par identité : retrouver les autres informations concernant un film à partir de son titre
 - Hachage du titre à rechercher
 - Lecture de la page disque identifiée par le *hash*⇒ complexité $O(1)$
- Dispersion uniforme de données non-uniformes → destruction de la structure de similarité des données ⇒ méthode inadaptée à la recherche par intervalle ou par similarité

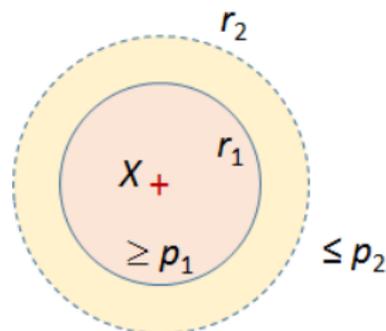


Hachage sensible à la similarité (*Locality-Sensitive Hashing, LSH*)

- Données dans un domaine \mathcal{D} , ensemble de *hash* (empreinte, condensé) \mathcal{Q} , métrique $d_{\mathcal{H}} : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}^+$
- **Définition** : $\mathcal{H} = \{h : \mathcal{D} \rightarrow \mathcal{Q}\}$ est un ensemble de fonctions de hachage (r_1, r_2, p_1, p_2) -sensibles (avec $r_2 > r_1 > 0$, $p_1 > p_2 > 0$) si (voir par ex. [2])

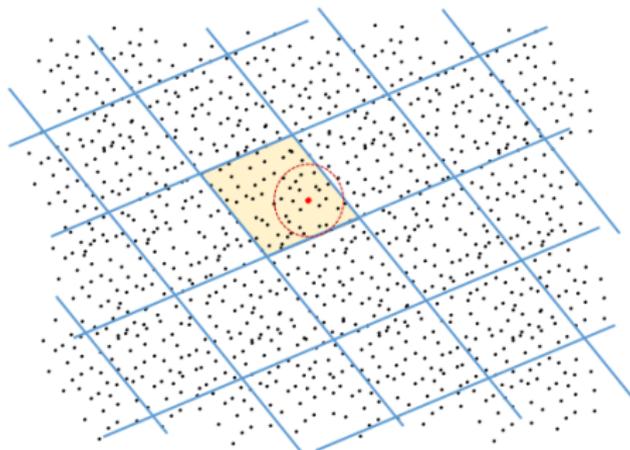
$$\forall x, y \in \mathcal{D}, \begin{cases} d_{\mathcal{H}}(x, y) \leq r_1 \Rightarrow P_{h \in \mathcal{H}}(h(x) = h(y)) \geq p_1 \\ d_{\mathcal{H}}(x, y) > r_2 \Rightarrow P_{h \in \mathcal{H}}(h(x) = h(y)) \leq p_2 \end{cases} \quad (1)$$

$x \neq y$, $h(x) = h(y)$: x et y **en collision**



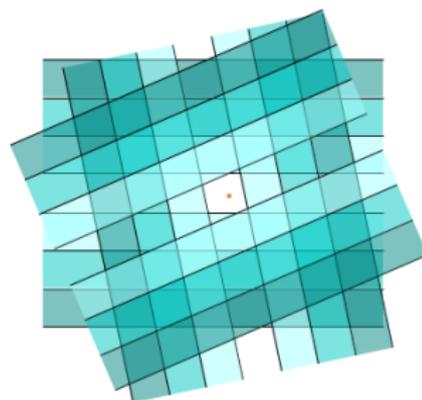
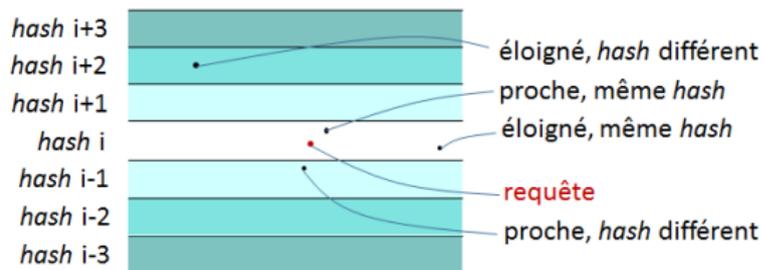
Recherche par similarité avec LSH

- Avant toute requête :
 - 1 Calculer la valeur de *hash* pour chacune des données de la base
 - 2 Stocker les données de même valeur de *hash* dans une même page (*bucket*)
 - Pour chaque donnée-requête :
 - 1 Hachage de la donnée-requête, lecture de la page (*bucket*) associée au *hash*
 - 2 Retour des données de cette page
 - 3 Éventuellement, filtrage de ces données par calcul des distances
- ⇒ Complexité $O(1)$



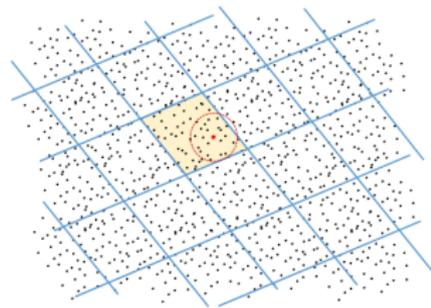
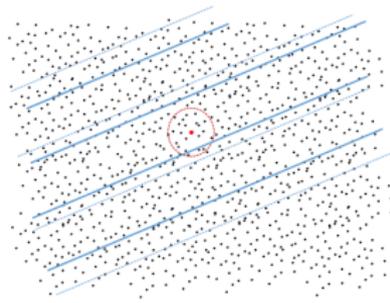
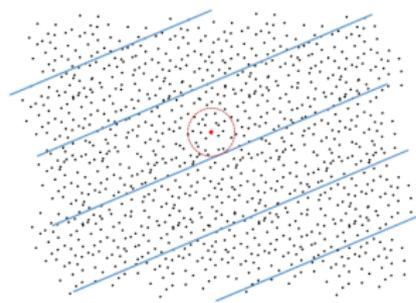
LSH pour la métrique euclidienne

- $\mathcal{D} = \mathbb{R}^m$, $\mathcal{Q} = \mathbb{Z}$, $d_{\mathcal{H}}$ est la métrique L_2
- **Fonctions élémentaires** $h : \mathbb{R}^m \rightarrow \mathbb{Z}$, $h_{\mathbf{a},b,w}(\mathbf{x}) = \left\lfloor \frac{\mathbf{a}^T \cdot \mathbf{x} + b}{w} \right\rfloor$ avec $\mathbf{a} \in \mathbb{R}^m$ de composantes tirées indépendamment suivant $\mathcal{N}(0, 1)$ et $b \in \mathbb{R}$ tiré suivant la loi uniforme dans $[0, 1)$, $w \in \mathbb{R}$
- **Table de hachage** (ou fonction de hachage composée) : concaténation de (ET logique entre) n fonctions élémentaires indépendantes \rightarrow *hash* de n entiers
- Exemple dans \mathbb{R}^2 , avec 3 fonctions élémentaires :



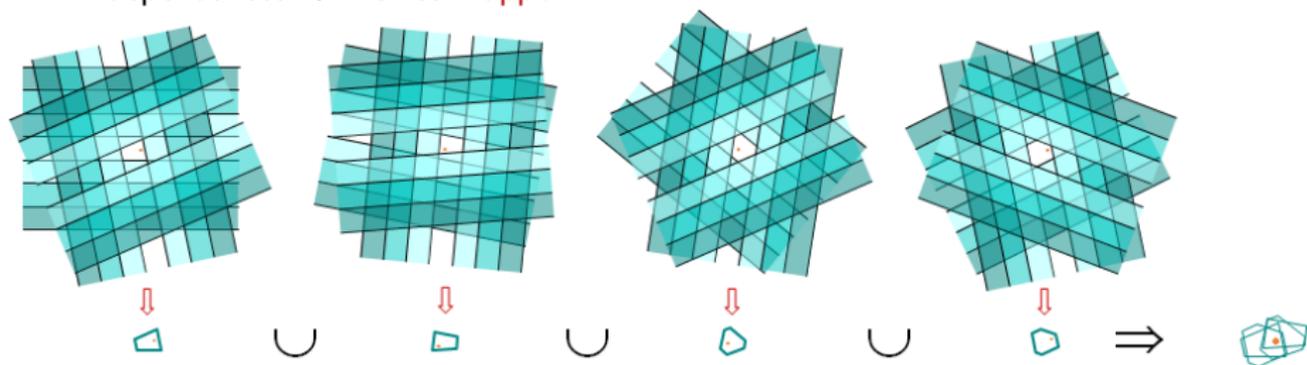
LSH : pourquoi regrouper des fonctions de hachage

- 1 fonction élémentaire n'est pas assez sélective \Rightarrow précision insuffisante (trop de faux positifs à filtrer ensuite par de coûteux calculs de distance)
- Grille plus fine \Rightarrow meilleure précision mais forte réduction du rappel
- Concaténation de (**ET** logique entre) $n (> 1)$ fonctions de hachage indépendantes (\rightarrow 1 **table** de hachage) \Rightarrow améliorer la **précision** sans diminuer trop fortement le rappel
- Attention, contrairement au cas présenté dans la figure, en général $n \ll$ dimension de l'espace !



LSH : pourquoi regrouper des tables de hachage

- 1 table de hachage \Rightarrow si requête proche d'une frontière alors des données similaires sont de l'autre côté \Rightarrow faux négatifs (réduction du rappel)
- Réunion (OU logique entre) des résultats issus de t (> 1) tables de hachage indépendantes \Rightarrow meilleur **rappel**



- *Multi-probe LSH* [3] : pour réduire le nombre de tables nécessaires, dans chaque table échantillonner aussi les *buckets* voisins



Similarité entre ensembles finis

- Indice (ou similarité) de Jaccard :
 - Soit un ensemble \mathcal{E} , $\mathcal{D} = \mathcal{P}(\mathcal{E})$ est l'ensemble des parties de \mathcal{E}
 - Similarité entre deux sous-ensembles $\mathcal{A}, \mathcal{B} \in \mathcal{P}(\mathcal{E})$: $s_J(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|}$
 - $d_J(\mathcal{A}, \mathcal{B}) = 1 - s_J(\mathcal{A}, \mathcal{B})$ est une distance (une métrique) sur $\mathcal{P}(\mathcal{E})$
- Définir les ensembles :
 - Texte : ensemble des mots présents dans le texte
 - Profil d'achat : ensemble d'articles achetés ou ensemble de *catégories* d'articles achetés
- Réduire l'ordre de complexité de la recherche par similarité (éviter de comparer une requête à chacune des N données de la base) : définir des fonctions LSH adaptées et s'en servir pour une recherche de complexité $O(1)$!

LSH pour ensembles finis

- Fonctions de hachage élémentaires : $h_\pi : \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{E}$, $h_\pi(\mathcal{A}) = \min \pi(\mathcal{A})$
 - On fixe un ordre des éléments de \mathcal{E}
 - π est une permutation des éléments de \mathcal{E}
 - $\min \pi(\mathcal{A})$ est l'élément de \mathcal{A} qui se retrouve **premier** après cette permutation
- Représentation des ensembles par leurs fonctions caractéristiques :
 - Exemple de permutation : $\pi = \begin{pmatrix} a & b & c & d & \dots \\ c & d & a & b & \dots \end{pmatrix}$

\mathcal{E}	\mathcal{A}	\mathcal{B}	\mathcal{C}	$\pi(\mathcal{E})$	$\pi(\mathcal{A})$	$\pi(\mathcal{B})$	$\pi(\mathcal{C})$
a	1	0	0	c	1	1	0
b	0	0	1	d	0	0	0
c	1	1	0	a	1	0	0
d	0	0	0	b	0	0	1
...				...			

$$\Rightarrow h_\pi(\mathcal{A}) = c = h_\pi(\mathcal{B}), h_\pi(\mathcal{C}) = b$$

LSH pour ensembles (2)

- Résultat important : pour toute fonction h_π et quels que soient les ensembles $\mathcal{A}, \mathcal{B} \in \mathcal{P}(\mathcal{E})$, la probabilité de collision est égale à la similarité entre les ensembles

$$p(h_\pi(\mathcal{A}) = h_\pi(\mathcal{B})) = s_J(\mathcal{A}, \mathcal{B})$$

- Justification :

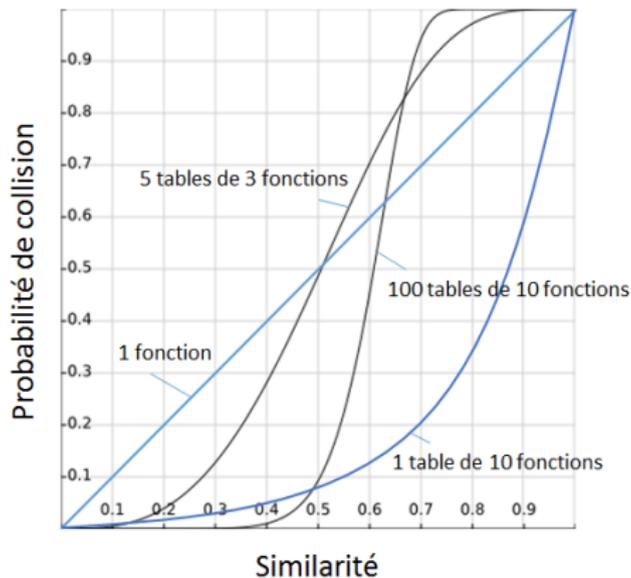
- Soit x le nombre d'éléments communs entre \mathcal{A} et \mathcal{B} (c'est à dire $|\mathcal{A} \cap \mathcal{B}|$)
- Soit y le nombre d'éléments spécifiques à \mathcal{A} ou \mathcal{B} (c'est à dire $|(\mathcal{A} - \mathcal{B}) \cup (\mathcal{B} - \mathcal{A})|$)
- Alors $s_J(\mathcal{A}, \mathcal{B}) = \frac{x}{x+y}$
- Pour toute fonction h_π , la probabilité de trouver en premier après la permutation π un élément commun plutôt qu'un élément spécifique est $\frac{x}{x+y}$
- Donc $p(h_\pi(\mathcal{A}) = h_\pi(\mathcal{B})) = \frac{x}{x+y}$

LSH pour ensembles (3)

- 1 Construction d'une table de hachage en regroupant n fonctions de hachage choisies aléatoirement de façon indépendante :
 - Le *hash* donné par la table est la concaténation des *hash* des n fonctions
 - 2 données sont en collision dans la table \Leftrightarrow elles sont en collision par rapport à la première fonction de hachage ET par rapport à la deuxième ET ... ET par rapport à la n -ème \Rightarrow probabilité de collision dans la table : s^n (s étant la similarité $s_J(\mathcal{A}, \mathcal{B})$)
- 2 Utilisation de *plusieurs* (t) tables de hachage :
 - 2 données sont en collision si elles sont en collision dans la première table OU dans la deuxième OU ...OU dans la t -ème \Rightarrow probabilité de collision dans **au moins** une table : $1 - (1 - s^n)^t$
Justification : probabilité de non collision dans 1 table = $1 - s^n$ \rightarrow probabilité de non collision dans les t tables = $(1 - s^n)^t$ \rightarrow probabilité de collision dans au moins 1 table = $1 - (1 - s^n)^t$

LSH pour ensembles (4)

- Graphique : probabilité de collision ($\in (0, 1)$) en fonction de $s_J(\mathcal{A}, \mathcal{B})$ ($\in (0, 1)$)



Généralisation : « amplification » de fonctions LSH

- Fonctions de hachage (r_1, r_2, p_1, p_2) -sensibles (avec $r_2 > r_1 > 0$, $p_1 > p_2 > 0$) :

$$\forall x, y \in \mathcal{D}, \begin{cases} d_{\mathcal{H}}(x, y) \leq r_1 \Rightarrow P_{h \in \mathcal{H}}(h(x) = h(y)) \geq p_1 \\ d_{\mathcal{H}}(x, y) > r_2 \Rightarrow P_{h \in \mathcal{H}}(h(x) = h(y)) \leq p_2 \end{cases} \quad (2)$$

- Amplification : rapprocher p_2 (probabilité de collision pour données dissimilaires) de 0 et p_1 (probabilité de collision pour données similaires) de 1
 - ET logique entre n fonctions : $h_{\text{AND}}(x) = h_{\text{AND}}(y)$ si et seulement si $h_i(x) = h_i(y)$ pour tout i , $1 \leq i \leq n \Rightarrow h_{\text{AND}}$ est (r_1, r_2, p_1^n, p_2^n) -sensible
 - justifie le regroupement de plusieurs fonctions dans une table
 - OU logique entre t fonctions : $h_{\text{OR}}(x) = h_{\text{OR}}(y)$ si et seulement si $h_i(x) = h_i(y)$ pour au moins une valeur de i , $1 \leq i \leq t \Rightarrow h_{\text{OR}}$ est $(r_1, r_2, 1 - (1 - p_1)^t, 1 - (1 - p_2)^t)$ -sensible
 - $h_{\text{OR,AND}}$ est $(r_1, r_2, 1 - (1 - p_1^n)^t, 1 - (1 - p_2^n)^t)$ -sensible
 - justifie l'utilisation conjointe de plusieurs tables

Plan du cours

- 2 Réduction de l'ordre de complexité : besoins et problèmes
 - Principe et typologie des besoins

- 3 Typologie des méthodes

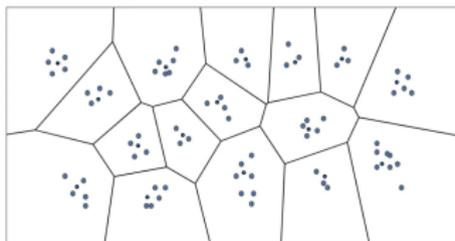
- 4 *Locality-Sensitive Hashing* (LSH)
 - Hachage et similarité
 - *LSH* pour métriques courantes
 - *LSH* et la similarité entre ensembles
 - Amplification de fonctions *LSH*

- 5 IVFFlat et HNSW

Inverted File with Flat Compression (IVFFlat)

■ Méthode :

- 1 Classification automatique des données (*K-means*, *GMM*) en k clusters
- 2 Les données de chaque *cluster* sont stockées dans une liste
- 3 Chaque requête retourne les données du *cluster* (ou des *clusters*) les plus proche(s)



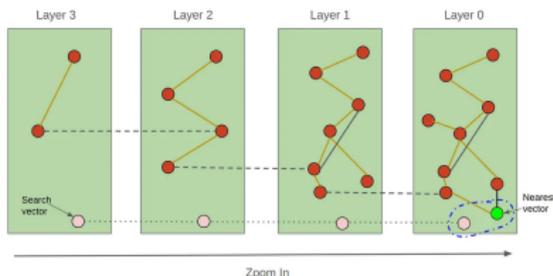
Source de l'illustration

- Complexité : construction $O(nk)$ (*K-means* : $k \sim \sqrt[3]{n}$), réponse à une requête $O(\frac{n}{k})$
- Employé dans : Faiss [1], Pinecone, PGVector (PostgreSQL), etc.
- IVFFlat : plus rapide à construire pour très grosses bases et moins gourmand en stockage, plus efficacement parallélisable
- Implémentation pour Spark dans [Spark Rapids ML](#)

Hierarchical Navigable Small Worlds (HNSW, [4])

■ Méthode :

- 1 Construction de graphes de similarité à différents niveaux de résolution
- 2 Le traitement de chaque requête suit les arêtes, du niveau le plus grossier au plus fin



Source de l'illustration

- Complexité : construction $O(n \log(n))$, réponse à une requête $O(\log(n))$
- Employé dans : Faiss [1], Milvus, PGVector (PostgreSQL), etc.
- L'efficacité de HNSW se dégrade moins vite avec la dimension que pour IVFFlat, HNSW est plus robuste par rapport à la distribution des données
- Implémentation pour Spark dans [Hnswlib](#)

Références I



M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou.

The faiss library, 2025.



A. Gionis, P. Indyk, and R. Motwani.

Similarity search in high dimensions via hashing.

In *Proceedings of the 25th International Conference on Very Large Data Bases, VLDB '99*, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.



Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li.

Multi-probe LSH : Efficient indexing for high-dimensional similarity search.

In *Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07*, pages 950–961. VLDB Endowment, 2007.



Y. A. Malkov and D. A. Yashunin.

Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, 2018.

Références II



H. Samet.

Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling).

Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.