Ingénierie de la fouille et de la visualisation de données massives (RCP216)

Recherche par similarité : LSH et bases vectorielles

Michel Crucianu
(prenom.nom@cnam.fr)
http://cedric.cnam.fr/vertigo/Cours/RCP216/

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

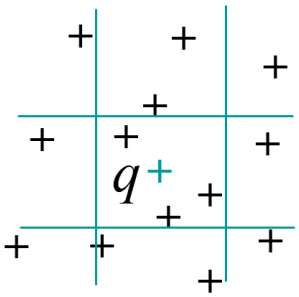
3 septembre 2026

Plan du cours

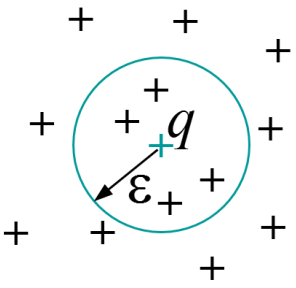
Recherche par similarité : définitions

Trouver dans une base $\mathcal{D}$ les données les plus similaires à une requête $\mathbf{q}$

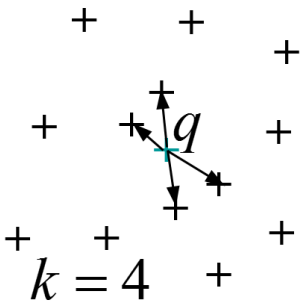
- Recherche par intervalle (*range query*) : $Range_r(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} \mid \forall i, |x_i - q_i| \leq r_i\}$
- Recherche dans un rayon (*sphere query*) : $Sphere_\epsilon(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} \mid d(\mathbf{x}, \mathbf{q}) \leq \epsilon\}$
- Recherche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

→ Recherche exhaustive : complexité $O(N)$ (linéaire en N)

Applications

- **Systèmes de recommandation** : articles, films, produits similaires à un profil utilisateur
- **Déduplication** et détection de quasi-copies à grande échelle
- **Recherche sémantique** : documents similaires à une requête en langage naturel, indépendamment des mots exacts employés
- **Systèmes RAG** (*Retrieval-Augmented Generation*) : augmenter un LLM avec des connaissances externes en lui fournissant les documents les plus pertinents

→ Une recherche par similarité efficace réduit aussi le coût de construction et d'utilisation de modèles prédictifs : la frontière de décision dépend surtout des données d'apprentissage les plus proches

Réduction de la complexité de la recherche par similarité

- Principe : regrouper au préalable les données par similarité et focaliser progressivement la recherche, en évitant les opérations sur les données éloignées

→ Structures d'index multidimensionnels ou métriques

- Deux hypothèses nécessaires pour pouvoir réduire la complexité
  1. **Très peu de données sont similaires de la requête** : en effet, la complexité ne peut pas être inférieure à la taille du résultat
  2. **Les données similaires sont** nettement plus similaires **que les autres** : l'efficacité de la réduction diminue avec les écarts de similarité

- Recherche **approximative**
  - *Approximate Nearest Neighbors* (ANN) : ne retourne pas tous les kNN et pas seulement des kNN
  - Pourquoi : relâcher les contraintes permet d'accélérer nettement la recherche
  - Certains index sont conçus dès le départ pour la recherche approximative, d'autres sont adaptables à la recherche approximative

Plan du cours

LSH *versus* hachage classique

- **Hachage classique** : associe à chaque donnée $v \in \mathcal{D}$ une valeur entière (*hash*) identifiant une page mémoire $\Rightarrow$ si même *hash* alors même page
  - Cherche à disperser uniformément les données pour uniformiser le remplissage des différentes pages $\Rightarrow$ deux données similaires ont des *hash* très différents
  - Adapté à la recherche par identité ($O(1)$), inadapté à la recherche par similarité

- **Hachage sensible à la similarité** (*Locality-Sensitive Hashing*, LSH) : principe inverse
  - Données proches $\rightarrow$ forte probabilité de même *hash* (donc même page)
  - Données éloignées $\rightarrow$ forte probabilité de *hash* différents (donc pages différentes)

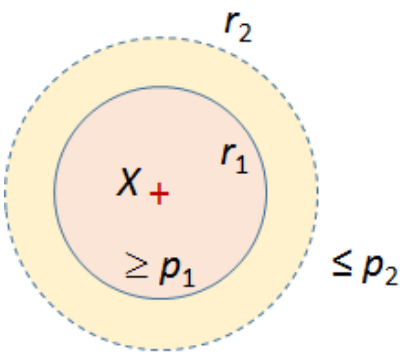
  $\Rightarrow$ Retourner les données similaires $\sim$ retourner les données de même *hash*, situées donc sur une même page

LSH : définition formelle

- Domaine $\mathcal{D}$ doté d'une métrique $d_{\mathcal{H}}$, ensemble de *hash* $\mathcal{Q}$
- $\mathcal{H} = \{h : \mathcal{D} \rightarrow \mathcal{Q}\}$ 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])

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

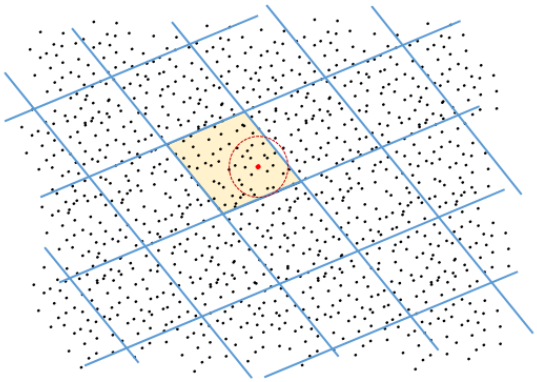
$\rightarrow$ Probabilité de collision élevée ($\geq p_1$) entre données proches ($d \leq r_1$), faible ($\leq p_2$) entre données éloignées ($d > r_2$)



Recherche par similarité avec LSH

- **En amont** (avant toute requête) : calculer le *hash* de chaque donnée, stocker les données de même *hash* dans une même page (*bucket*)
- **Pour chaque requête** $\mathbf{q}$: calculer $h(\mathbf{q})$, lire la page correspondante et retourner ses données (éventuellement filtrées par calcul de distances)

→ Complexité $O(1)$ par requête



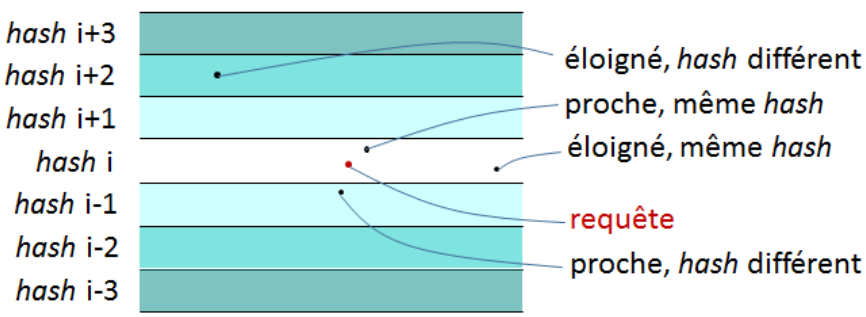
- Résultats approximatifs : données proches dans une page voisine non retournées (« faux négatifs »), données de même page mais éloignées retournées (« faux positifs »)

Familles de fonctions LSH : métrique euclidienne

- $\mathcal{D} \subset \mathbb{R}^m$, métrique L_2 : famille de fonctions (r_1, r_2, p_1, p_2)-sensibles

$$h_{\mathbf{a},b,w}(\mathbf{x}) = \left\lfloor \frac{\mathbf{a}^T \cdot \mathbf{x} + b}{w} \right\rfloor$$

- $\mathbf{a} \in \mathbb{R}^m$ de composantes tirées indépendamment suivant $\mathcal{N}(0,1)$; b uniforme dans $[0,1)$; $w \in \mathbb{R}^+$
- Géométriquement : chaque fonction élémentaire correspond à une famille de droites (hyperplans) parallèles qui découpent l'espace en bandes de largeur w :



Familles de fonctions LSH : distance cosinus

- Distance cosinus : $d_{\cos}(\mathbf{x}, \mathbf{y}) = \arccos \frac{\mathbf{x}^T \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$ – l'*angle* entre $\mathbf{x}$ et $\mathbf{y}$
- Fonctions élémentaires $h_{\mathbf{v}} : \mathbb{R}^m \to \{0, 1\}$

$$h_{\mathbf{v}}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x}^T \cdot \mathbf{v} \geq 0 \\ 0 & \mathbf{x}^T \cdot \mathbf{v} < 0 \end{cases}$$

  avec $\mathbf{v} \in \mathbb{R}^m$ tiré uniformément sur l'hypersphère unité ($\|\mathbf{v}\| = 1$)
- $\mathcal{H}_{\cos}$ est $(r_1, r_2, 1 - \frac{r_1}{180}, 1 - \frac{r_2}{180})$-sensible (angles en degrés)

→ Une fonction élémentaire associe 0 aux données d'un côté de l'hyperplan de vecteur normal $\mathbf{v}$ (passant par l'origine), 1 de l'autre côté

Familles de fonctions LSH : similarité de Jaccard (MinHash)

- Données = sous-ensembles d'un ensemble fini $\mathcal{E}$ (ex. texte → ensemble de mots ; profil d'achat → ensemble d'articles)
- Indice de Jaccard entre $\mathcal{A}, \mathcal{B} \in \mathcal{P}(\mathcal{E})$: $s_J(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|}$
- $d_J = 1 - s_J$ est une métrique !
- Fonctions élémentaires : π = permutation des éléments de $\mathcal{E}$, $h_\pi(\mathcal{A}) = \min \pi(\mathcal{A})$ (premier élément de $\mathcal{A}$ après permutation) → famille (r_1, r_2, p_1, p_2)-sensible

$\mathcal{E}$	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\pi(\mathcal{E})$	$\pi(\mathcal{A})$	$\pi(\mathcal{B})$	$\pi(\mathcal{C})$
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

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

MinHash : collision et indice de Jaccard

- Soit $x = |\mathcal{A} \cap \mathcal{B}|$ (éléments communs) et $y = |(\mathcal{A} - \mathcal{B}) \cup (\mathcal{B} - \mathcal{A})|$ (éléments spécifiques)
- Alors $s_J(\mathcal{A}, \mathcal{B}) = \dfrac{x}{x+y}$
- Pour toute fonction h_π, la probabilité de trouver en premier (après permutation) un élément commun plutôt que spécifique est $\dfrac{x}{x+y}$

$\Rightarrow$ Résultat clé : $p(h_\pi(\mathcal{A}) = h_\pi(\mathcal{B})) = s_J(\mathcal{A}, \mathcal{B})$

- La probabilité de collision est égale à l'indice de Jaccard entre les deux ensembles

$\rightarrow$ Base de l'algorithme **MinHash**, largement utilisé pour la détection de quasi-copies de documents (entre autres)

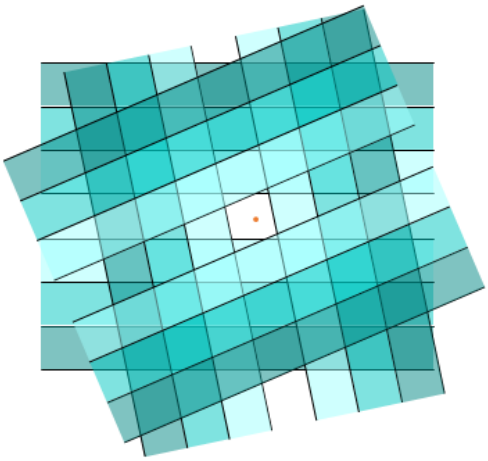
Composition ET (table de hachage)

- Une seule fonction élémentaire n'est en général pas assez sélective

$\rightarrow$ Regrouper n fonctions **indépendantes** en une table de hachage : *hash* = n-uplet des *hash* des fonctions individuelles

- Collision dans la table $\Leftrightarrow$ collision par rapport à chacune des n fonctions
- Probabilité de collision : s (une fonction MinHash) $\rightarrow s^n$ (n fonctions)

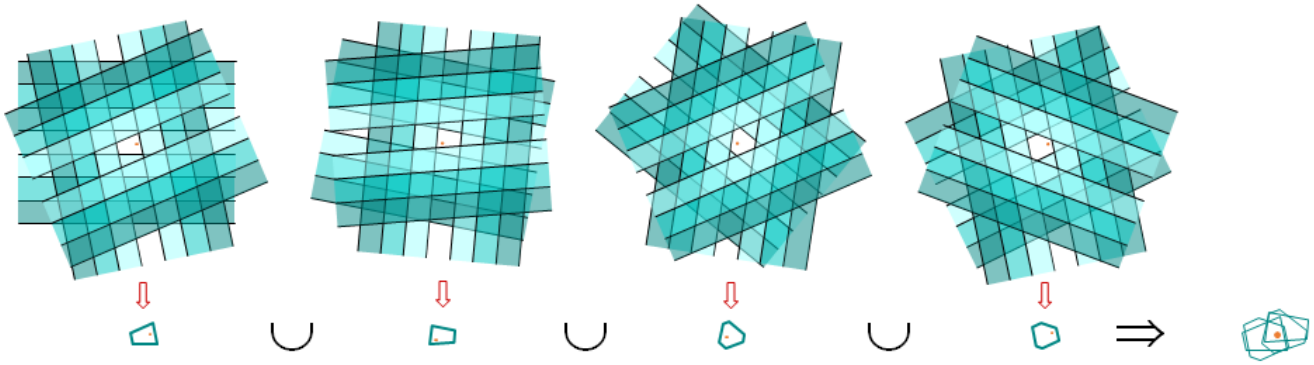
$\Rightarrow$ Réduit les faux positifs, mais augmente les faux négatifs



Composition OU (tables multiples)

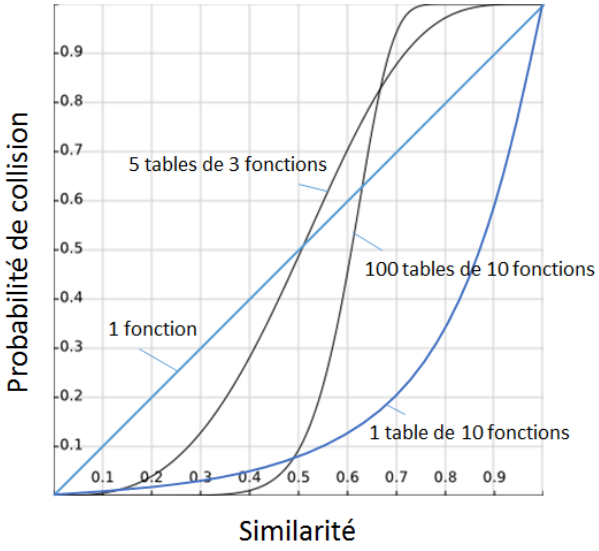
- Utiliser t tables **indépendantes**, retourner la réunion des résultats
  - Collision si collision dans au moins une table
  - Probabilité de collision : s^n (une table) $\rightarrow 1-(1-s^n)^t$ (t tables)

$\Rightarrow$ Augmente le rappel, au prix d'un espace de stockage et d'un temps de requête plus élevés



Effet d'amplification

- ET dans chaque table, OU entre les tables $\Rightarrow$ courbe de probabilité de collision qui tend vers une fonction à seuil
  - Les données de similarité $> s^*$ sont presque certainement retournées, les autres presque certainement non
  - Le seuil s^* s'ajuste en modifiant n et t (mais augmenter t augmente linéairement espace de stockage et temps de recherche)

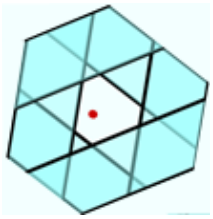


Généralisation de l'amplification et *Multi-probe LSH*

- Pour une famille dont les fonctions individuelles sont (r_1, r_2, p_1, p_2)-sensible, l'amplification rapproche p_2 de 0 et p_1 de 1 pour la fonction combinée
  - Fonctions individuelles (r_1, r_2, p_1, p_2)-sensibles
  - Combinaison AND de n fonctions ($h_{\texttt{AND}}$) : (r_1, r_2, p_1^n, p_2^n)-sensible
  - Combinaison OR de t fonctions ($h_{\texttt{OR}}$) : $(r_1, r_2, 1-(1-p_1)^t, 1-(1-p_2)^t)$-sensible
  - Combinaison AND et OR ($h_{\texttt{OR,AND}}$) : $(r_1, r_2, 1-(1-p_1^n)^t, 1-(1-p_2^n)^t)$-sensible

- ***Multi-probe LSH*** [5] : pour limiter le nombre de tables tout en gardant un bon rappel, interroger aussi les pages voisines (taux d'échantillonnage décroît avec la distance)

  → À rappel constant, réduit le nombre de tables d'un ordre de grandeur



Plan du cours

Malédiction de la dimension

- Les données massives ont souvent un nombre élevé de variables, parfois représentées en très grande dimension (variable nominale à nombreuses modalités, variable « ensemble » de grande cardinalité…)
- La grande dimension engendre des difficultés concernant la modélisation statistique et l'efficacité des index/recherches : « malédiction » ou « fléau » de la dimension (*curse of dimensionality*)

Difficulté 1 : dilution de la densité

- À nombre de données fixé, la densité diminue exponentiellement avec la dimension

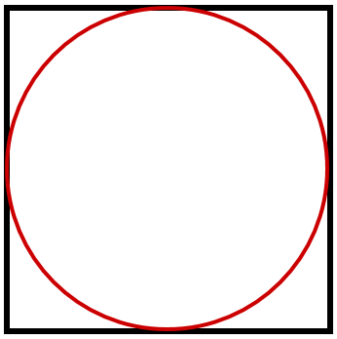
→ Les données deviennent « rares », « isolées » dans l'espace

→ Estimation de densité peu fiable, tests statistiques inexploitables

⇒ Le nombre de données devrait croître exponentiellement avec la dimension pour conserver les capacités de modélisation

Difficulté 2 : les données se concentrent près des surfaces externes

- Les données uniformément distribuées dans un volume de dimension d sont proches des hypersurfaces externes (de dimension $d-1$)
- Rapport volumes hypersphère / hypercube englobant diminue rapidement avec d



Dim.	Volume sphère / cube
1	1
2	0,78732
4	0,329707
6	0,141367
8	0,0196735
10	0,00399038

$\Rightarrow$ Avec LSH pour la distance euclidienne (par exemple), la plupart des données d'une page (hyper-parallélépipède) ne sont pas dans l'hypersphère de recherche $\Rightarrow$ filtrage coûteux, coût qui augmente avec d

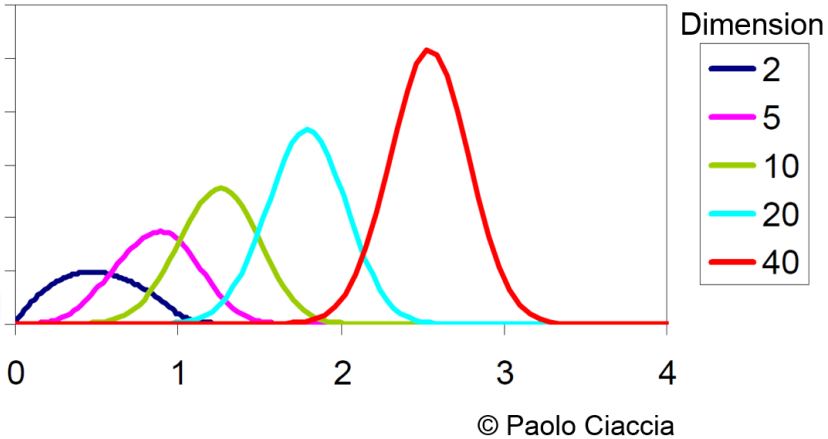
Difficulté 3 : concentration des mesures

- La variance de la distribution des distances entre données diminue avec la dimension

$\rightarrow$ Décision par k plus proches voisins moins fiable (les voisins ne sont plus significativement plus représentatifs que les autres données)

$\rightarrow$ Regroupements moins nets $\rightarrow$ intérêt réduit de la classification automatique

$\Rightarrow$ L'hypothèse « les données similaires sont nettement plus similaires que les autres » devient fausse $\Rightarrow$ les index perdent leur efficacité



Facteurs modérateurs

1. **Distribution des données** : plus elle est non uniforme, plus élevée est la dimension à partir de laquelle modélisation et index se dégradent nettement
2. **Dimension intrinsèque** : la dimension qui compte peut être bien plus faible que la dimension apparente
   - Réductible par des méthodes linéaires (ACP, AFD…) ou non linéaires (t-SNE, UMAP…)
   - Plusieurs définitions formelles, certaines avec estimateurs opérationnels

Plan du cours

Motivations et contexte

- Avec les *transformers* et les LLM, des objets hétérogènes (textes, images, sons, graphes) sont représentés par des **vecteurs denses de grande dimension** (512 à 4096) → on parle de plongements vectoriels (*embeddings*)
- Produits par un modèle (*encoder*), reflètent une similarité sémantique : objets proches sémantiquement ⇔ vecteurs proches (distance cosinus ou euclidienne)

→ Bases de données **vectorielles** : indexer et interroger efficacement (requêtes ANN) de grandes collections de vecteurs → utilisables partout où une bonne technique d'encodage est disponible

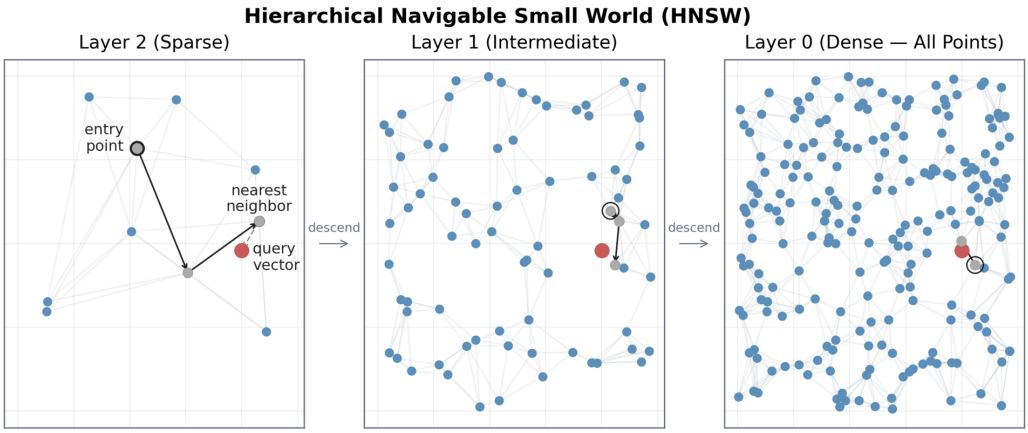
Structures d'index ANN : LSH, IVF, PQ, ScaNN

- **LSH** : une des premières approches pour ANN ; complexité sous-linéaire, facilité de mise en œuvre distribuée → adapté aux très grandes bases
- **IVF** (*Inverted File Index*) : quantification préalable en k centroïdes (*k-means*), chaque vecteur assigné au centroïde le plus proche
  - Requête : identifier les centroïdes les plus proches (*nprobe*), recherche exhaustive parmi leurs vecteurs
  - Qualité très dépendante du choix de k et *nprobe*
  - Implémentation pour Spark dans Spark Rapids ML
- **PQ** (*Product Quantization* [4]) : découpe chaque vecteur en m sous-vecteurs de dimension d/m, quantifiés indépendamment (2^b codes sur b bits)
  - Stockage sur $m \times b$ bits au lieu de $d \times 32$; distances approximatives rapides ; souvent combiné à IVF (IVF-PQ)
- **ScaNN** (Google, [3]) : quantification anisotrope + rééquilibrage des partitions ; utilisé en production pour des milliards de vecteurs

Structures d'index ANN : *Hierarchical Navigable Small World* (HNSW, [6])

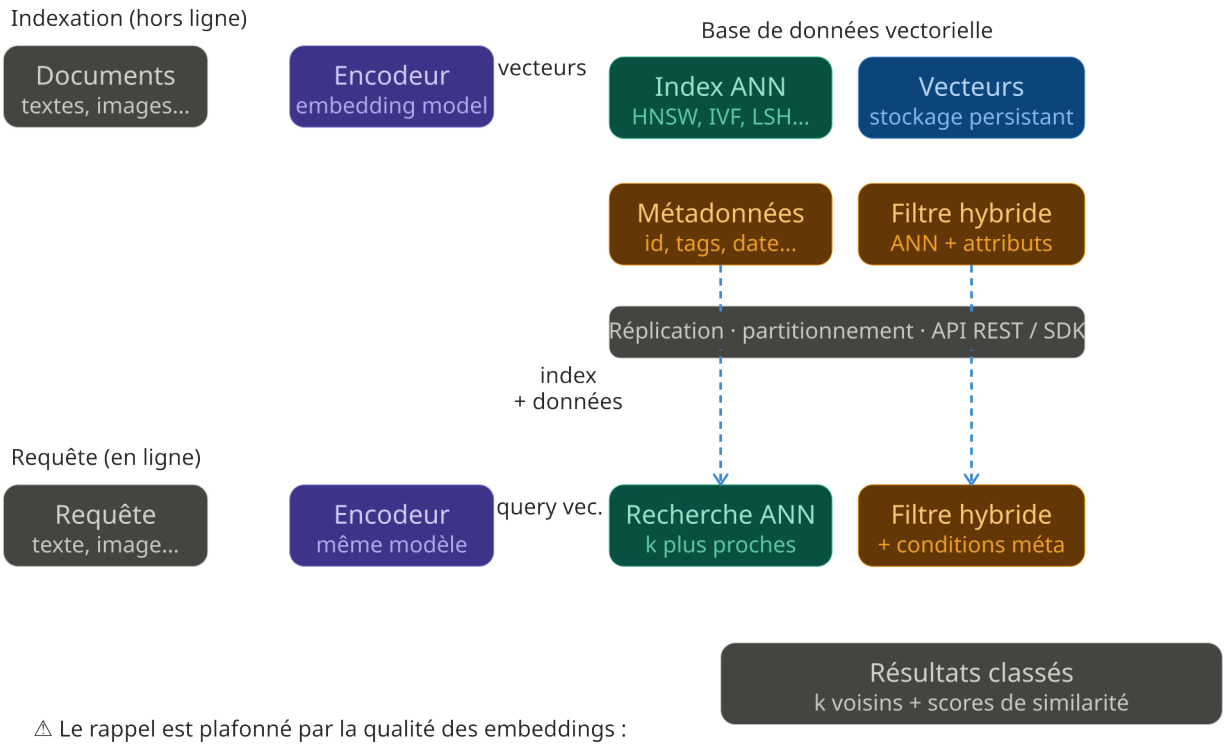
- Graphe hiérarchique à plusieurs niveaux de résolution : tous les points dans couche 0, sous-ensembles aléatoires dans couches supérieures, connectivité kNN intra-couche
- Recherche : descente gloutonne depuis un point d'entrée fixe en couche haute, affinage progressif suivant liens intra-couche jusqu'à la couche 0
- Employé dans Faiss [1], Milvus, PGVector (PostgreSQL), etc. Pour Spark : Hnswlib

→ 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



(source de l'image)

Architecture d'une base de données vectorielles



⚠ Le rappel est plafonné par la qualité des embeddings : un encodeur inadapté au domaine dégrade les résultats indépendamment de la qualité de l'index ANN.

Architecture d'une base de données vectorielles

- **Stockage persistant** des vecteurs et métadonnées (identifiant, attributs filtrables, horodatage)
- **Filtrage hybride** : combiner condition sur métadonnées et recherche vectorielle, sans perte majeure de performance
- **Mise à jour en ligne** (*upsert*, suppression) : index ANN difficiles à mettre à jour incrémentalement
- **Réplication et partitionnement** (*sharding*) pour passer à l'échelle sur un *cluster*
- **Interface de requête** (API REST, SDK Python), intégration IA (LangChain, LlamaIndex)

Solutions actuelles

- **Bases de données vectorielles natives** : Pinecone, Weaviate, Qdrant, Milvus/Zilliz
  - Conçues dès le départ pour la recherche vectorielle, fonctionnalités avancées (filtrage, réplication, API cloud)
- **Extensions vectorielles de SGBD existants** : PGVector (PostgreSQL), Redis Search, Elasticsearch *dense vector*
  - Recherche vectorielle dans une infrastructure existante, au prix d'une moindre performance à très grande échelle
- **Bibliothèques d'indexation** : Faiss (Meta), Annoy (Spotify), ScaNN (Google)
  - Bas niveau, sans persistance ni métadonnées ; souvent utilisées comme moteur d'indexation dans les bases natives

Compromis précision / vitesse / mémoire

- Métriques de performance :
  - **Rappel** (*recall@k*) : proportion des vrais k plus proches voisins effectivement retournés
  - **Latence** de requête (ou **débit** pour un flux de requêtes)
  - **Empreinte mémoire** : taille de l'index en RAM

→ Métriques interdépendantes : augmenter le rappel nécessite en général plus de latence (ou moins de débit) à mémoire donnée

- Le choix du bon index et de ses paramètres exige une évaluation empirique (taille de base, distribution des données, ressources disponibles…)
- Comparaison systématique des principaux algorithmes : ANN Benchmarks

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.


R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar.
Accelerating large-scale inference with anisotropic vector quantization.
In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.


H. Jegou, M. Douze, and C. Schmid.
Product quantization for nearest neighbor search.
IEEE Trans. Pattern Anal. Mach. Intell., 33(1) :117–128, 2011.

Références II


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.