

Recherche par similarité

G.Vanwormhoudt

UV OData - 2021

Plan

1. Introduction : définitions, applications et problèmes
2. Recherche exactes à base d'arbres
3. Recherche approchée par hachage
4. Autres approches
5. Conclusion

Introduction

Essor du BigData

- De nouvelles capacités technologiques
 - Des capacités réseaux décuplées grâce à l'internet et aux réseaux nouvelle génération (anywhere, anytime)
 - Des capacités de stockage quasi illimitées à un coût abordable
 - L'augmentation de la puissance de calculs disponibles (cluster, GPU, cloud)
 - La miniaturisation des processeurs (terminaux mobiles, capteurs)
- Développement de l'informatique connectée et des objets intelligents
 - Fourniture de contenu par les utilisateurs (texte, image, video,via le web ou les mobiles)
 - Fourniture de contenu par les objets intelligents (internet des objets, compteurs électriques, capteurs, ... toute source d'évènements)



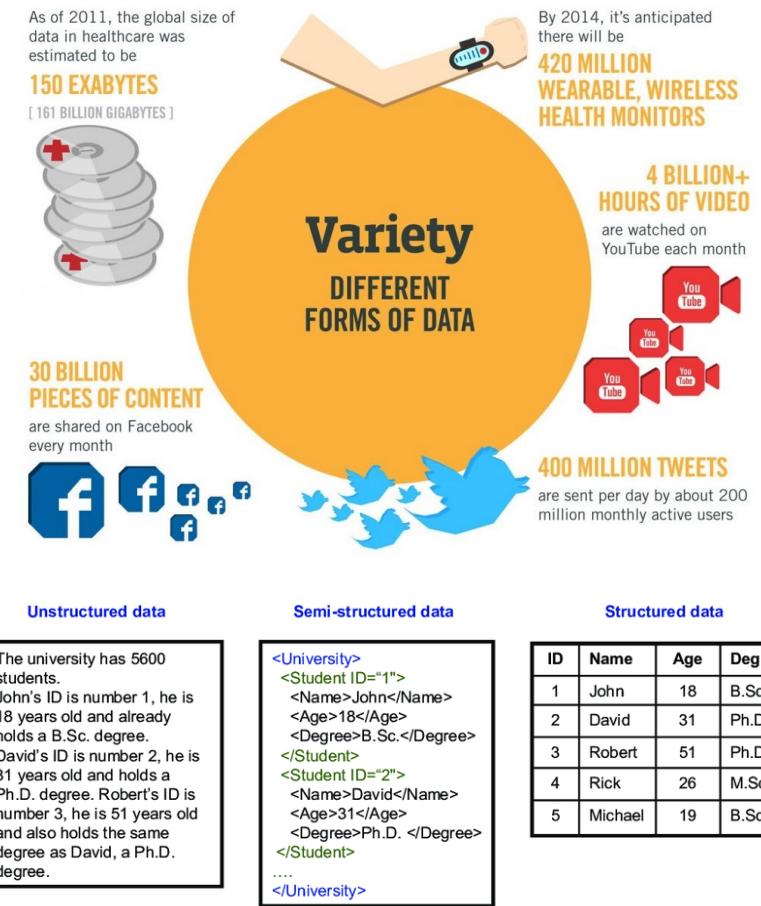
Données en 2012 (source practicalanalytics)

2019 This Is What Happens In An Internet Minute



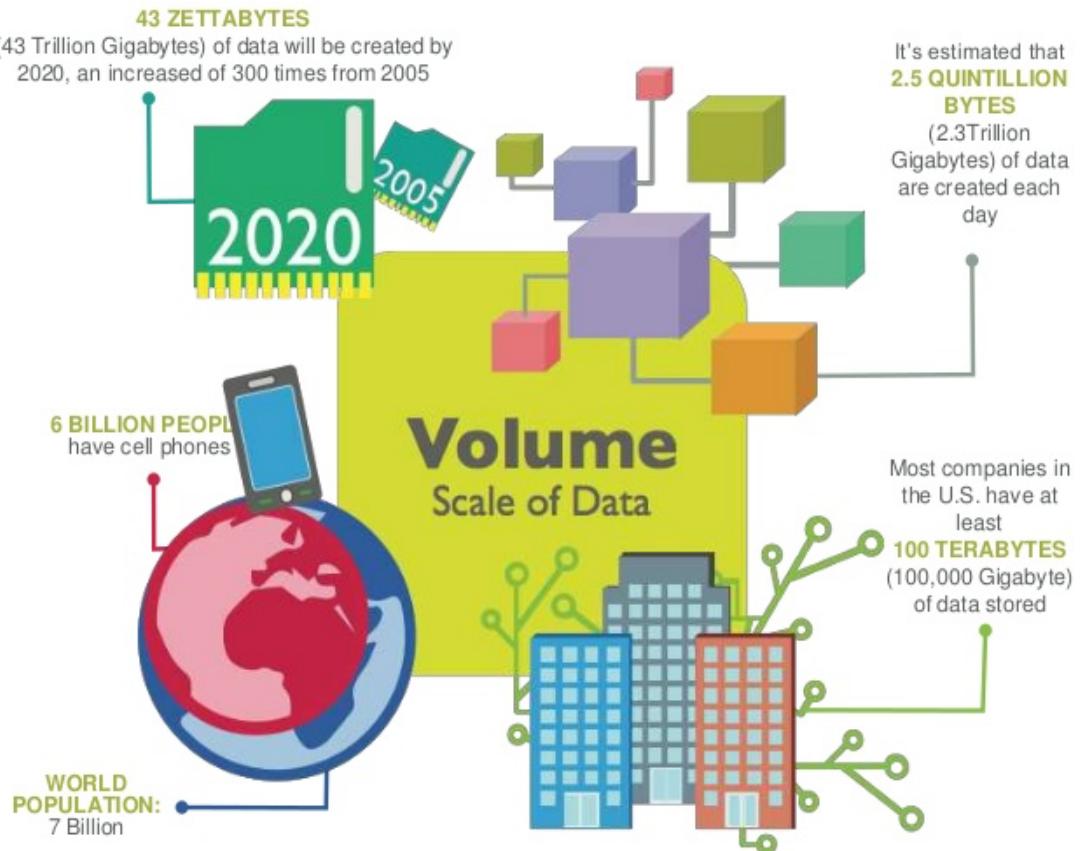
Variété des données

- Une grande variété des données :
 - brutes, non structurées, semi-structurés, structurées
 - Données issues du Web (pages Web, click utilisateurs, publicités ciblées, sites visités, durée de visionnage, ...)
 - Des capteurs (météo, finance, géolocalisation GPS, adresse IP, puce RFID, montre connectées, véhicules connectés, bâtiments intelligents, logistiques, ...)
 - Des textes (blog, tweet, avis des réseaux sociaux, avis de site e-commerce, emails...)
 - Des objets multimédia: son et parole (podcast), image (photos), vidéo, 3D
- Des données de plus en plus complexe
 - Dimensions spatiales et temporelles des données publiées à prendre en compte
 - Liens entre données : Graphes
 - Grandes dimensions : Vecteurs de caractéristiques, Descripteurs (Algo ML/IA)



Volume des données

- L'ordre de grandeur des données numériques créées dans le monde est le zetaoctets (40 ZB en 2020 ~ 43×10^{18})
- Raisons
 - Accroissement rapide des appareils connectés (capteur physiques, internet des objets)
 - Augmentation du nombre de données produites par un utilisateur (texte, image, vidéo)
 - Augmentation de la fréquence d'observation (compteur électrique, quantified self, ...)

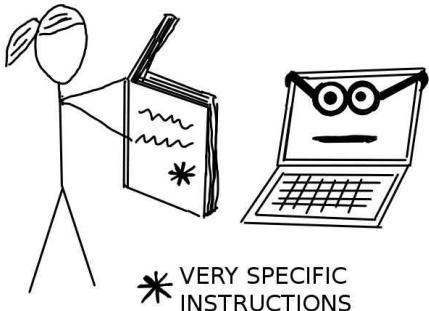


Essor de l'IA

- Capitalisation sur les capacités apportées par le Big Data
 - Production et collecte de gros volumes de données
 - Stockage de gros volumes de données
- Augmentation des capacités de traitements des données
 - Augmentation exponentielle de la puissance de calcul des ordinateurs
 - GPU, Cloud, Clusters de machines/GPU, Modèles de programmation parallèle
 - Amélioration et disponibilité des algorithmes d'apprentissage (Machine Learning)
 - Nouvelles approches d'apprentissage et nouvelles architectures logicielles : réseaux de neurones profonds
 - Mise à disposition de données étiquetées pour entraîner les modèles avec les algorithmes d'apprentissage
 - Mise à disposition de bibliothèques : scikit-learn, tensorflow, pytorch

Machine Learning

Without Machine Learning



With Machine Learning



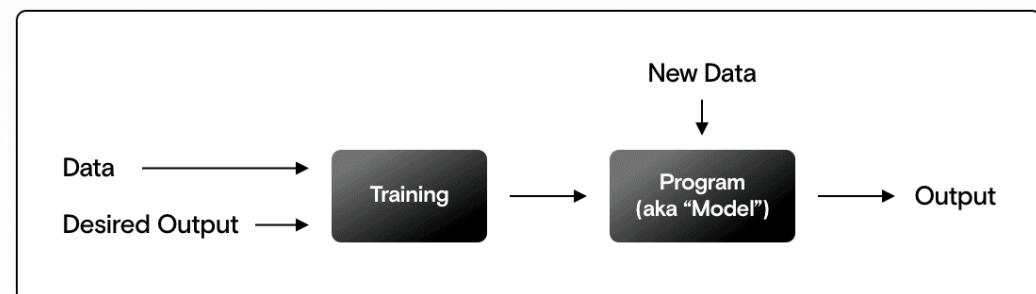
Traditional Programming

Developers write rules (program) that produce an output.



Machine Learning

Developers write a training algorithm, that finds rules, which produce the desired output.



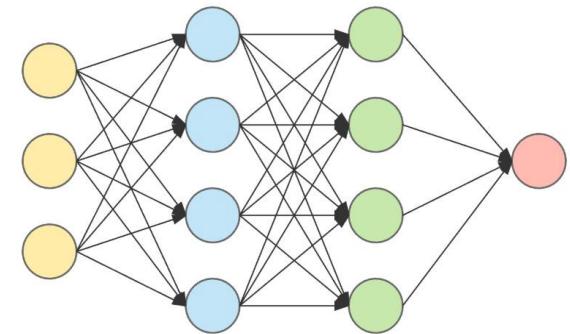
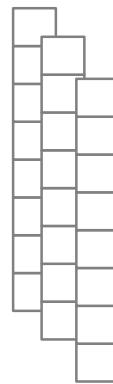
Données de l'IA (ML)

- Les algorithmes d'apprentissage (classique et profond) prennent en entrées des vecteurs, matrices, tenseurs (binaire, entier, réel, ...)
- Nécessité de transformer les données brutes en une représentation acceptable par les algorithmes
- L'ingénierie des caractéristiques s'intéresse à cette transformation de façon conceptuelle et programmatique
 - Conceptualisation des caractéristiques utiles à l'apprentissage (de façon directe ou indirecte)
 - Sélection, Aggrégation, Synthèse, Apprentissage des caractéristiques
 - Schéma de transformation des exemples bruts
 - Stockage des caractéristiques
- Nécessite de la créativité, du savoir-faire, de l'expérience mais l'apprentissage profond apporte de nouvelles solutions

Exemple du Texte

Date collected	Plot	Species	Sex	Weight
1/9/78	1	DM	M	40
1/9/78	1	DM	F	36
1/9/78	1	DS	F	135
1/20/78	1	DM	F	39
1/20/78	2	DM	M	43
1/20/78	2	DS	F	144
3/13/78	2	DM	F	51
3/13/78	2	DM	F	44
3/13/78	2	DS	F	146

transformation



UNITED STATES

Obama tried to give
Zuckerberg a wake-up call
over fake news on Facebook

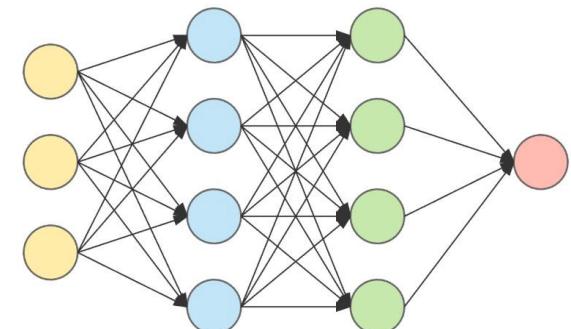
The Washington Post - By Adam Entous, Elizabeth...

Economy September 25, 2017 at
7:45 AM Facebook CEO Mark
Zuckerberg's company recently said
it would turn over to Congress more
than 3,000 politically themed

transformation

?

Apprentissage



Caractéristiques du Texte

- Caractéristiques observées

- ▶ Mots

- taille, préfix, suffixe, capital

- n-grams de caractères

- ...

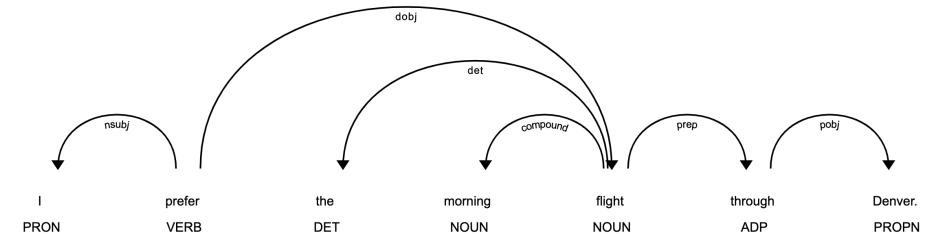
- ▶ Ensemble de mots

- présence, comptage et poids des mots

- séquence des mots: n-grams, colocalisation

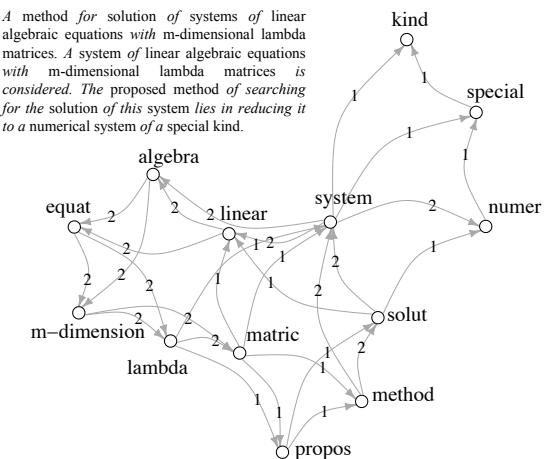
- ...

- Caractéristiques inférées



TEXT	LEMMA	POS	TAG	DEP
Apple	apple	PROPN	NNP	nsubj
is	be	VERB	VBZ	aux
looking	look	VERB	VBG	ROOT
at	at	ADP	IN	prep
buying	buy	VERB	VBG	pcomp
U.K.	u.k.	PROPN	NNP	compound
startup	startup	NOUN	NN	dobj
for	for	ADP	IN	prep
\$	\$	SYM	\$	quantmod
1	1	NUM	CD	compound
billion	billion	NUM	CD	pobj

A method for solution of systems of linear algebraic equations with m-dimensional lambda matrices. A system of linear algebraic equations with m-dimensional lambda matrices is considered. The proposed method of searching for the solution of this system lies in reducing it to a numerical system of a special kind.



Sac de mots - Encodage One-Hot

Document 1	Love, love is a verb
Document 2	Love is a doing word
Document 3	Feathers on my breath
Document 4	Gentle impulsion
Document 5	Shakes me, makes me lighter
Document 6	Feathers on my breath



Document 1	[Love, love, is, a, verb]
Document 2	[Love, is, a, doing, word]
Document 3	[Feathers, on, my, breath]
Document 4	[Gentle, impulsion]
Document 5	[Shakes, me, makes, me, lighter]
Document 6	[Feathers, on, my, breath]



a breath doing feathers
 gentle impulsion is lighter
 love makes me my
 on shakes verb word

	a	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	word
Document 1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	
Document 2	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	
Document 3	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	
Document 4	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
Document 5	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	0	
Document 6	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	

a	breath	doing	feathers	gentle	impulsion	is	lighter	love	makes	me	my	on	shakes	verb	word
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Conversion des étiquettes en nombres

- Concerne les attributs catégoriels
- Méthodes
 - Recherche et remplacement
 - four => 4, six => 6, ...
 - Encodage d'étiquettes
 - Association d'un entier à chaque étiquette et remplacement par l'entier
 - Création d'une colonne binaire par étiquette (ex: colonne drive_wheels)

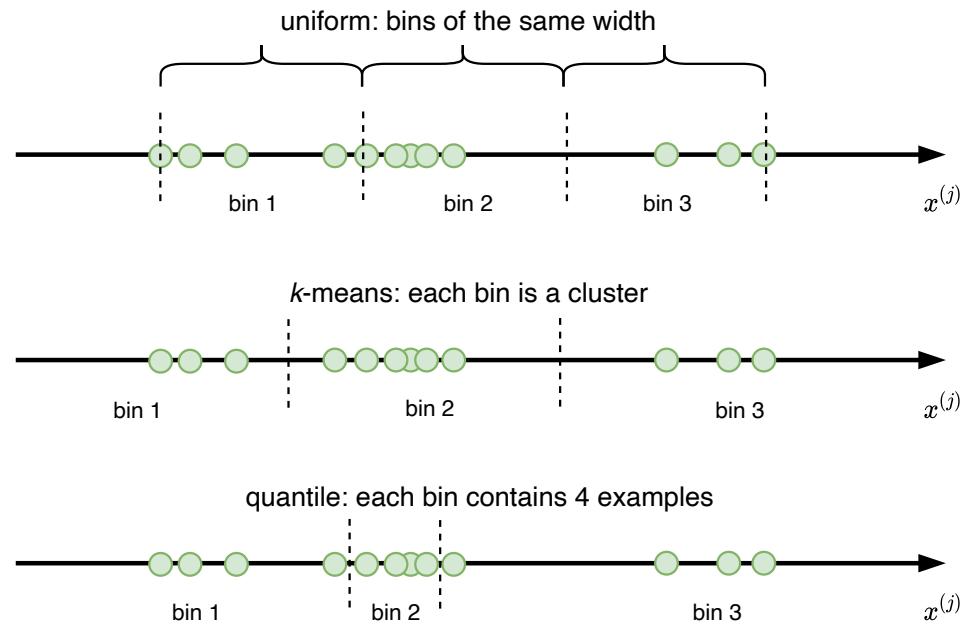
	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engine_type	num_cylinders	fu
0	alfa-romero	gas	std	two	convertible	rwd	front	dohc	four	m
1	alfa-romero	gas	std	two	convertible	rwd	front	dohc	four	m
2	alfa-romero	gas	std	two	hatchback	rwd	front	ohcv	six	m
3	audi	gas	std	four	sedan	fwd	front	ohc	four	m
4	audi	gas	std	four	sedan	4wd	front	ohc	five	m

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engine_type	num_cylinders	fu
0	alfa-romero	gas	std	2	convertible	rwd	front	dohc	4	m
1	alfa-romero	gas	std	2	convertible	rwd	front	dohc	4	m
2	alfa-romero	gas	std	2	hatchback	rwd	front	ohcv	6	m
3	audi	gas	std	4	sedan	fwd	front	ohc	4	m
4	audi	gas	std	4	sedan	4wd	front	ohc	5	m

.style_cat	body_convertible	body_hardtop	body_hatchback	body_sedan	body_wagon	drive_4wd	drive_fwd	drive_rwd
	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Discrétisation de valeurs numériques

- Regroupement en groupes de valeurs discrets
 - Groupes d'âge
 - Groupe selon des tranches de salaire
- Peut conduire à une meilleure précision de prédiction
- Ajoute des informations utiles à l'algorithme d'apprentissage lorsque l'ensemble de données d'apprentissage est relativement petit.
- Peut-être plus simple pour un humain d'interpréter la prédiction
- Techniques de Binning



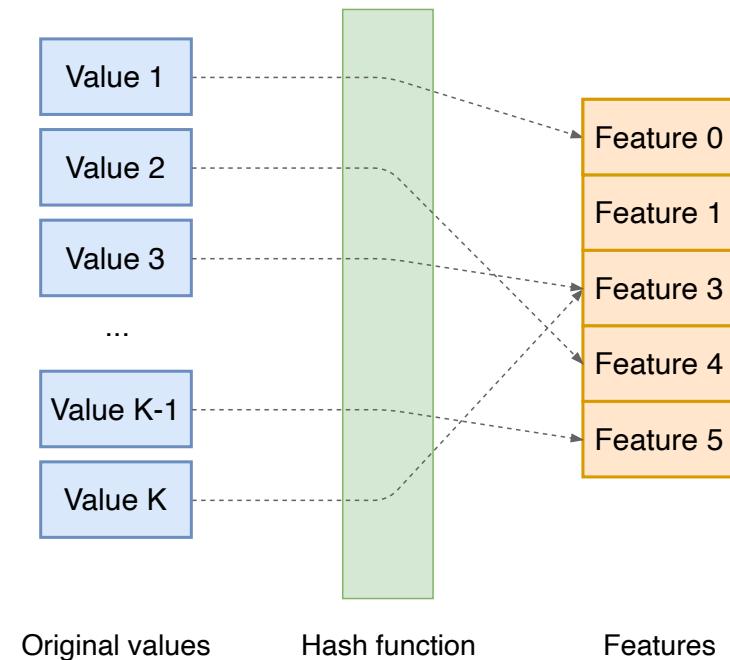
Hachage de caractéristiques

- Utile pour les données textuelles ou les attributs catégoriels avec de nombreuses valeurs
- Objectif: limiter le nombre de dimensions
- Principe:
 - Conversion de toutes les valeurs de l'attribut catégoriel (ou tous les tokens de la collection de documents) en un nombre avec une fonction de hachage
 - Conversion du nombre en un index du vecteur de caractéristiques

$h(\text{love}) \bmod 5 = 0$
 $h(\text{is}) \bmod 5 = 3$
 $h(\text{a}) \bmod 5 = 1$
 $h(\text{doing}) \bmod 5 = 3$
 $h(\text{word}) \bmod 5 = 4.$

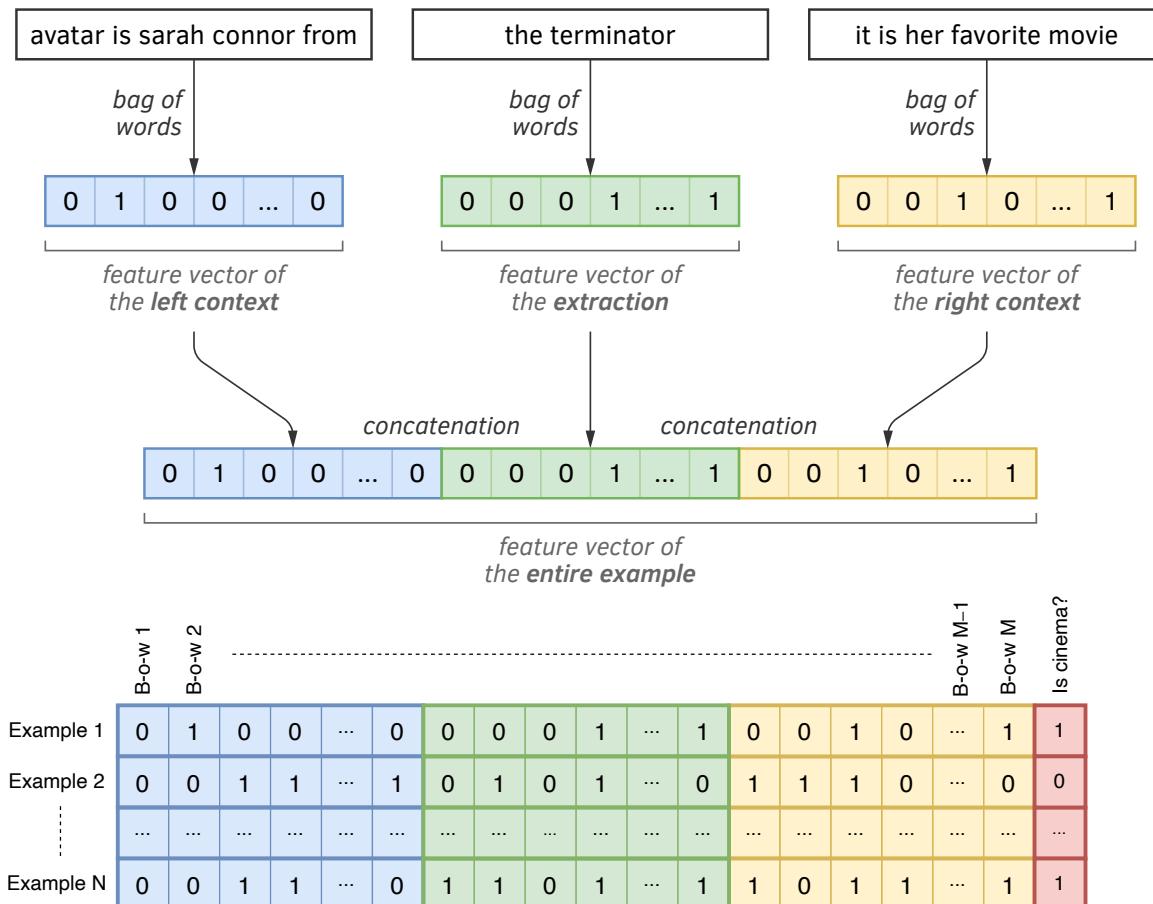
“Love is a doing word”
[1, 1, 0, 2, 1]

a	breath	doing	feathers	gentle	impulsion	is	lighter	love	makes	me	my	on	shakes	verb	word
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



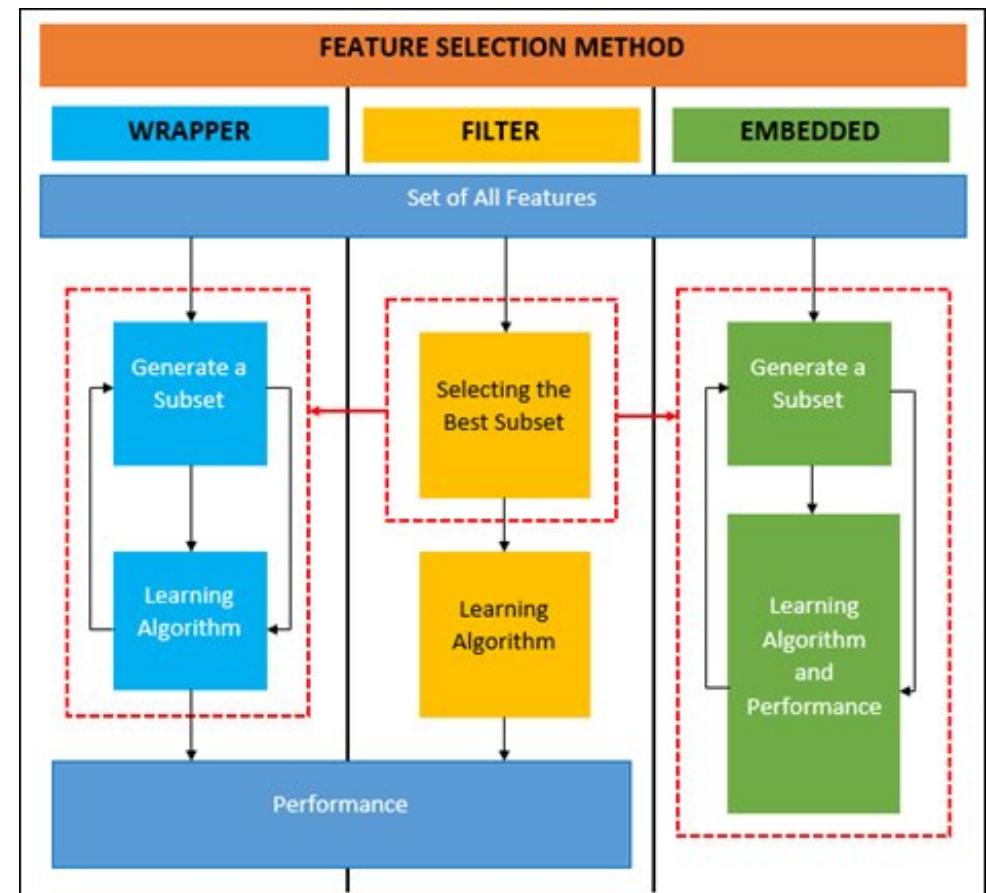
Empilement de caractéristiques

- Combinaison de plusieurs textes
 - ex: requête / réponse, texte + sujet
- Combinaison de données et du texte
 - ex: attributs d'un produit accompagné de l'avis utilisateur
- L'ordre n'est pas important en général
- Respect du même ordre pour tous les exemples



Réduction des caractéristiques

- Objectifs
 - Eviter un vecteur de caractéristiques trop large qui peut conduire à des temps d'apprentissage prohibitif
 - Diminuer la taille globale des données pour tenir dans la mémoire vive
- Plusieurs méthodes
 - Eliminer les caractéristiques redondantes (mesure de corrélation)
 - Aggréger plusieurs caractéristiques en une seule
 - Méthodes trouvant les caractéristiques les plus importantes:
 - Evaluation de la corrélation entre chaque variable d'entrée et le résultat en sortie
 - Sélection de sous-ensemble et évaluation des résultats
 - Réduction de dimensions : PCA par exemple



Synthèse de caractéristiques

- Aider l'algorithme d'apprentissage à apprendre en lui fournissant un ensemble plus riche de caractéristiques
- Construction
 - ▶ A partir des données relationnelles
 - Statistiques sur des colonnes de tables en lien avec la donnée
 - ▶ A partir des autres caractéristiques
 - Discréétisation (vue précédemment)
 - Elévation au carré, Moyenne à partir des plus proches voisins, opérations arithmétiques

User				
User ID	Gender	Age	...	Date Subscribed
1	M	18	...	2016-01-12
2	F	25	...	2017-08-23
3	F	28	...	2019-12-19
4	M	19	...	2019-12-18
5	F	21	...	2016-11-30

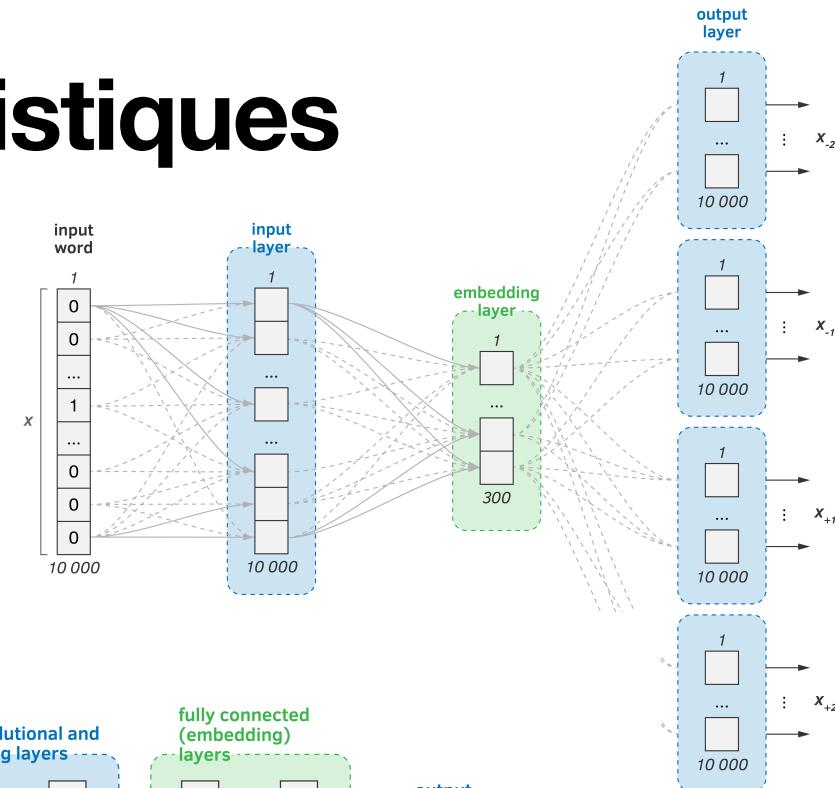
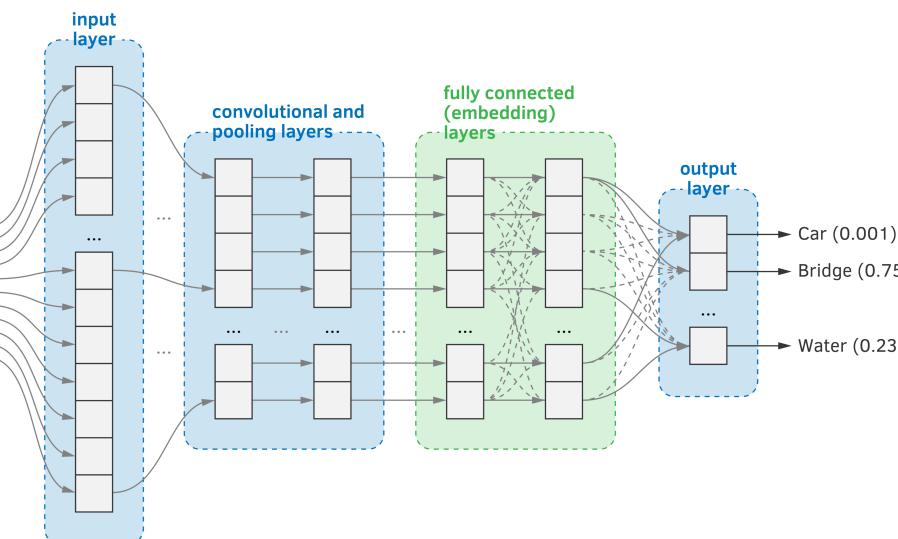
Order				
Order ID	User ID	Amount	...	Order Date
1	2	23	...	2017-09-13
2	4	18	...	2018-11-23
3	2	7.5	...	2019-12-19
4	2	8.3	...	2016-11-30

Call				
Call ID	User ID	Call Duration	...	Call Date
1	4	55	...	2016-01-12
2	2	235	...	2016-01-13
3	3	476	...	2016-12-17
4	4	334	...	2019-12-19
5	4	14	...	2016-11-30

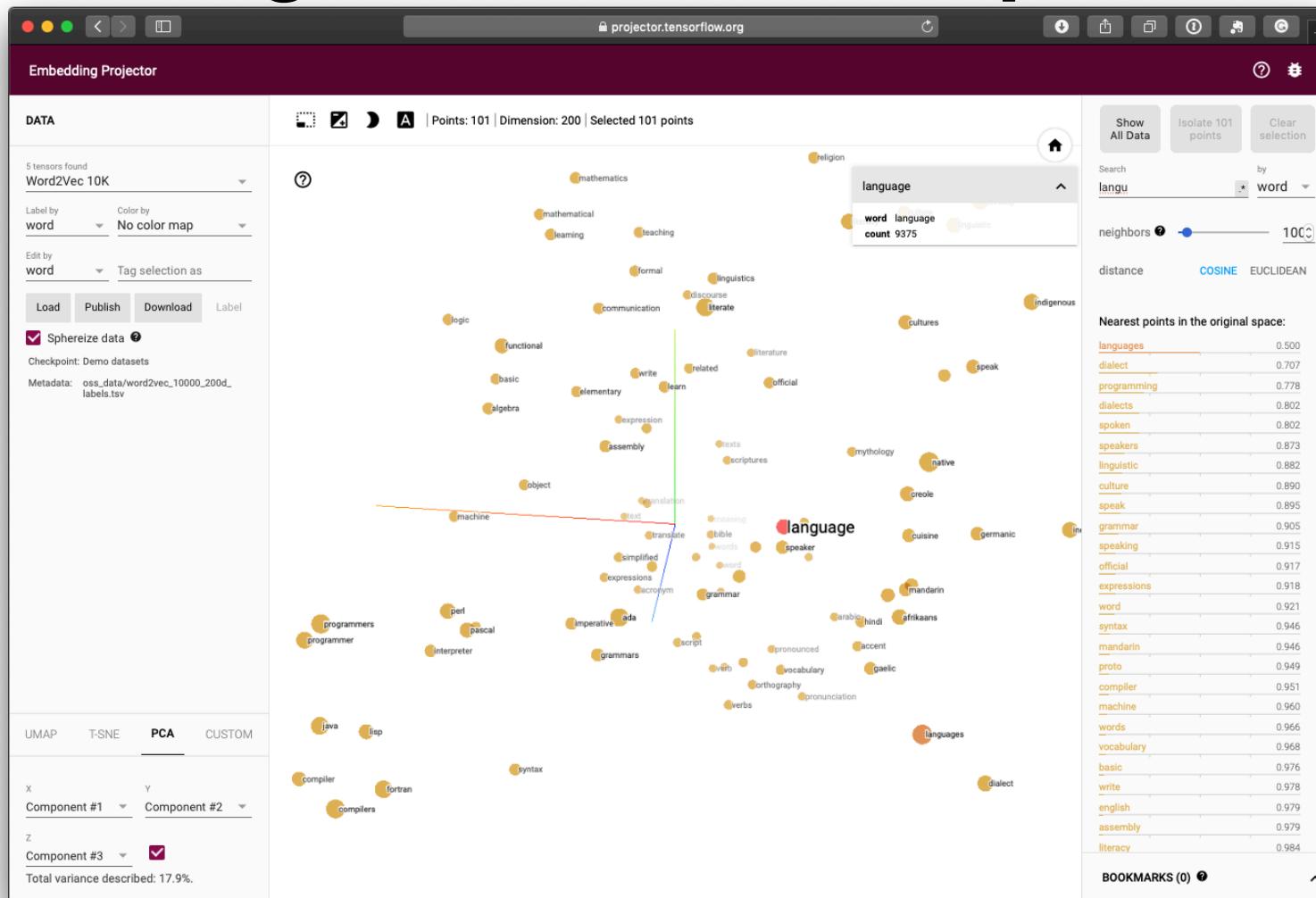
User features						
User ID	Gender	Age	Mean Order Amount	Std Dev Order Amount	Mean Call Duration	Std Dev Call Duration
2	F	25	12.9	7.1	235	0
4	M	19	18	0	134.3	142.7

Apprentissage des caractéristiques

- Il est possible d'apprendre des caractéristiques en exploitant les relations (non linéaires) dans les données
 - cooccurrence de mots, prédiction du prochain mot
 - motifs de région similaires dans les images
- Technique très en vogue actuellement (Word Embedding, Image Embedding, ...)
 - promu par réseaux de neurones profonds qui facilitent ce type d'apprentissage (sorties près de la couche supérieure)
 - fonctionne bien pour tout type de donnée
 - Text
 - Image
 - Son
 - ...
- Les caractéristiques apprises ne sont pas interprétables

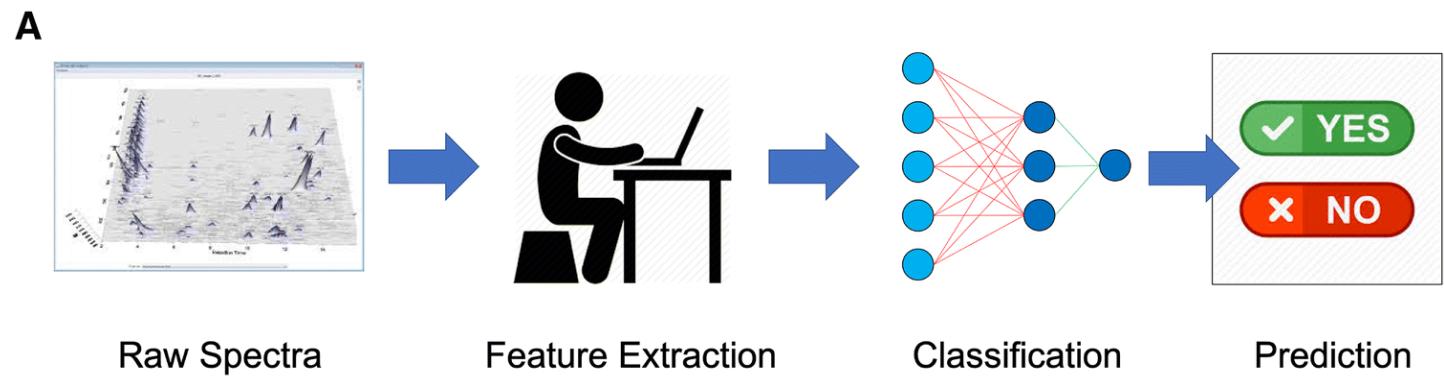


Apprentissage de caractéristiques



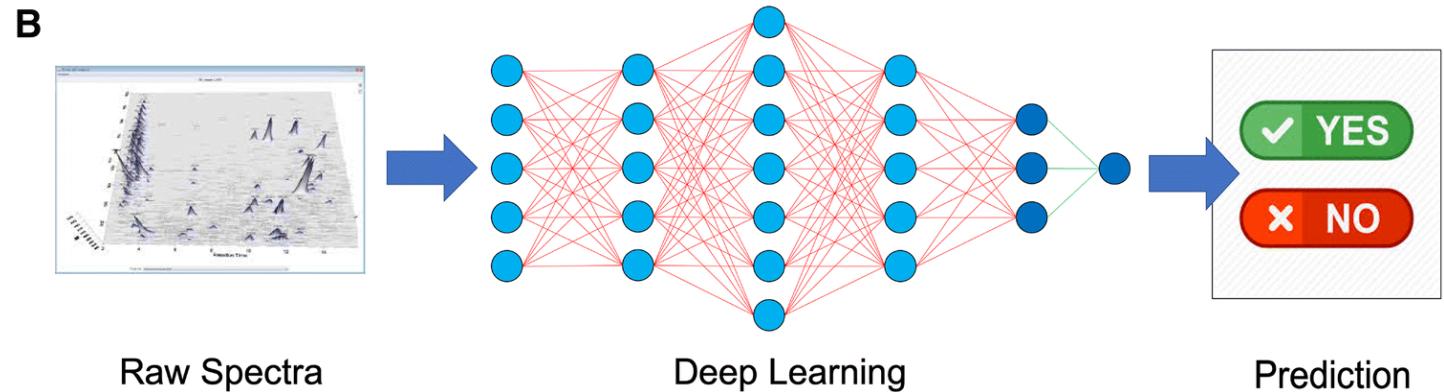
Ingénierie des caractéristiques & Deep Learning

Machine Learning classique



Deep Learning

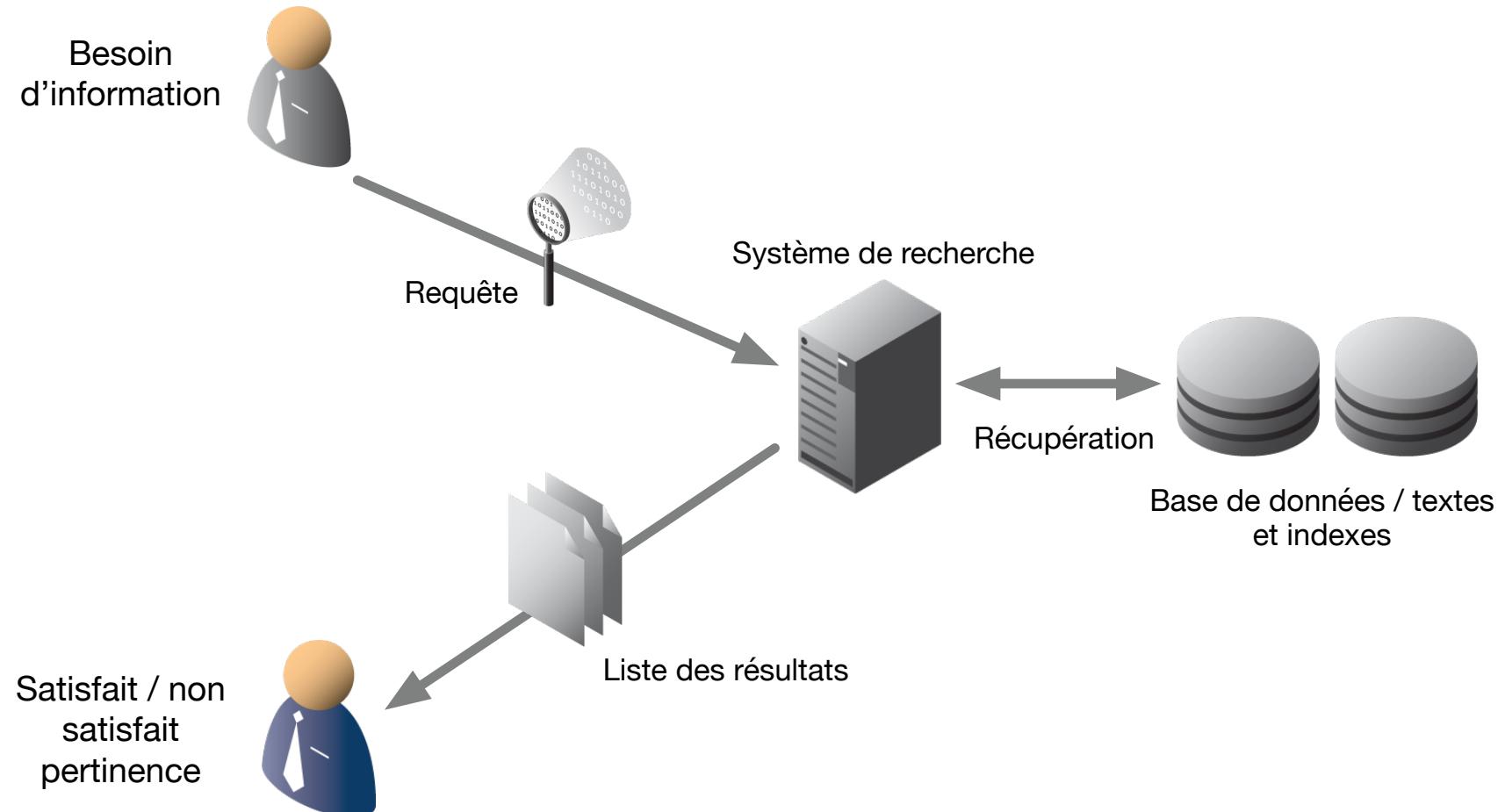
Les caractéristiques intéressantes des données sont apprises automatiquement au lieu d'être spécifiés et extraites par un expert (souvent trop spécifiques, incomplètes, longues à concevoir et à valider)



Focus du cours

- Avec l'essor du BigData, de la Data Science et de l'IA
 - De plus en plus de données multi-dimensionnelles à traiter et à stocker
 - Les dimensions ont du sens prises ensemble
 - Transformation en vecteurs/matrices/tenseurs pour les algorithmes utilisés en Datascience et en IA (ML) -> transformation des données en valeurs numériques
- Des besoins
 - d'exploration, de visualisation, de recherche, de comparaison, de corrélation, de regroupement, d'interprétation, de ces données multi-dimensionnelles
- Suite du cours : focus sur la recherche d'informations

Recherche d'informations



Recherche d'informations

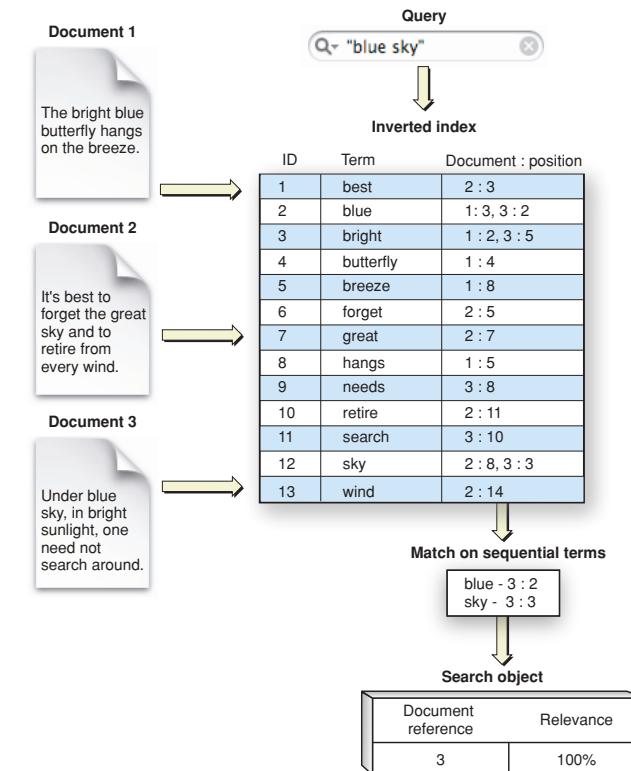
- Données tabulaires (BD relationnelle)
 - Langage SQL

```
select BookID, Title from Books where
IntStock > 4
```

- Données textuelles
 - Utilisation de mots-clés et d'opérateurs booléens

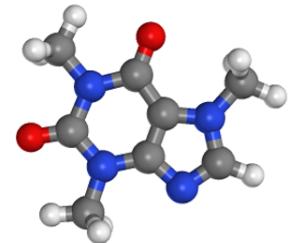
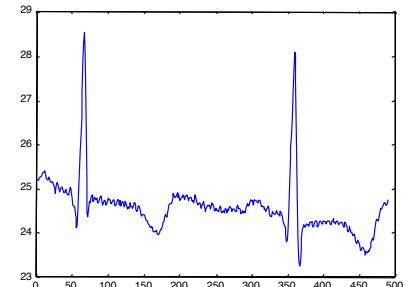
(blue OR great) AND sky

BookID	Author	Title	Price	InStock
12458	Baudelaire	Les fleurs du mal	8.99	6
32658	Dick	Ubik	6.99	25
19625	Vapnik	The Nature of Statistical Learning	55.00	3
11785	Pratchett	Small Gods	12.49	0
21915	Vian	L'arrache-cœur	6.25	17
14219	Knuth	The Art of Computer Programming	97.90	4

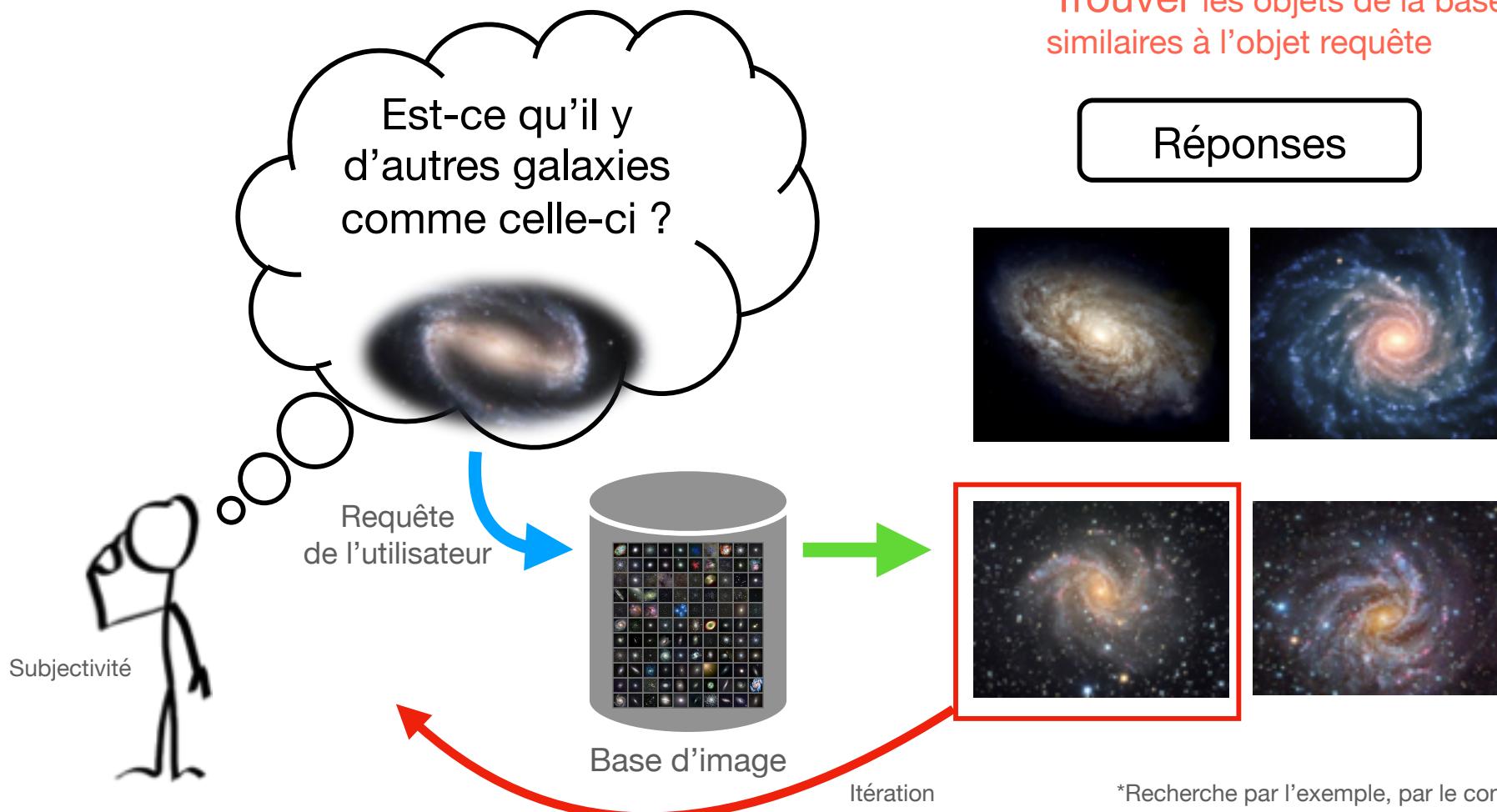


Recherche d'information

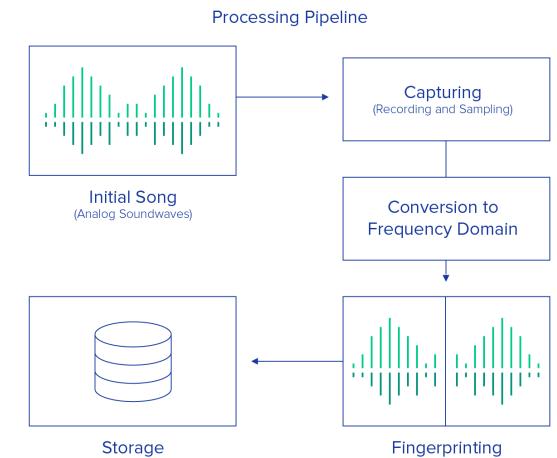
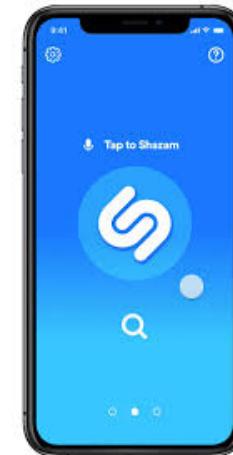
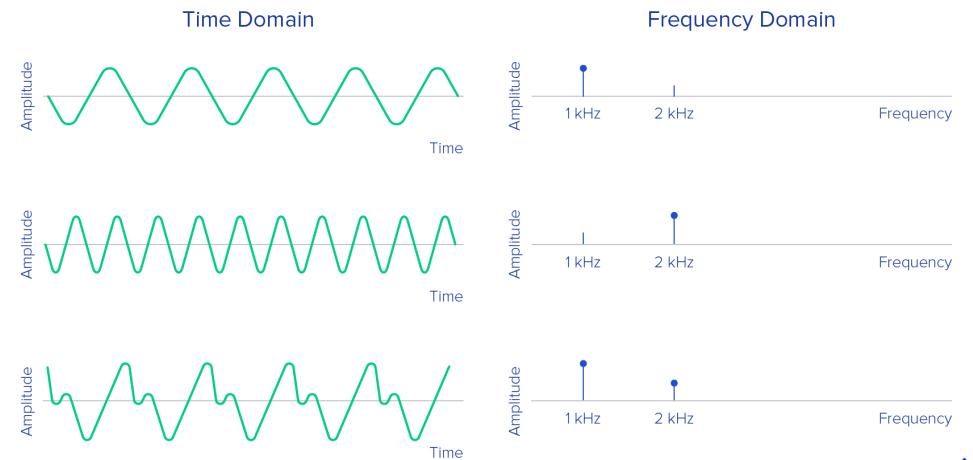
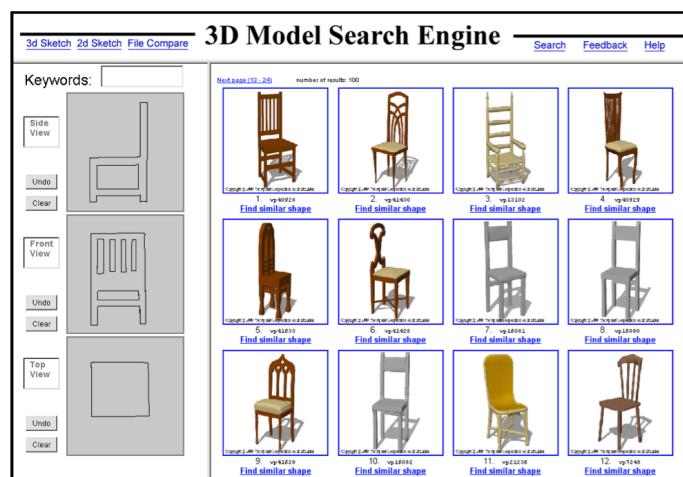
- Mais existence d'autres types de données
 - Non structurés ou semi-structurés
 - Numériques/Multimédia : Image, Son/Parole, Vidéo, 3D, ...
 - Avec des structures spécifiques : Molécule chimique, Séries temporelles, ...
- Les techniques précédentes basées sur SQL ou à base de mots clés ne sont pas adaptées
 - Absence de structure en colonnes, pas de mots-clés associés aux objets, incapacité à formuler des conditions de recherche sur les caractéristiques (ex: forme, couleur, ...) car trop complexe
- Solution : Recherche en donnant un ou plusieurs exemples de façon à obtenir des objets similaires



Recherche par similarité

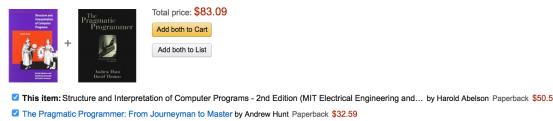


Applications: Recherche



Recommendations

Frequently Bought Together



Customers Who Bought This Item Also Bought

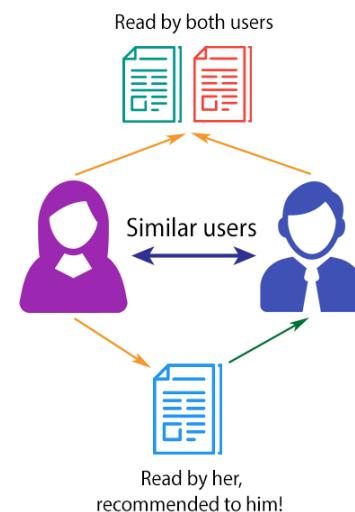


Radio

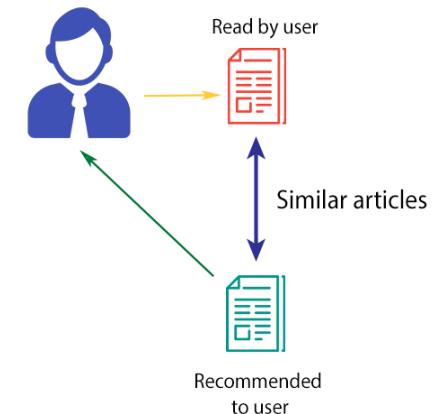
Recommended Stations



COLLABORATIVE FILTERING



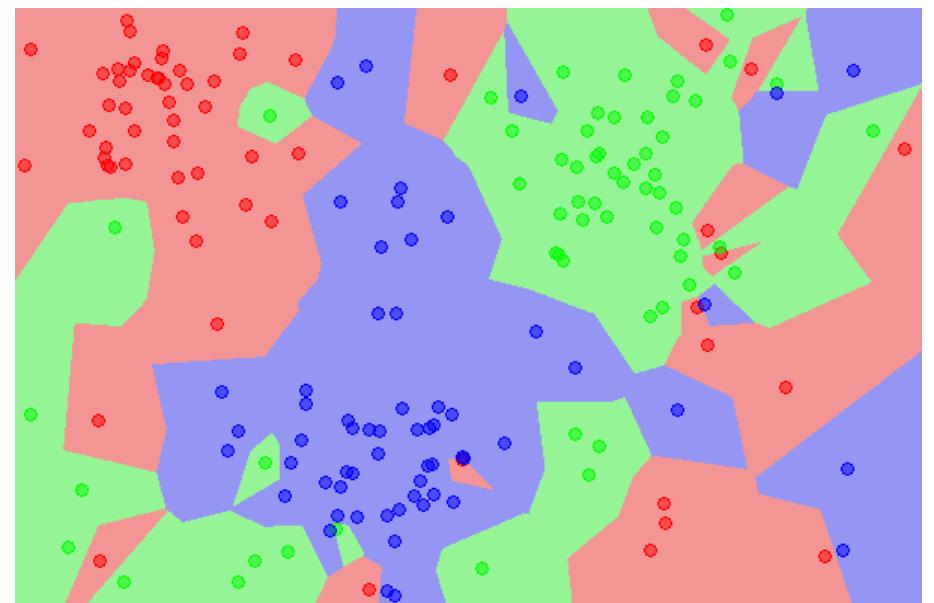
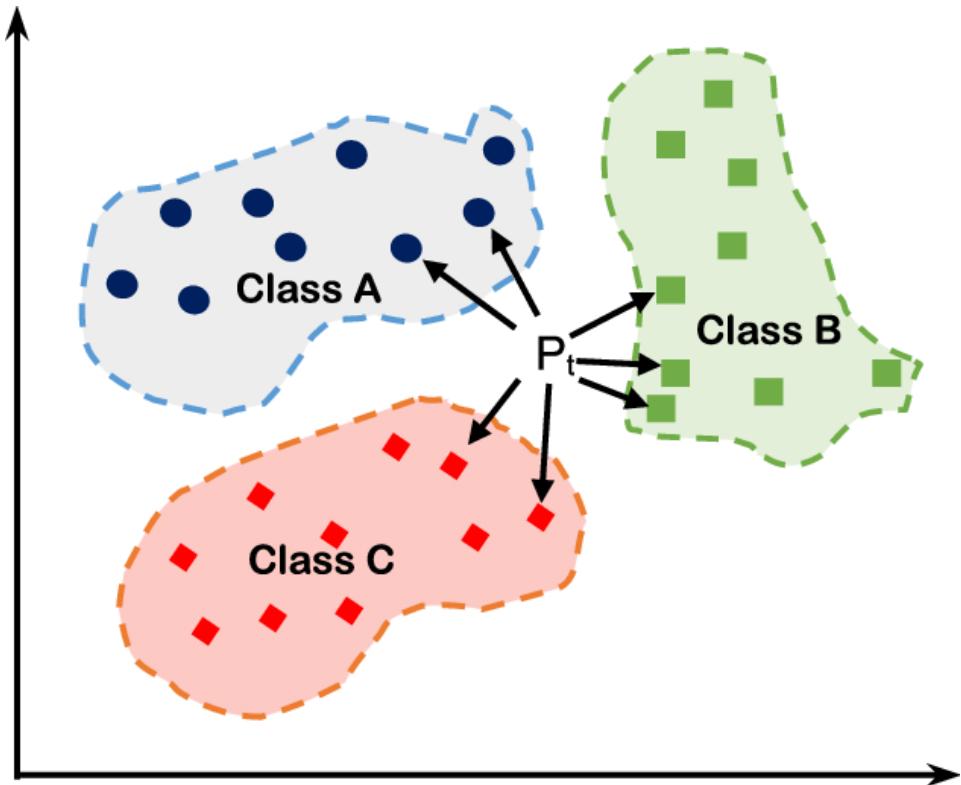
CONTENT-BASED FILTERING



	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

sim = 0,85
sim = 0,00
sim = 0,70
sim = -0,79

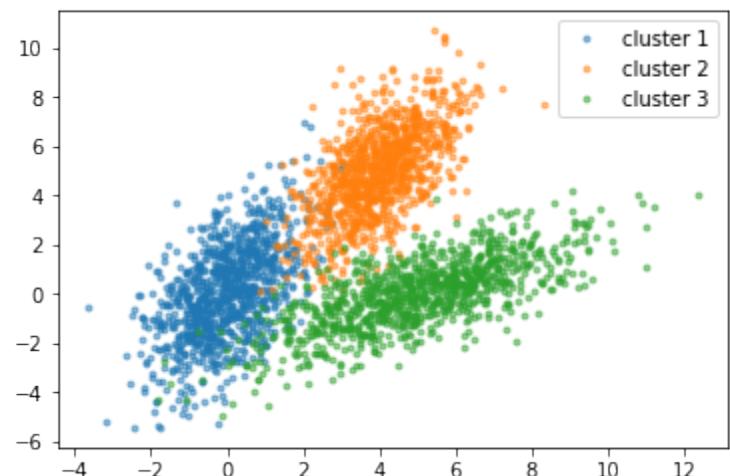
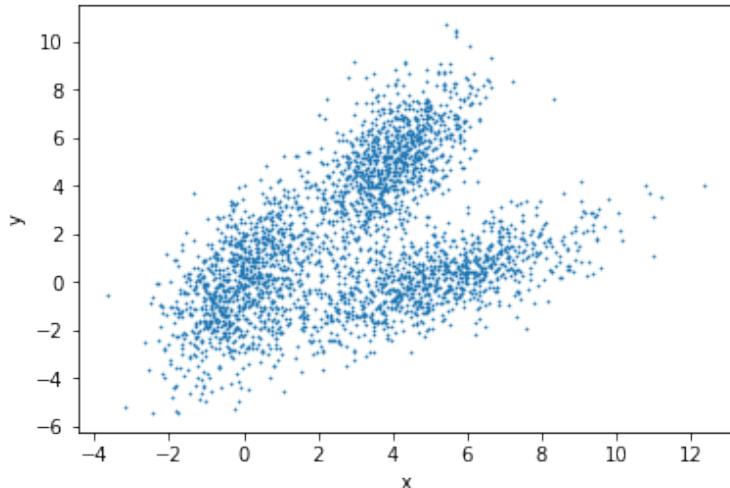
Classification / Regression



- Classification : statistiques des classes d'appartenance des + proches voisins (vote majoritaire, pondération en prenant en compte des distances)
- Regression: moyenne des + proches voisins

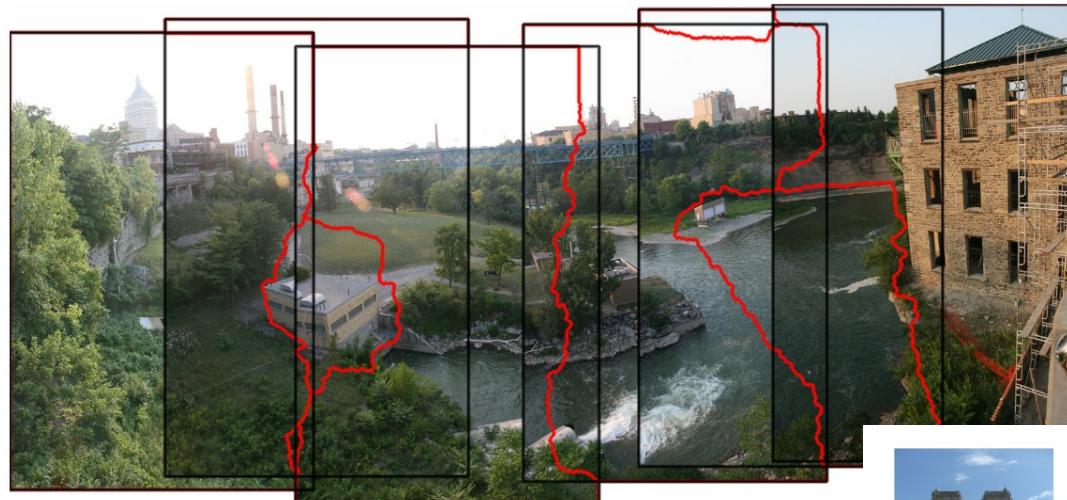
Partitionnement

Analyse exploratoire des données



Regroupement des données selon leur similitudes
(même gamme de couleurs, même forme, même objet détecté,)

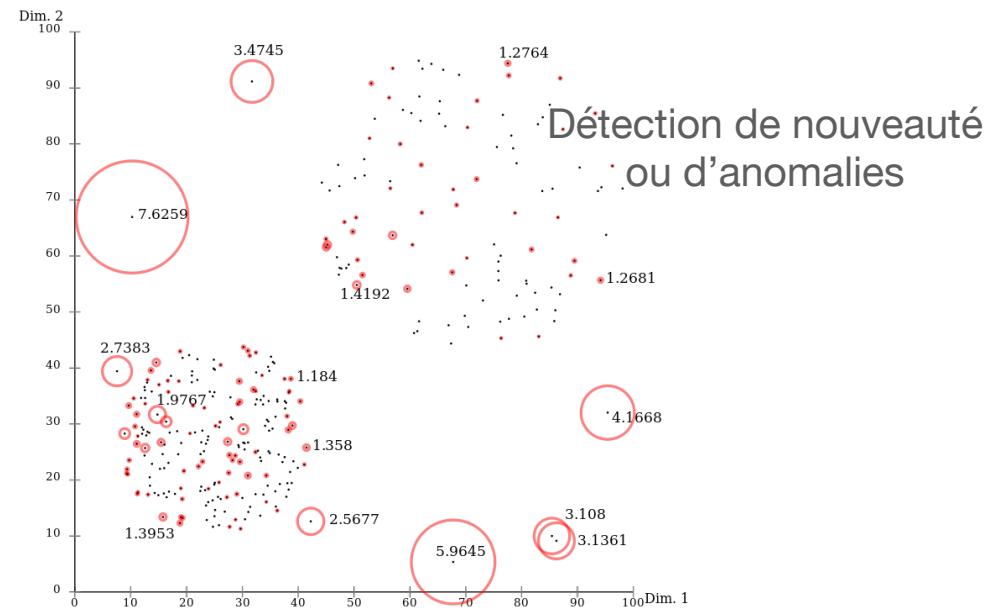
Autres applications



Reconstruction de l'image en identifiant les parties qui sont similaires

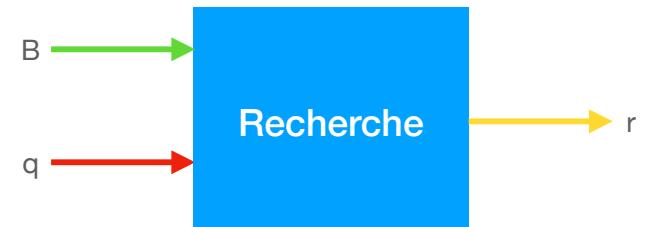


Géolocalisation à postériori d'images



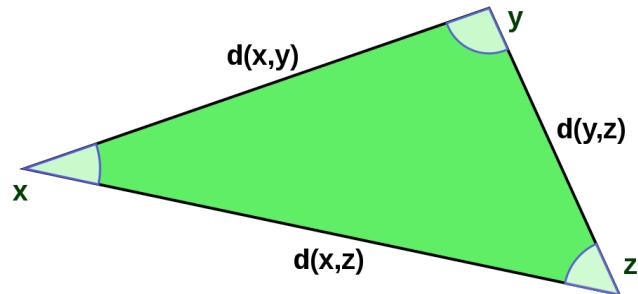
Formulation du problème

- Objectif: Trouver le ou les objets de la base qui sont les plus similaires à l'objet requête
- Entrée: $B = \{o_i\}, o_i \in \mathbb{U}$ $q \in \mathbb{U}$ Sortie: $r \in \mathbb{U}$ ou $\{r_i\}, r_i \in \mathbb{U}$
- Questions :
 - comment définir la similarité entre objets ?
 - fonction de similarité : $s : \mathbb{U} \times \mathbb{U} \longrightarrow \mathbb{R}$
 - fonction de dissimilarité (ou **distance**) : $\delta : \mathbb{U} \times \mathbb{U} \longrightarrow \mathbb{R}$
 - comment définir le plus similaire ?
 - $s(x, y) \geq s(x, z) \iff \delta(x, y) \leq \delta(x, z), \forall x, y, z \in \mathbb{U}$



Espaces de recherche

- La structure des objets de l'univers \mathbb{U} peut être un vecteur, une séquence, une chaîne de caractère, un ensemble (ordonné ou non), un arbre, ...
- Espace de similarité (resp de dissimilarité) : (\mathbb{U}, s) - (\mathbb{U}, δ)
- Espace métrique (distance avec des propriétés topologiques):
 - Réflexivité : $\delta(x, y) = 0 \iff x = y$
 - Positivité : $\delta(x, y) > 0 \iff x \neq y$
 - Symétrie : $\delta(x, y) = \delta(y, x)$
 - Inégalité triangulaire : $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$
- Ces propriétés permettent de concevoir des algorithmes de recherche plus efficaces



Fonctions de distance

- Discrete vs Continue
 - fonction qui retourne un petit ensemble de valeurs prédéfinies: {0, 1}, [0,5]
 - fonction dont la cardinalité de l'ensemble est large voire infinie: \mathbb{N} , \mathbb{R} , [0.0, 1.0]
- Pour différents types de données
 - Vecteurs, chaîne de caractères, ensemble d'objets, distribution, sequence,
- Respect des propriétés topologiques ou non
 - Non symétriques, non réflexives, non respect de l'inégalité triangulaire
- Propriétés pratiques
 - Coût calculatoire, dérivabilité, comportement en cas de dimensions élevées,
 - Robustesse % faibles changements, Localité (capacité à ignorer certains portions de l'objet)

Distances entre vecteurs

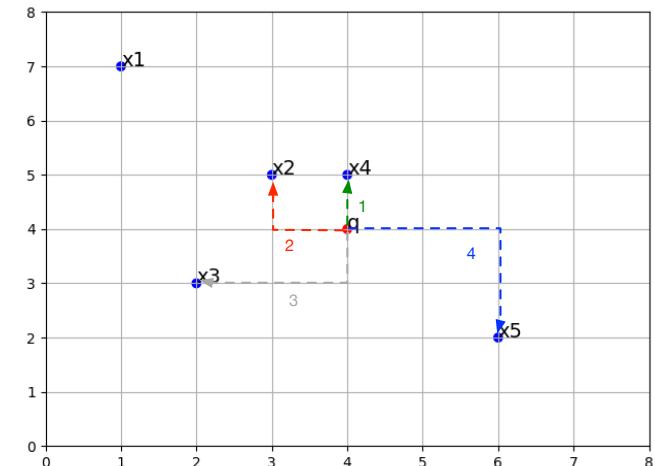
- Distance de Minkowski (également appelé L_p) définie sur des vecteurs de dimension n

$$\delta_{L_k}(x, q) = \sqrt[k]{\sum_{i=1}^n |x_i - q_i|^k}$$

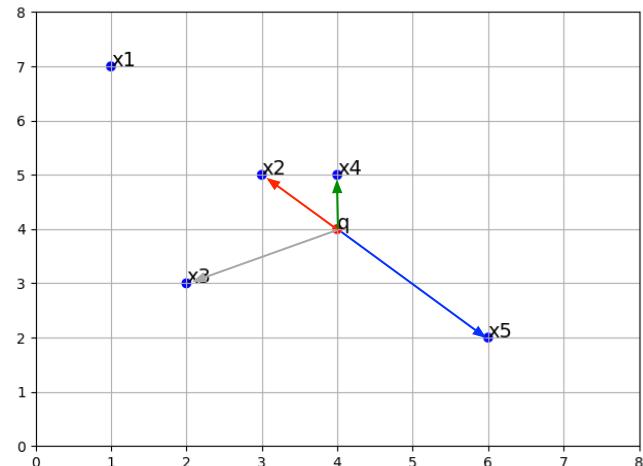
- Cas usuels :
 - Distance de Manhattan (L_1) : $\delta_{L_1}(x, q) = \sum_{i=1}^n |x_i - q_i|$
 - Distance Euclidienne (L_2) : $\delta_{L_2}(x, q) = \sqrt{\sum_{i=1}^n |x_i - q_i|^2}$
 - Distance Tchebychev (L_∞) : $\delta_{L_\infty}(x, q) = \max_{i=1}^n |x_i - q_i|$

Distances entre vecteurs

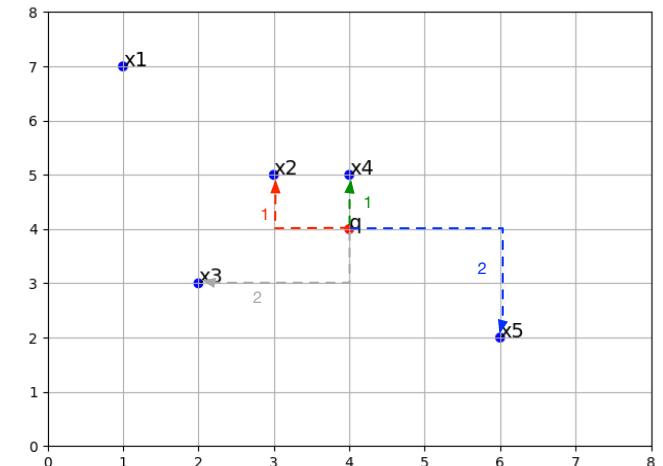
Distance Manhattan



Distance Euclidienne



Distance Tchebychev



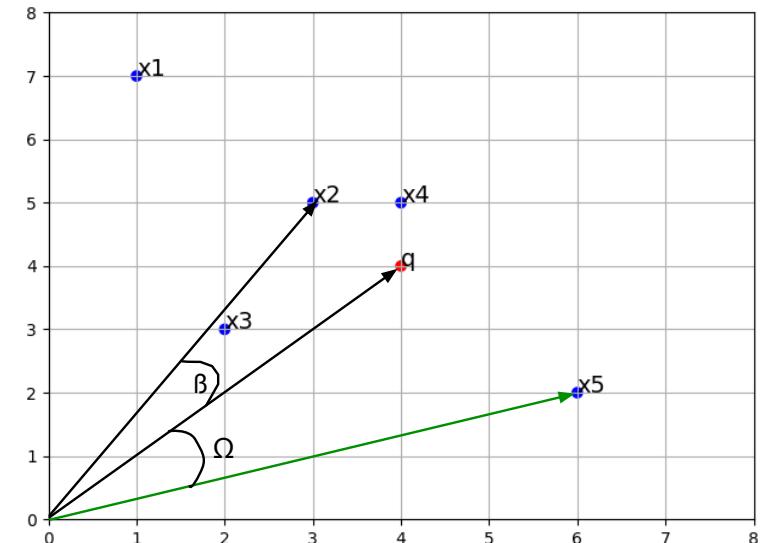
Distances entre vecteurs

- Distance cosinus (cosinus de l'angle de 2 vect.)

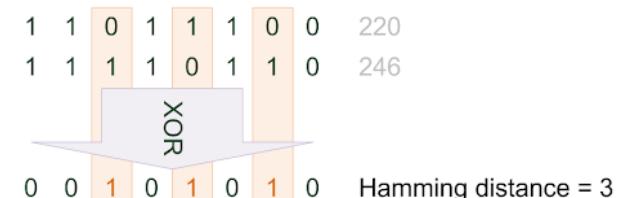
$$\delta_{cos}(x, q) = \frac{\langle x, q \rangle}{\|x\|_2 \cdot \|q\|_2} = \frac{\sum_{i=1}^d x_i \cdot q_i}{\sqrt{\sum_{i=1}^d x_i^2 \cdot \sum_{i=1}^d q_i^2}}$$

- Distance de Hamming : dans \mathbb{B}^d
 - $\delta_h(x, q) = \sum_{i=1}^n x_i \oplus q_i$ (\oplus = opérateur XOR)
- Tableau des distances sur les vecteurs

	Euclidean	Manhattan	Tchebychev	Cosine	Hamming
Space	\mathbb{R}^d	\mathbb{R}^d	\mathbb{R}^d	\mathbb{R}^d	\mathbb{B}^d
Derivability	Yes	No	No	Yes	No
Cost	$\Theta(n)$	* $\Theta(n)$	$\Theta(n)$	$\Theta(n)$ *	$\Omega(1)^*, O(n)$
Behavior in high dimensions	Favors large values	Equal contributions	Large values	Favors large values	Equal contributions



Utilisation pour comparer des textes



* pas de calcul de la racine ou des normes

1 instruction machine

Distance entre chaînes de caractères

- Distance de Levenshtein : nombre d'opérations pour transformer la chaîne x en chaîne y
 - **Insertion** du caractère c dans la chaîne x à la position i : $ins(x, i, c) = x_1, x_2, \dots, x_i, c, x_{i+1}, \dots, x_n$
 - **Suppression** du caractère à la position i : $del(x, i) = x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n$
 - **Remplacement** du caractère à la position i dans x par le nouveau caractère c :
 $replace(x, i, c) = x_1, x_2, \dots, x_{i-1}, c, x_{i+1}, \dots, x_n$
 - Exemple: $w_{insert} = 2, w_{delete} = 1, w_{replace} = 1$
 - $\delta_{edit}(combine, combination) = 9 - replacement[e \Rightarrow a], insertion[t, i, o, n]$
 - $\delta_{edit}(combination, combine) = 5 - replacement[a \Rightarrow e], del[t, i, o, n]$
- Si le poids d'une insertion et d'une suppression diffère, la distance n'est pas symétrique
- Complexité $\Theta(nm)$ pour des chaînes de longueurs n et m

Distances entre ensembles

- Distance de Jaccard
 - utilise le rapport entre le **cardinal** de l'**intersection** des **ensembles** et le cardinal de l'**union** des ensembles.

$$\delta_{jacc}(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

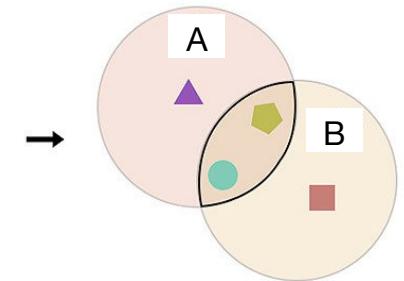
- Distance de Haussdorff

- mesure l'éloignement de deux **sous-ensembles** d'un **espace métrique** **sous-jacent**

$$\delta_{hauss}(A, B) = \max\{\delta_s(A, B), \delta_s(B, A)\}$$

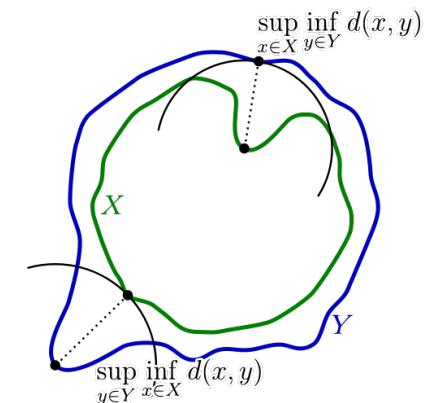
- Complexité $\Theta(nm)$ pour des ensembles de cardinalité n et m mais algorithme en $\Theta((n + m)\log(n + m))$

	●	■	▲	◆
A	●		▲	◆
B	●	■		◆
C	●		▲	



$$\frac{2}{4+1+1} = \frac{2}{6} = 0.$$

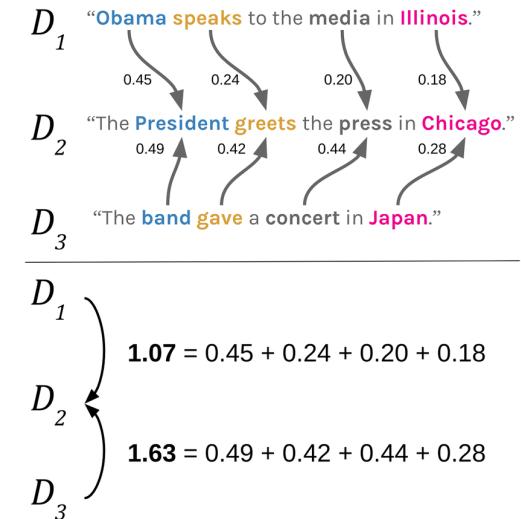
$$\begin{aligned}\delta_p(x, B) &= \inf_{y \in B} \delta_e(x, y) \\ \delta_p(A, y) &= \inf_{x \in A} \delta_e(x, y) \\ \delta_s(A, B) &= \sup_{x \in A} \delta_p(x, B) \\ \delta_s(B, A) &= \sup_{y \in B} \delta_p(A, y)\end{aligned}$$



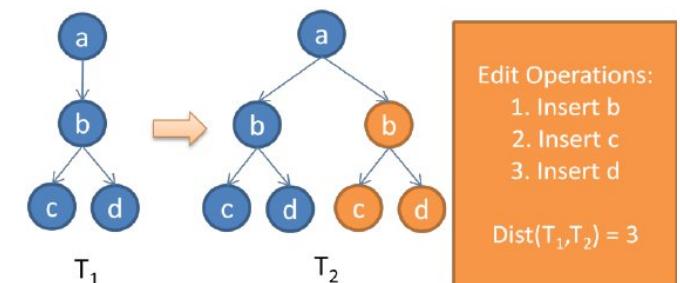
plus grande de toutes les distances d'un point d'un ensemble au point le plus proche dans l'autre ensemble

Complexité des distances

- Certaines distances peuvent avoir une complexité élevée
 - Distance de Levenshtein
 - Distance de Jaccard et de Haussdorf
 - Distance entre séquences de mots (Word Mover Distance): $\Theta(n^3 \log n)$
 - Distance entre arbres, entre graphes (tree and graph edit distances),
- Impact significatif sur la recherche par similarité
- Un objectif important est de minimiser le calcul des distances
 - Partitionnement
 - Utilisation des propriétés de l'espace des données



$$\begin{aligned}D_1 &\leftarrow 1.07 = 0.45 + 0.24 + 0.20 + 0.18 \\D_2 &\leftarrow 1.63 = 0.49 + 0.42 + 0.44 + 0.28 \\D_3 &\end{aligned}$$



Types de recherche

- Voisins dans région

- $R(q, r) = \{x \in \mathbb{U} \mid d(q, x) \leq r\}$

“Tous les musées situés à deux kms de mon hotel”

- K plus proches voisins

- $kNN(q) = A$

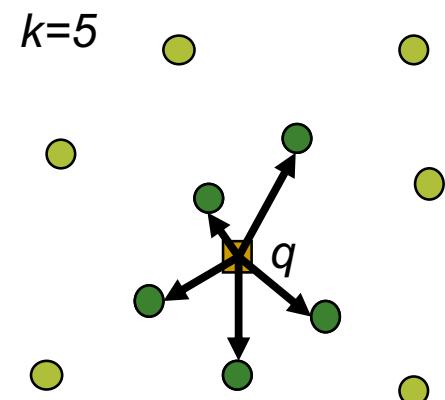
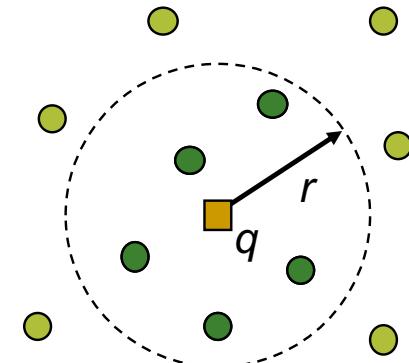
- $A \subseteq \mathbb{U}, |A| = k$

- $\forall x \in A, y \in \mathbb{U} - A : d(q, x) \leq d(q, y)\}$

“Les K musées les plus proches de mon hotel”

- Combinaison des deux

Le nombre de résultats est variable
(dépend de la concentration
des éléments autour de q)



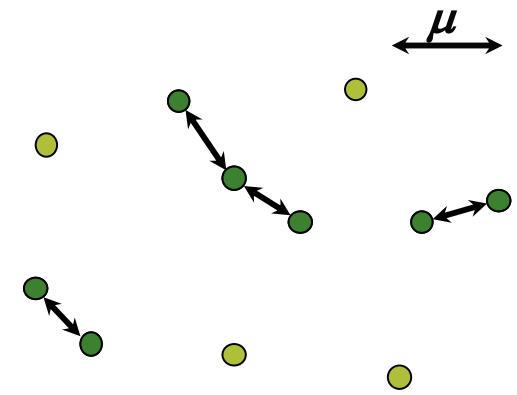
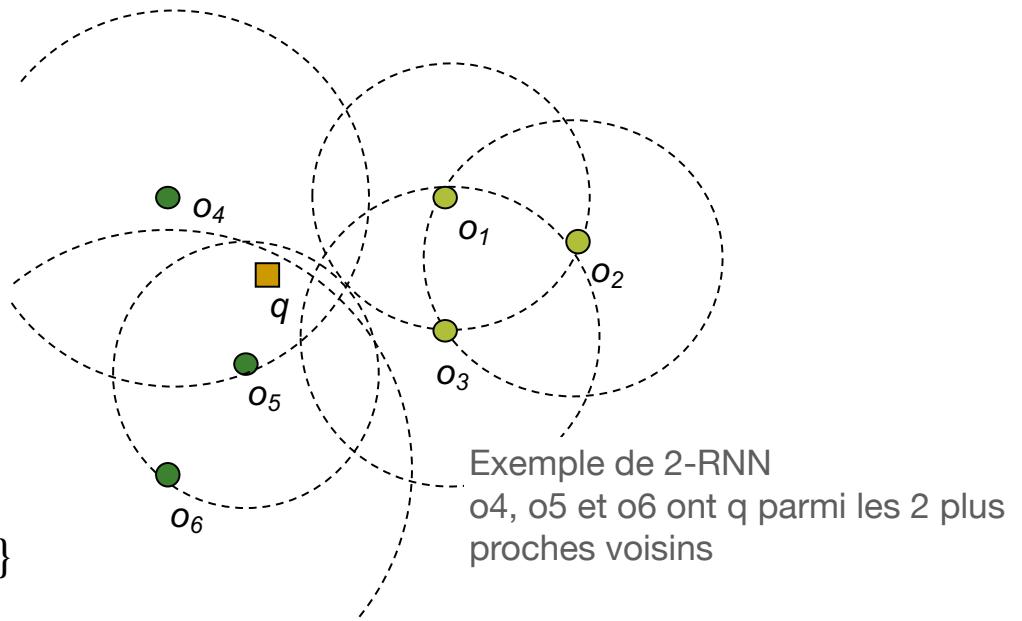
Types de recherche

- K plus proches voisins inversées
 - $kRNN(q) = A$
 - $A \subseteq U$
 - $\forall x \in A : q \in kNN(x) \wedge \forall x \in U - A : q \notin kNN(x)$

“Tous les hotels avec un musée spécifique comme site culturel”

- Paires similaires
 - $X \subseteq U, Y \subseteq U$
 - $J(X, Y, \mu) = \{(x, y) \in X \times Y : d(x, y) \leq \mu\}$

“paires d’hotels et musées qui sont à 5mn l’un de l’autre”

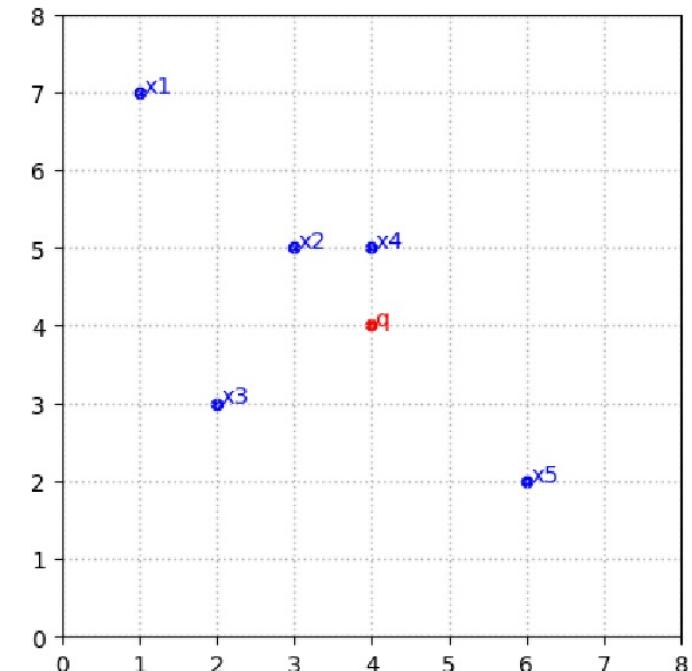


Algorithme Force Brute : Voisins dans Region

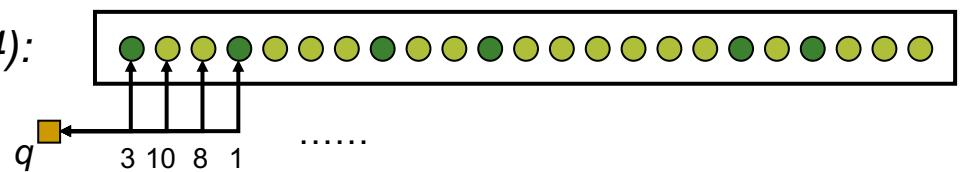
Radius search $\{x_1^*, x_2^*, \dots, x_m^*\} = \{x \in \mathcal{D} | \delta(x, q) \leq r\}$

Algorithm 5: Brute-force radius search.

```
1 function radius_search( $\mathcal{D}$ : Dataset,  $q$ : Query,  $r$ :  
   Float): List  
   input :  $\mathcal{D}$ : Dataset of  $n$  vectors in  $\mathbb{R}^d$   
            $q$ : Vector in  $\mathbb{R}^d$   
            $r$ : search radius  
   output: nearests :  $k$  nearest neighbors of  $q$  in  $\mathcal{D}$   
2   nearests  $\leftarrow \emptyset$   
3   for  $x \in \mathcal{D}$  do  
4     if  $\delta(x, q) \leq r$  then  
5       nearests.add( $x$ )  
6     end  
7   end  
8   return nearests  
9 end
```



$$R(q, 4):$$



Algorithme Force Brute : Voisins dans Region

Radius search $\{x_1^*, x_2^*, \dots, x_m^*\} = \{x \in \mathcal{D} | \delta(x, q) \leq r\}$

Algorithm 6: Brute-force radius search.

```
1 function radius_search( $\mathcal{D}$ : Dataset,  $q$ : Query,  $r$ :  
    Float): List  
    input :  $\mathcal{D}$ : Dataset of  $n$  vectors in  $\mathbb{R}^d$   
             $q$ : Vector in  $\mathbb{R}^d$   
             $r$ : search radius  
    output: nearests :  $k$  nearest neighbors of  $q$  in  $\mathcal{D}$   
2     nearests  $\leftarrow \emptyset$   
3     for  $x \in \mathcal{D}$  do  
4         if  $\delta(x, q) \leq r$  then  
5             | nearests.add( $x$ )  
6         end  
7     end  
8     return nearests  
9 end
```

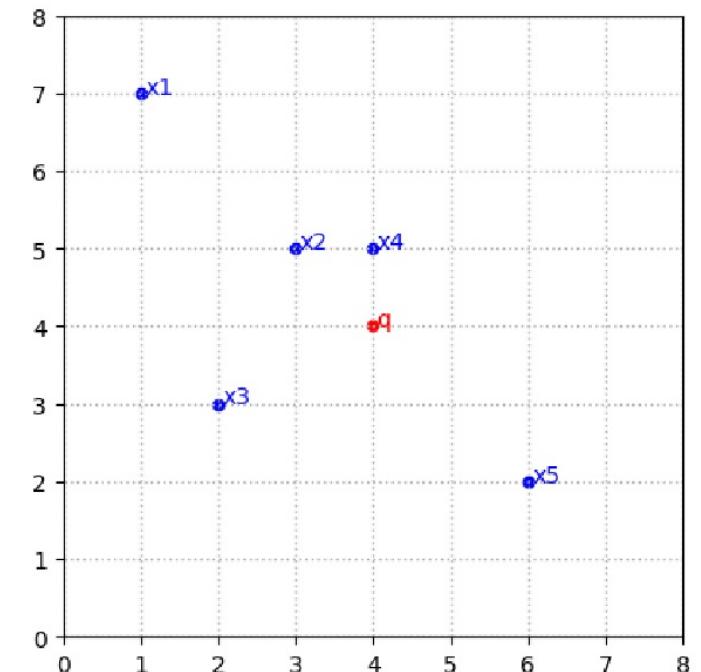
- Analyse de complexité
 - calcul de $\delta = d$ itérations
 - boucle principale = n iterations pour les n éléments
 - Coût total = $\Theta(nd)$

Algorithme Force Brute : Plus proche voisin

NN search: $x^* = \arg \min_{x \in \mathcal{D}} \delta(x, q)$

Algorithm 1: Brute-force NN search.

```
1 function nn_search( $\mathcal{D}$ : Dataset,  $q$ : Query) : Vector
    input :  $\mathcal{D}$ : Dataset of  $n$  vectors in  $\mathbb{R}^d$ 
             $q$ : Vector in  $\mathbb{R}^d$ 
    output: nearest : Nearest neighbor of  $q$  in  $\mathcal{D}$ 
2     min_distance  $\leftarrow +\infty$ 
3     for  $x \in \mathcal{D}$  do
4         if  $\delta(x, q) < min\_distance$  then
5              $min\_distance \leftarrow \delta(x, q)$ 
6             nearest  $\leftarrow x$ 
7         end
8     end
9     return nearest
10 end
```



Algorithme Force Brute : Plus proche voisin

NN search: $x^* = \arg \min_{x \in \mathcal{D}} \delta(x, q)$

Algorithm 2: Brute-force NN search.

```
1 function nn_search( $\mathcal{D}$ : Dataset,  $q$ : Query): Vector
  input :  $\mathcal{D}$ : Dataset of  $n$  vectors in  $\mathbb{R}^d$ 
           $q$ : Vector in  $\mathbb{R}^d$ 
  output: nearest : Nearest neighbor of  $q$  in  $\mathcal{D}$ 
2   min_distance  $\leftarrow +\infty$ 
3   for  $x \in \mathcal{D}$  do
4     if  $\delta(x, q) < min\_distance$  then
5        $min\_distance \leftarrow \delta(x, q)$ 
6       nearest  $\leftarrow x$ 
7     end
8   end
9   return nearest
10 end
```

- Analyse de complexité
 - calcul de δ = d itérations
 - boucle principale = n iterations pour les n éléments
 - Coût total = $\Theta(nd)$

Algorithme Force Brute : K + proches voisins

k-NN search $x_1^*, x_2^* \dots x_k^* = \underset{x \in \mathcal{D}}{\text{k-arg min}} \delta(x, q)$

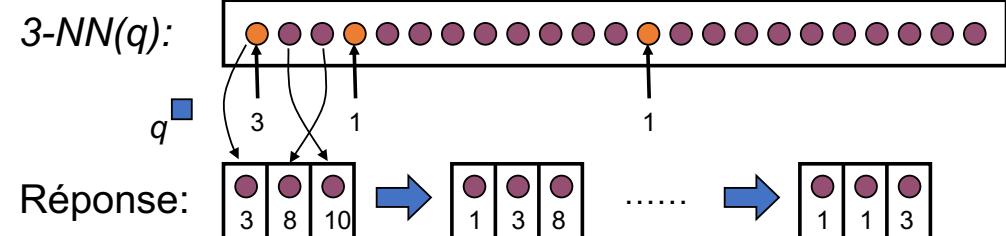
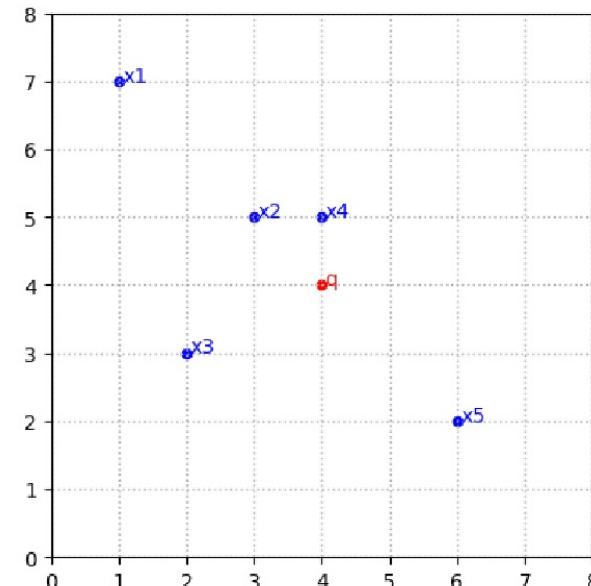
Algorithm 3: Brute-force k-NN search.

```

1 function knn_search( $\mathcal{D}$ : Dataset,  $q$ : Query,  $k$ : Integer):
    Priority Queue
        input :  $\mathcal{D}$ : Dataset of  $n$  vectors in  $\mathbb{R}^d$ 
                 $q$ : Vector in  $\mathbb{R}^d$ 
                 $k$ : Number of neighbors to retrieve
        output: nearests :  $k$  nearest neighbors of  $q$  in  $\mathcal{D}$ 
2     nearests  $\leftarrow \emptyset$ 
3     for  $x \in \mathcal{D}$  do
4         if  $\text{nearests.size()} < k$  then
5             nearests.add( $(x, \delta(x, q))$ )
6         else if  $\delta(x, q) < \text{nearests.get\_max\_distance}()$  then
7             nearests.remove_max_element()
8             nearests.add( $(x, \delta(x, q))$ )
9         end
10    end
11    return nearest
12 end

```

Mémorisation des distances avec les points



Algorithme Force Brute : K + proches voisins

k-NN search $x_1^*, x_2^* \dots x_k^* = \underset{x \in \mathcal{D}}{\text{k-arg min}} \delta(x, q)$

Algorithm 4: Brute-force k-NN search.

```
1 function knn_search( $\mathcal{D}$ : Dataset,  $q$ : Query,  $k$ : Integer):  
Priority Queue  
    input :  $\mathcal{D}$ : Dataset of  $n$  vectors in  $\mathbb{R}^d$   
             $q$ : Vector in  $\mathbb{R}^d$   
             $k$ : Number of neighbors to retrieve  
    output: nearests :  $k$  nearest neighbors of  $q$  in  $\mathcal{D}$   
2     nearests  $\leftarrow \emptyset$   
3     for  $x \in \mathcal{D}$  do  
4         if  $\text{nearests.size()} < k$  then  
5              $\text{nearests.add}((x, \delta(x, q)))$   
6         else if  $\delta(x, q) < \text{nearests.get\_max\_distance}()$  then  
7              $\text{nearests.remove\_max\_element}()$   
8              $\text{nearests.add}((x, \delta(x, q)))$   
9         end  
10    end  
11    return nearests  
12 end
```

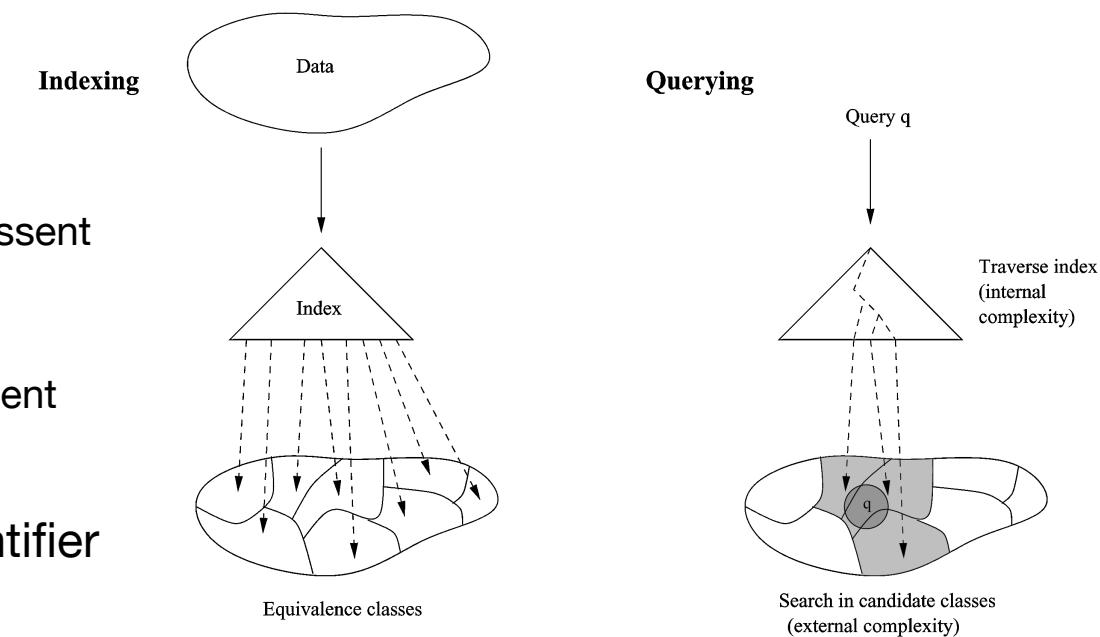
- Analyse de complexité
 - calcul de $\delta = d$ itérations
 - boucle principale = n iterations pour les n éléments
 - Accès au max:
 - Liste non triée:
 - Insertion: $\Theta(1)$, max: $\Theta(k)$
 - Liste triée:
 - Insertion: $\Theta(\log k)$, max: $\Theta(1)$
 - Coût total = $\Theta(nd \log k)$

Bilan sur les approches force-brute

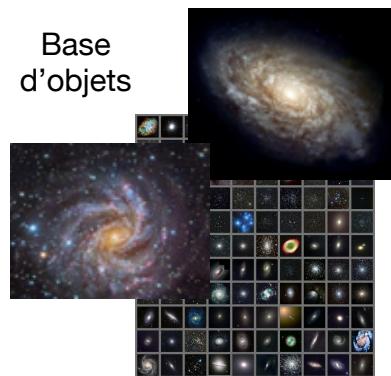
- La complexité de la recherche basique est linéaire en taille du jeu de données et linéaire en dimension
- Recherche de paires similaires: $\Theta(n^2d)$
- En pratique:
 - Trop coûteux pour une utilisation à large échelle : $n \approx 1000000$ et $d \approx 200$
 - Les applications requièrent généralement des réponses rapides
- Solutions:
 - Réduire le temps de recherche en partitionnant l'espace et en explorant que les parties pertinentes
 - Recherche approchée : Les “presque” plus proches voisins

Indexation et Recherche

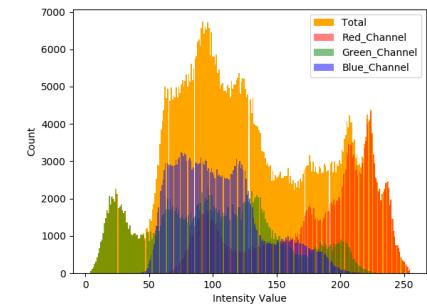
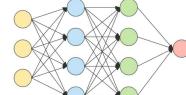
- **Indexation** : Organisation des données pour accélérer la recherche
 - Création d'une structure de donnée = Index
 - L'index est conçu pour que les objets similaires puissent être facilement retrouvés
 - Traitement effectué hors-ligne et une seule fois -> Complexité de construction peut être raisonnablement élevée : $\Theta(n^2)$
- **Recherche** : Exploitation de l'index pour identifier les candidats de la recherche
 - Inspecter seulement un sous-ensemble des objets
 - Traitement effectué en ligne et pour chaque requête -> Complexité doit être faible (sous-linéaire) : $< \Theta(n)$



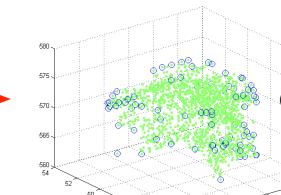
Indexation et recherche



Extraction des caractéristiques et transformation

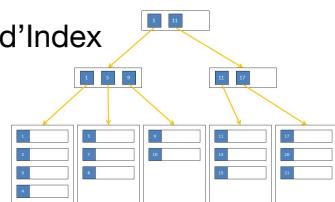


Vecteurs de caractéristiques



ID: 1	ID: 2	...	ID: 256
[0.13]	[0.32]	...	[1.03]
[0.98]	[0.27]	...	[0.08]
ID: 1	ID: 2	...	ID: 256
[0.3]	[0.35]	...	[1.28]
[1.28]	[0.12]	...	[1.13]
ID: 1	ID: 2	...	ID: 256
[0.13]	[0.72]	...	[1.03]
[0.98]	[1.34]	...	[0.08]

Structure d'Index



Indexation



Recherche

0.34
0.22
0.68
1.02
0.03
0.71

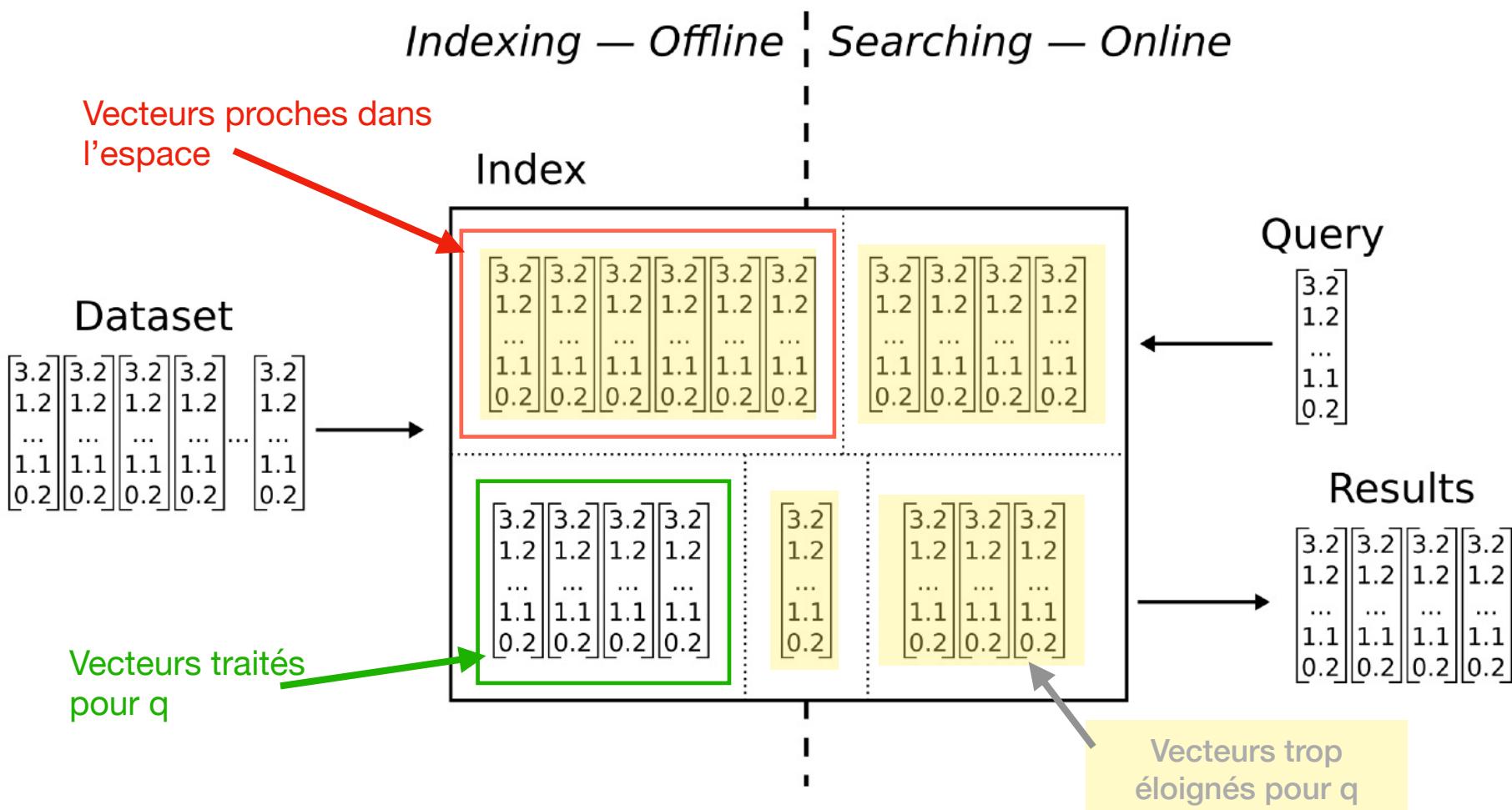
Requête

Résultat



Objets similaires

Indexation et Recherche



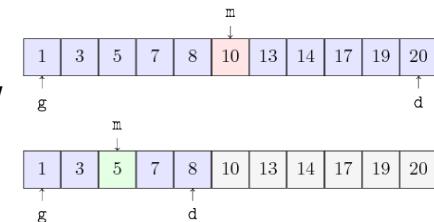
Recherche exacte à base d'arbres

Partitionnement selon les dimensions

- La technique de partitionnement consiste à diviser l'espace de recherche en sous-groupes de façon à ce que seulement certains groupes soient explorés lors de la recherche
- Espace de données multi-dimensionnels
 - Vecteurs d'entiers \mathbb{N}^d , vecteurs de réels \mathbb{R}^d
- Division récursive de l'espace selon les **dimensions**
 - Utilisation d'hyperplans parallèles aux axes (base de représentation initiale des vecteurs)
 - Nombreuses possibilités pour la sélection des axes
- Structure de donnée hiérarchique de type Arbre

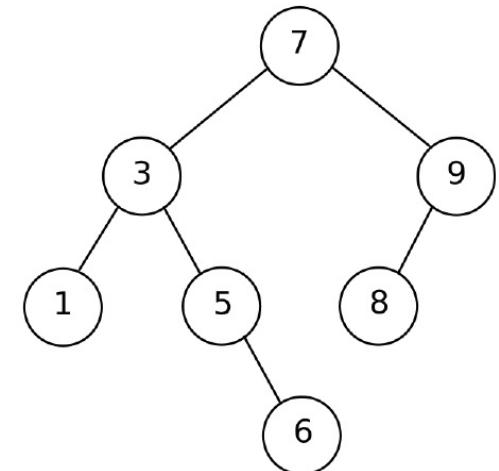
Rappel : Arbre binaire de recherche

- Comment résoudre efficacement le problème suivant ? (pensez à bcp d'entiers dans l'ensemble et bcp de recherche d'entiers q)
- soit $X = \{5,9,1,14,3,21,7,2,17,26\}$
- soit un entier quelconque (ex: $q = 6$), quel est l'entier le plus proche dans l'ensemble X ?
- Solution basique: $\Theta(n)$
 - Trier X puis parcourir séquentiellement le résultat du tri jusqu'à trouver les 2 entiers x et y qui encadrent q puis calculer la différence
- Solution plus performante: $\Theta(\log n)$
 - Trier X puis faire une recherche dichotomique pour trouver les 2 entiers x et y qui encadrent q puis calculer la différence



Rappel : Arbre binaire de recherche

- Structure de donnée de type arbre
 - représentant un **ensemble** dont les éléments sont ordonnables (relation d'ordre)
 - adaptée à la recherche
- Caractéristiques
 - 1 seule racine, chaque noeud correspond à un élément, chaque noeud possède 0, 1 ou 2 noeuds fils (sous-arbres)
 - Noeuds fils du sous-arbre gauche : éléments inférieurs au noeud courant
 - Noeuds fils du sous-arbre : éléments supérieurs au noeud courant



Algorithm 1: Structures for a binary search tree.

```
1 structure Node
2   |   value: Value
3   |   left_child: Node
4   |   right_child: Node
5 end
6 structure Tree
7   |   root: Node
8 end
```

Rappel : Arbre binaire de recherche

Algorithm 2: Binary tree search (recursive).

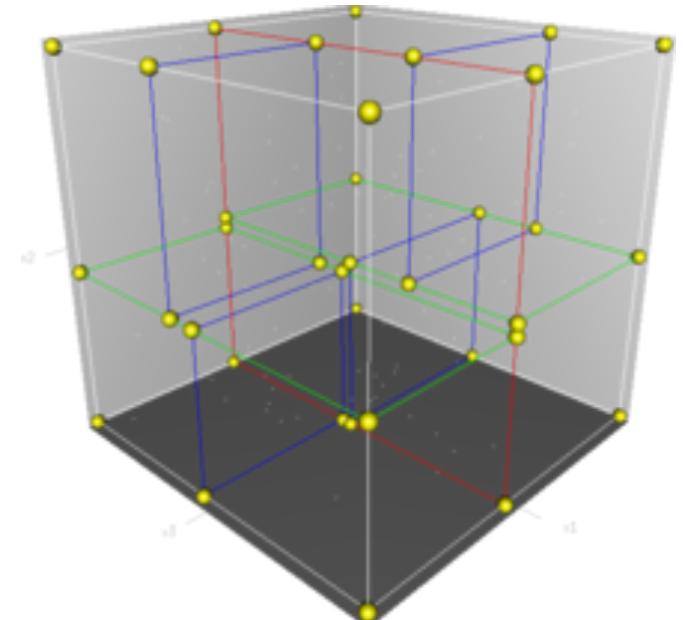
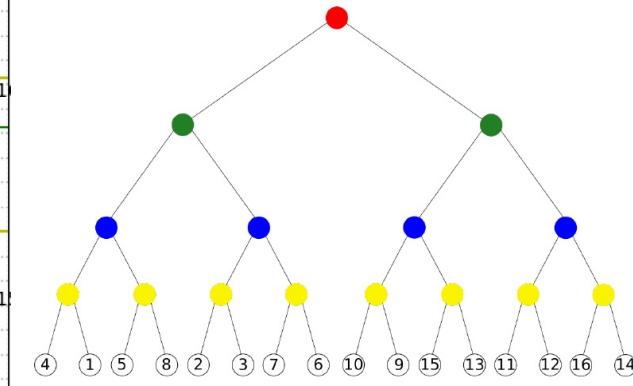
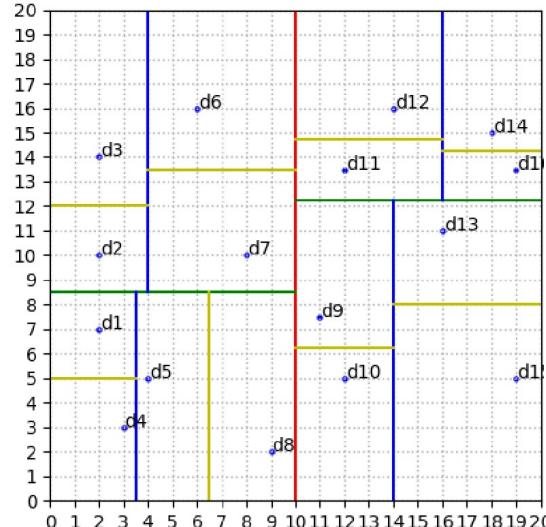
```
1 function tree_search(node: Node, v: Value): Value
2     res: Value
3     if is_null(node) then
4         | res ← NOT_FOUND
5     else if v = node.value then
6         | res ← node.value
7     else if v < node.value then
8         | res ← tree_search(node.left_child, v)
9     else
10        | res ← tree_search(node.right_child, v)
11    end
12    return res
13 end
```

Algorithm 3: Binary tree search (iterative).

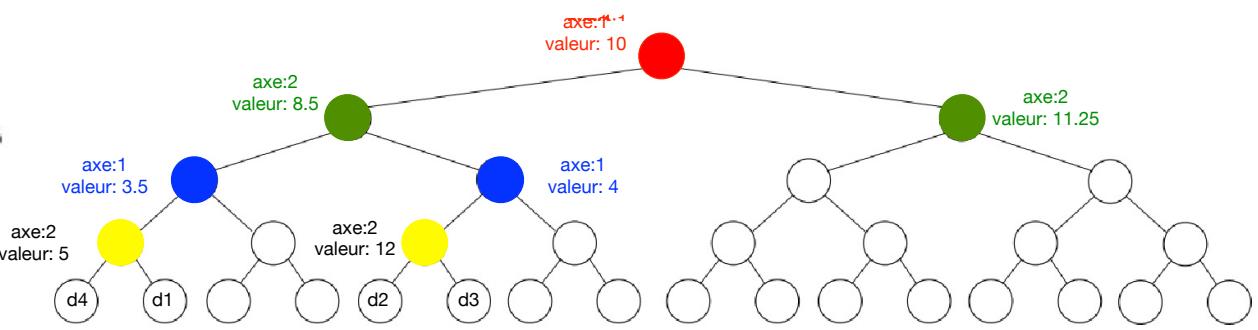
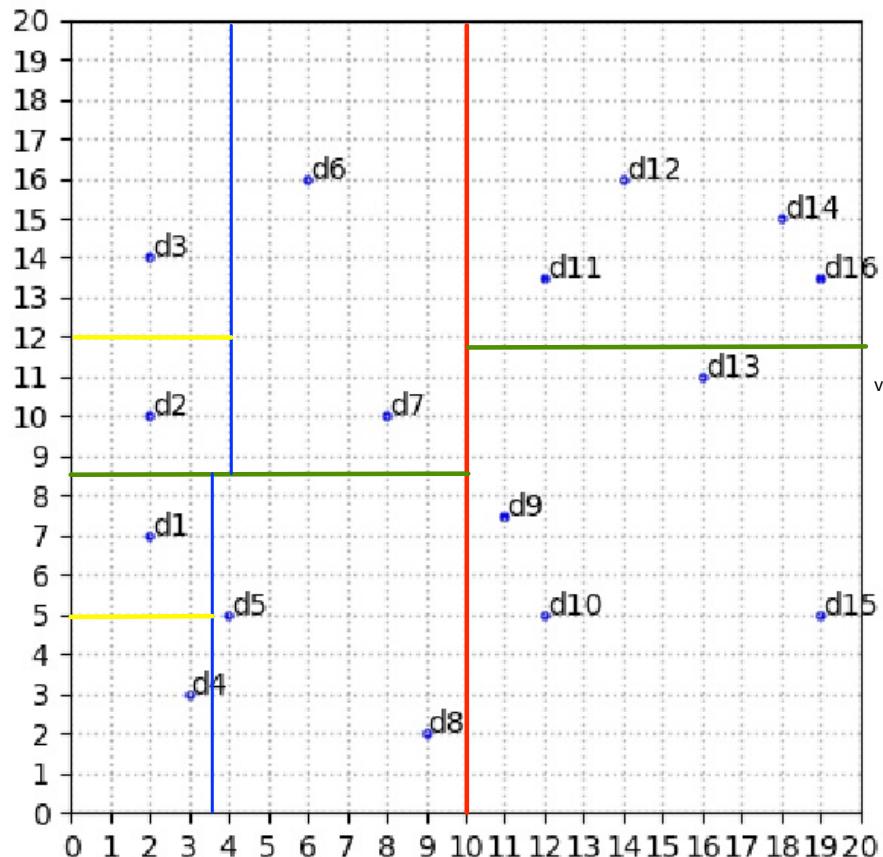
```
1 function tree_search(t: Tree, v: Value): Value
2     res: Value
3     node: Node
4     res ← NOT_FOUND
5     node ← t.root
6     while not(is_null(node)) and node.value ≠ v do
7         if node.value < v then
8             | node ← node.left_child
9         else
10            | node ← node.right_child
11        end
12    end
13    if not(is_null(node)) then
14        | res ← node.value
15    end
16    return res
17 end
```

Arbre k-d

- Extension des arbres binaires de recherche à des vecteurs de k-dimensions
- Espace découpé récursivement en utilisant des hyperplans parallèles aux axes (base de représentation initiale du vecteur)
 - point de découpage = valeur médiane sur l'axe considéré
 - choix du prochain axe
 - plus grande variance résiduelle
 - équilibrage de l'arbre, vecteurs proches situés du même côté dans les sous-arbres



Construction Arbre kd



- Cellules disjointes
- Cellules de tailles équilibrés par construction
- 1 objet par cellule ici mais possibilité d'avoir plusieurs objets dans les feuilles

Construction Arbre kd

Algorithm 4: Structures for a kd-tree.

```
1 structure Node
2   split_value: Float
3   split_axis: Index
4   x: Vector in  $\mathbb{R}^d$ 
5   left_child: Node
6   right_child: Node
7 end
8 structure Tree
9   root: Node
10 end
```

Algorithm 5: kd-tree construction.

```
1 function build_tree( $\mathcal{D}: \mathbb{R}^{n \times d}$ ): Node
2   result: Node
3   if  $n = 1$  then
4     result.x  $\leftarrow \mathcal{D}[0]$ 
5   else
6     result.split_axis  $\leftarrow \arg \max_{1 \leq i \leq d} (\text{variance}(\mathcal{D}[:, i]))$ 
7     result.split_value  $\leftarrow \text{median}(\mathcal{D}[:, i])$ 
8      $\mathcal{D}_{\text{left}} = \{x \in \mathcal{D} \mid x[\text{split\_axis}] < \text{split\_value}\}$ 
9      $\mathcal{D}_{\text{right}} = \{x \in \mathcal{D} \mid x[\text{split\_axis}] \geq \text{split\_value}\}$ 
10    result.left_child  $\leftarrow \text{build\_tree}(\mathcal{D}_{\text{left}})$ 
11    result.right_child  $\leftarrow \text{build\_tree}(\mathcal{D}_{\text{right}})$ 
12  end
13  return result
14 end
```

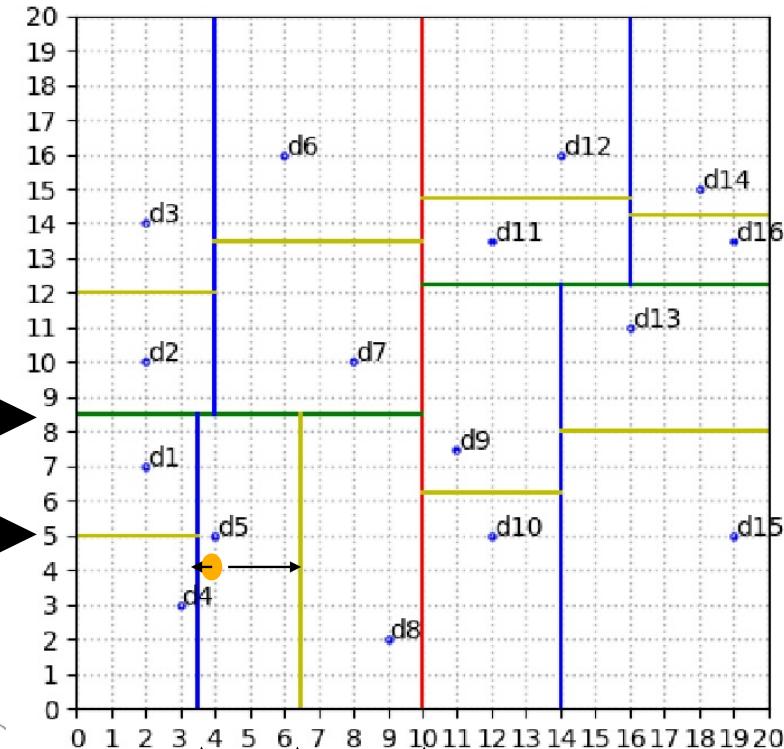
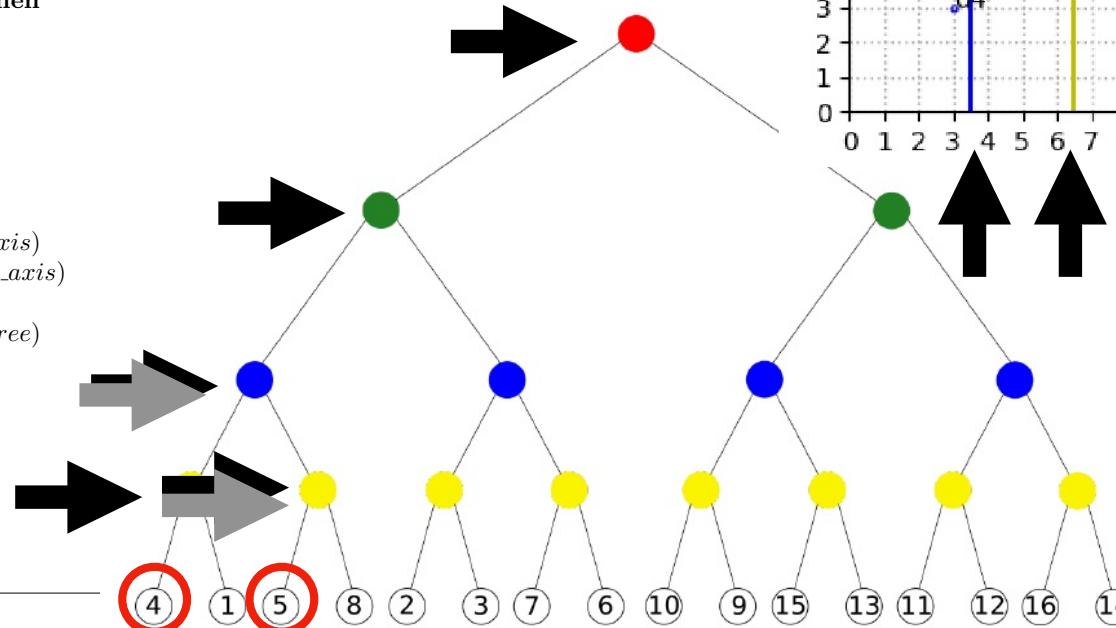
Recherche Arbre kd

Algorithm 1 Search in a kd tree

```

1: function TREE_SEARCH(node : Node, q :  $R^d$ ) $R^d$ 
2:   res :  $R^d$ 
3:   res.dist : R
4:   other_dist : R
5:   near_tree : Node
6:   far_tree : Node
7:   if is_leaf(node) then
8:     res  $\leftarrow$  node.x
9:   else
10:    if q[node.split_axis] < node.split_value then
11:      near_tree  $\leftarrow$  node.left_child
12:      far_tree  $\leftarrow$  node.right_child
13:    else
14:      near_tree  $\leftarrow$  node.right_child
15:      far_tree  $\leftarrow$  node.left_child
16:    end if
17:    res  $\leftarrow$  TREE_SEARCH(near_tree)
18:    res.dist  $\leftarrow$  distance.k(q, res, node.split_axis)
19:    axis_dist  $\leftarrow$  distance.k(q, axis, node.split_axis)
20:    if axis_dist < res.dist then
21:      other_res  $\leftarrow$  TREE_SEARCH(far_tree)
22:      other_dist  $\leftarrow$  distance(q, other_res)
23:      if other_dist < res.dist then
24:        res  $\leftarrow$  other_res
25:      end if
26:    end if
27:  end if
28:  Return res
29:
```

$q = (3, 6, 4)$



- Visite d'un nombre limité de feuilles
- Elagage des sous-arbres en évaluant la distance au meilleur résultat trouvé

Variantes Arbres kd

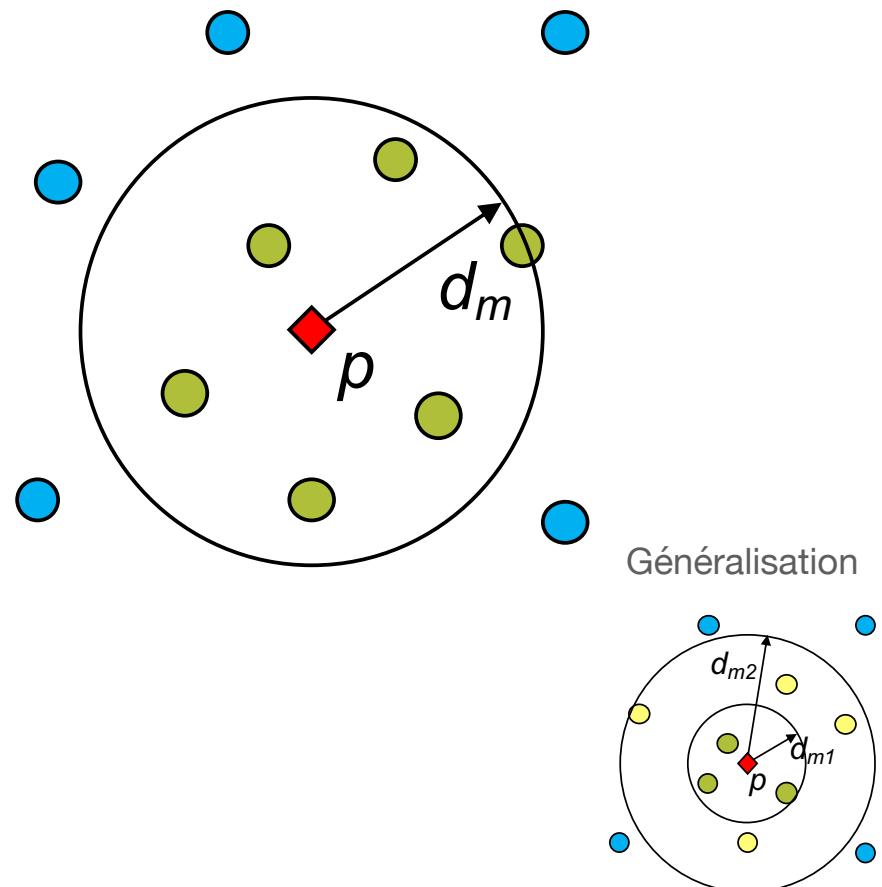
- Illustré sur une recherche du plus proche voisin
- Applicable à la recherche des plus proches voisins et des voisins dans region
 - Utilisation d'une file avec priorité pour les plus proches voisins
- Construction de l'arbre
 - Choix des axes (PCA), Point de découpage (aléatoire, moyenne, k-means, ...)
- Stratégies de recherche: utilisation de plusieurs arbres
- Complexité :
 - construction $\Theta(n \log n)$,
 - recherche $\Theta(n^{1-1/k} + m)$, $k = \text{dim}$, $m = \text{nb résultats}$
- Problèmes : ajout de nouveaux objets, jeux de données partiellement en mémoire

Partitionnement selon les distances

- Utilisation des **distances** pour diviser l'espace de recherche en sous-groupes de façon à ce que seulement certains groupes soient explorés lors de la recherche
- Le recours aux distances permet une application plus large
 - Pas uniquement des vecteurs mais tout espace métrique
 - Exploitation des propriétés de la distance pour ignorer des sous-groupes
- Plusieurs techniques de partitionnement
 - Balle
 - Hyperplan Généralisé
 - ...
- Arbre métriques binaire (et n-aires)

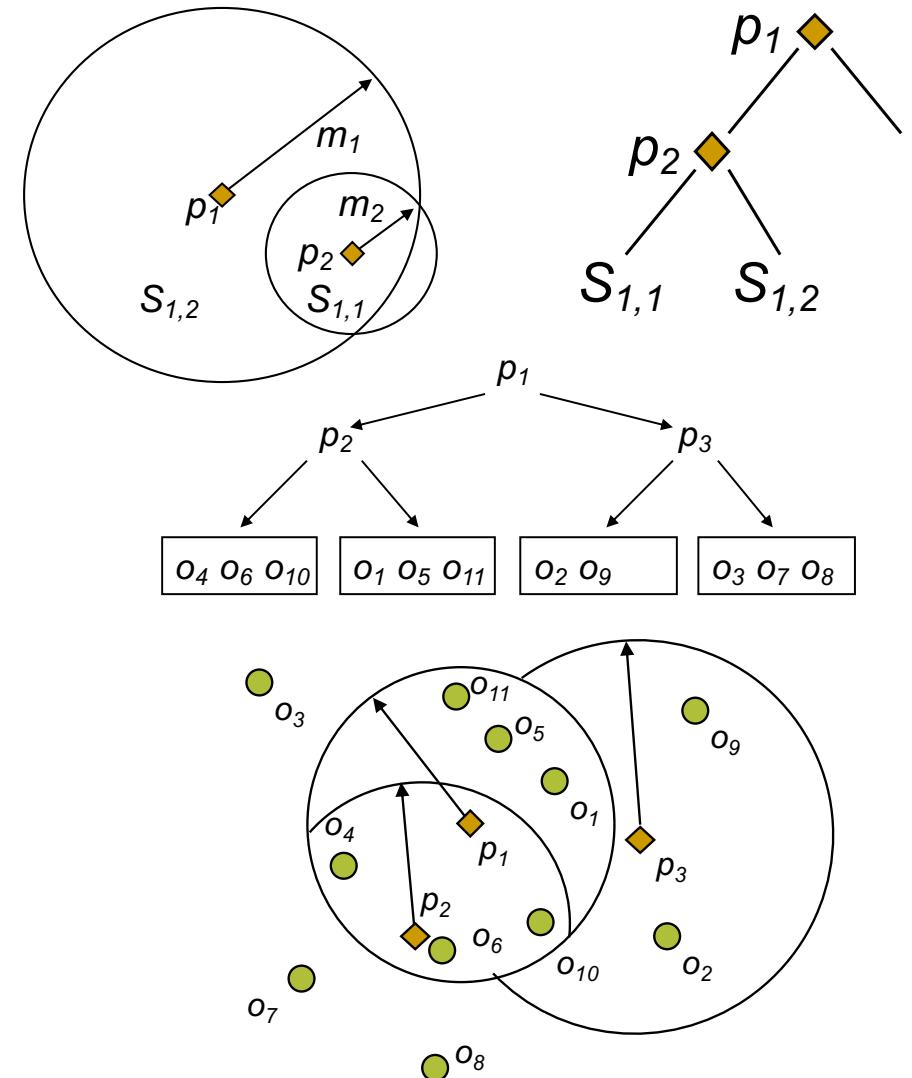
Partitionnement à l'aide de Balles

- Décomposition de l'espace en deux sous-ensembles S_1 et S_2 en utilisant un pivot $p \in \mathbb{U}$ et la distance médiane δ_m entre p et les objets de \mathbb{U}
- $S_1 = \{o_j \mid \delta(o_j, p) \leq \delta_m\}$
- $S_2 = \{o_j \mid \delta(o_j, p) \geq \delta_m\}$
- L'égalité des conditions \leq and \geq est utilisée pour équilibrer l'affectation des éléments quand la distance médiane n'est pas unique

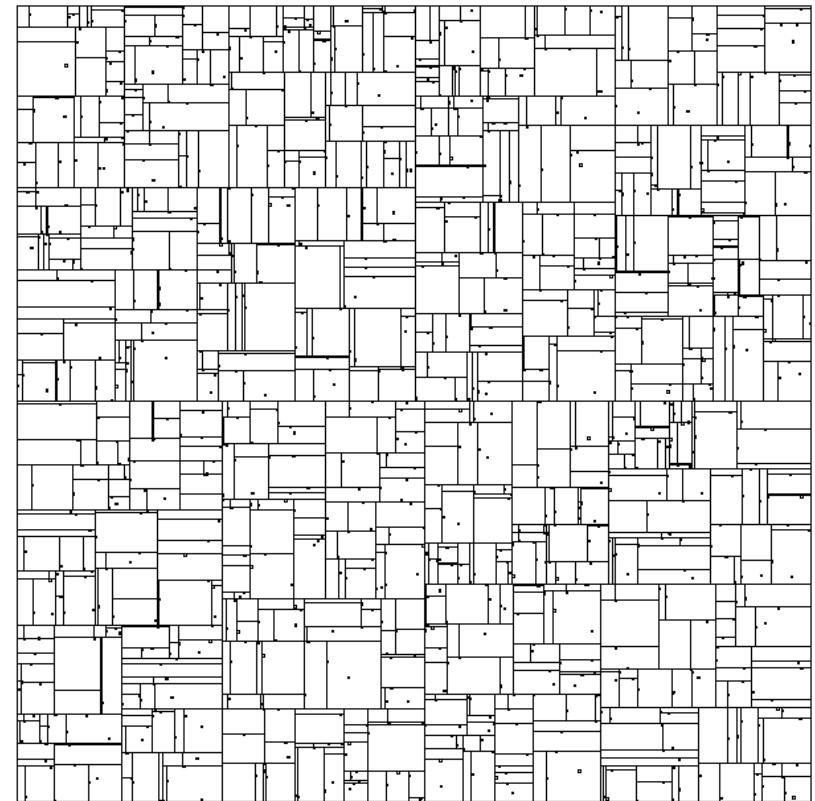
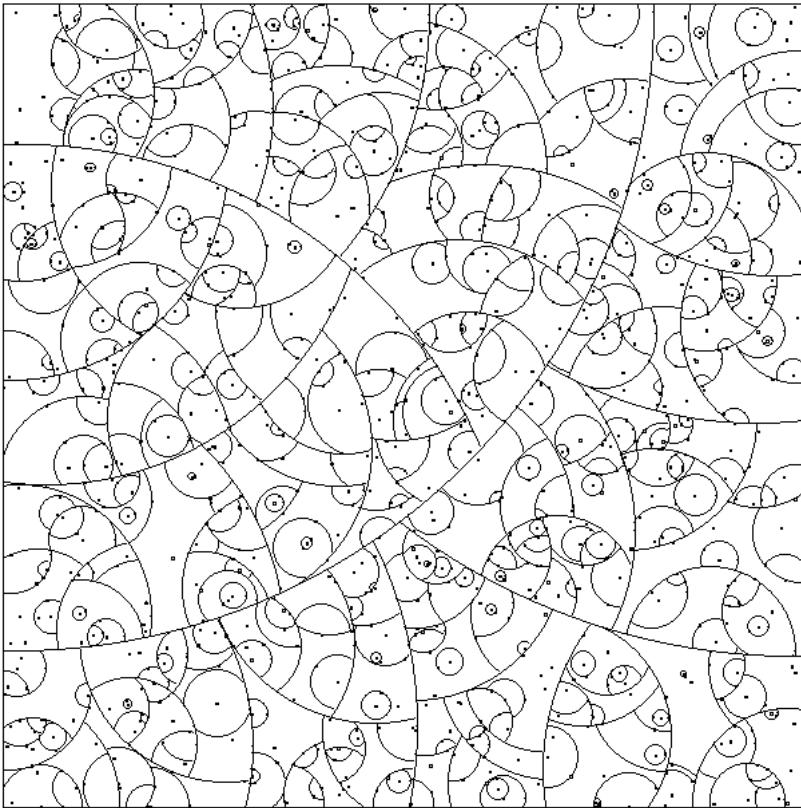


Vantage Point Tree (VPT)

- Application récursive du partitionnement par balle pour construire un arbre binaire
- Chaque noeud intermédiaire référence un pivot et une distance médiane
- Noeuds fils du sous-arbre gauche : éléments de S_1
- Noeuds fils du sous-arbre droit : éléments de S_2
- Noeuds feuilles référencent un ou plusieurs éléments selon la capacité souhaitée
- Les noeuds pivots font partie des candidats pour la recherche

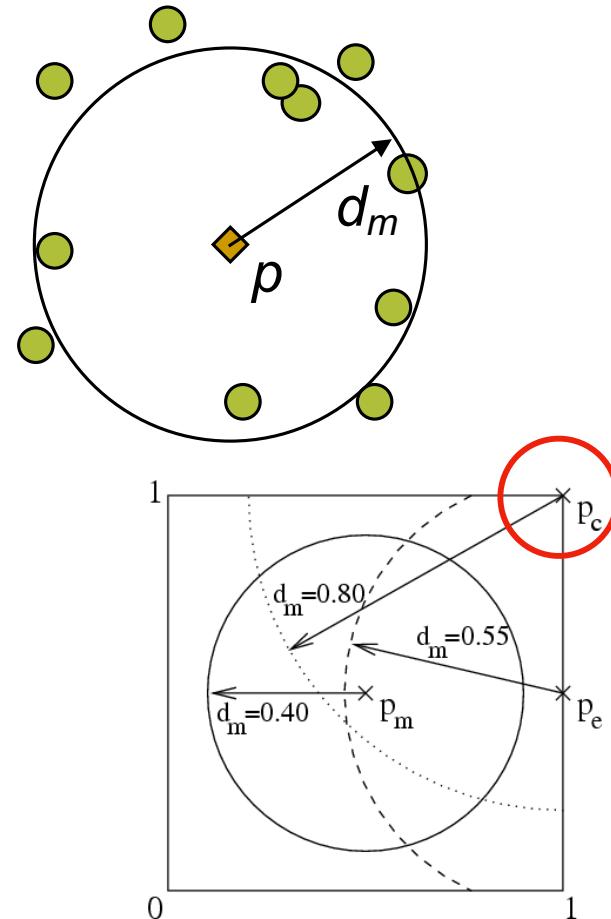


VPT vs Arbre K-d



VPT : Sélection du pivot

- Plusieurs stratégies
 - Sélection aléatoire d'un élément dans le sous-groupe
 - Utilisation d'un élément correspondant à la distance médiane du sous-groupe
 - Problème si les distances entre cet élément et les autres sont proches : nécessité de parcourir les deux sous-arbres
 - Sélection aléatoire d'un élément, calcul des distances avec les autres éléments, sélection de l'élément le plus éloigné comme pivot
 - ...

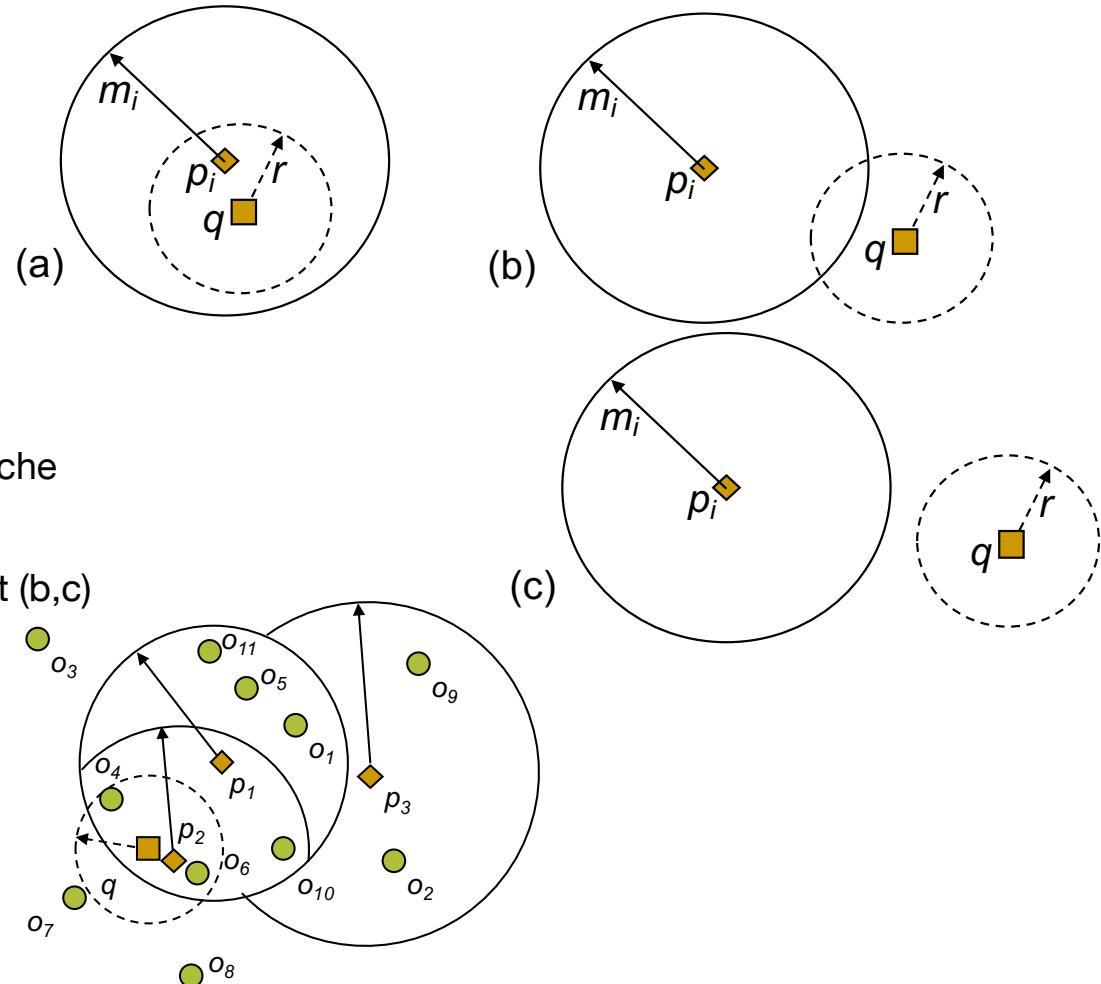


carré unitaire avec une distribution uniforme

probabilité d'avoir à entrer dans les deux régions

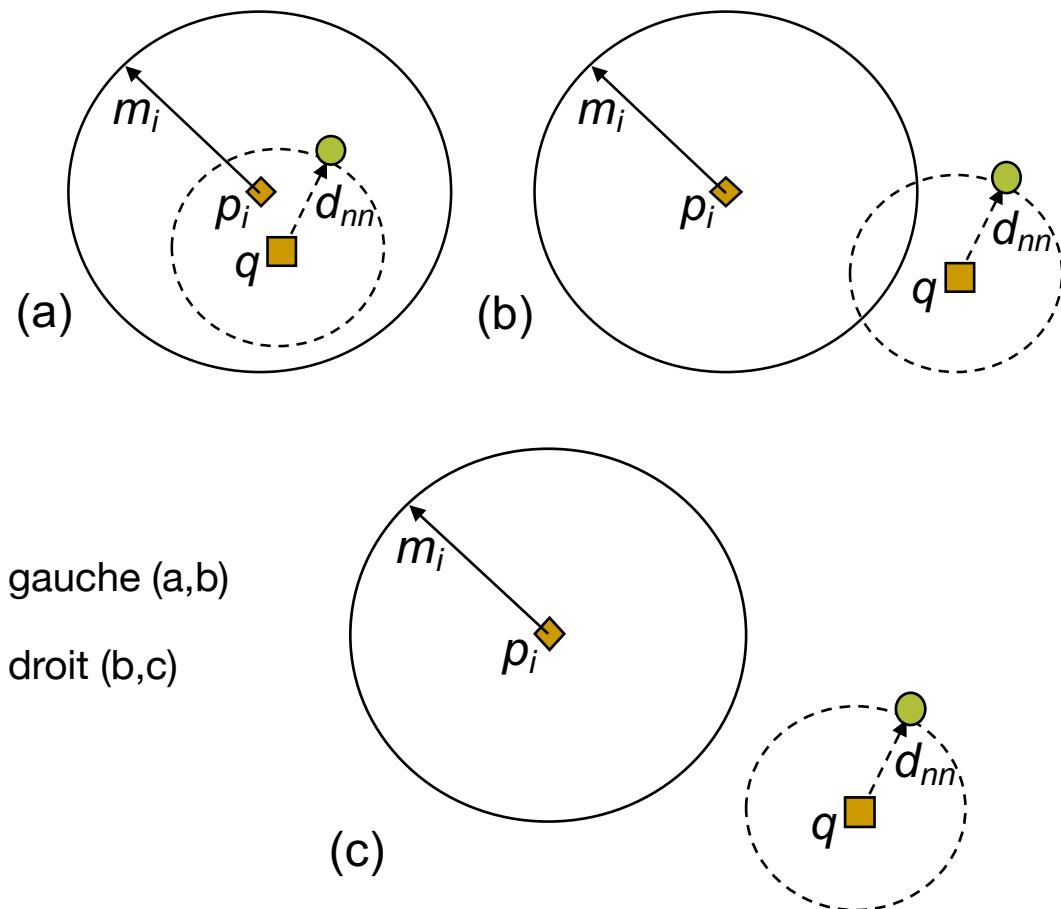
Recherche VPT : Voisins dans Région

- Soit une recherche $VPT(q, r)$
- Traversée de l'arbre depuis le noeud racine
- Pour chaque noeud interne (p_i, m_i)
 - si $\delta(q, p_i) \leq r$: ajout de p_i au résultat
 - si $\delta(q, p_i) - r \leq m_i$: recherche dans le sous-arbre gauche (a,b)
 - si $\delta(q, p_i) + r \geq m_i$: recherche dans le sous-arbre droit (b,c)
- Retour sur l'exemple: $R(q, r)$
 - Résultat : $\{o_4, o_6\}$
 - Nombre de distances : Brute = 14, VTP = 12



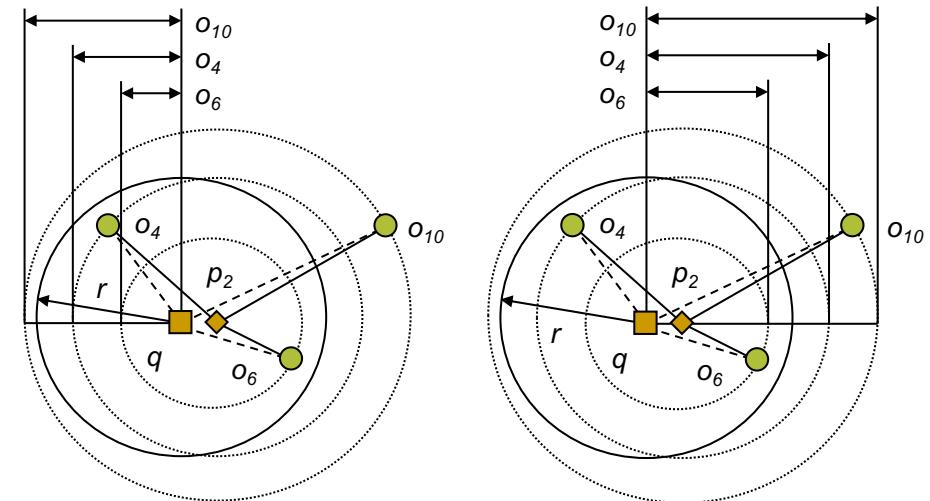
Recherche VPT : Plus proches voisins

- Soit une recherche $1 - NN(q)$
- Initialisation : $\delta_{nn} = \delta_{max}$ $NN = nil$
- Traversée de l'arbre depuis le noeud racine
- Pour chaque noeud interne (p_i, m_i)
 - si $\delta(q, p_i) \leq \delta_{nn}$: $\delta_{nn} \leftarrow \delta(q, p_i)$, $NN \leftarrow p_i$
 - si $\delta(q, p_i) - \delta_{nn} \leq m_i$: recherche dans le sous-arbre gauche (a,b)
 - si $\delta(q, p_i) + \delta_{nn} \geq m_i$: recherche dans le sous-arbre droit (b,c)
- Extension à une recherche k-NN:
 - utilisation de tableaux ordonnées $\delta_{nn}[k]$, $NN[k]$



Economie des calculs

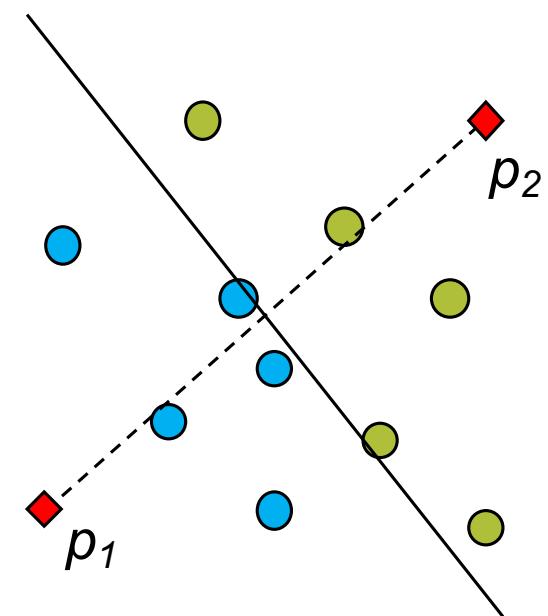
- Pour les noeuds feuilles, on peut économiser des calculs de distances entre o_i et q grâce aux distances connus entre o_i et le pivot p
- Plusieurs règles connues (contraintes sur les distances limites)
 - **Contrainte de distance Objet-Pivot**, Contrainte de distance Région-Pivot, Contrainte de distance Pivot-Pivot
- Estimation de $\delta(q, o_{10})$
 - Distances connues: $\delta(q, p_2), \delta(o_4, p_2), \delta(o_6, p_2), \delta(o_{10}, p_2)$
 - Limite Basse : $|\delta(p_2, o_{10}) - \delta(p_2, q)|$
 - si limite basse > r , o_{10} est disqualifié
 - Limite Haute : $|\delta(q, p_2) + \delta(p_2, o_{10})|$
 - si limite haute < r , o_{10} est directement qualifié



$$|\delta(q, p) - \delta(p, o)| < \delta(q, o) < \delta(q, p) + \delta(p, o)$$

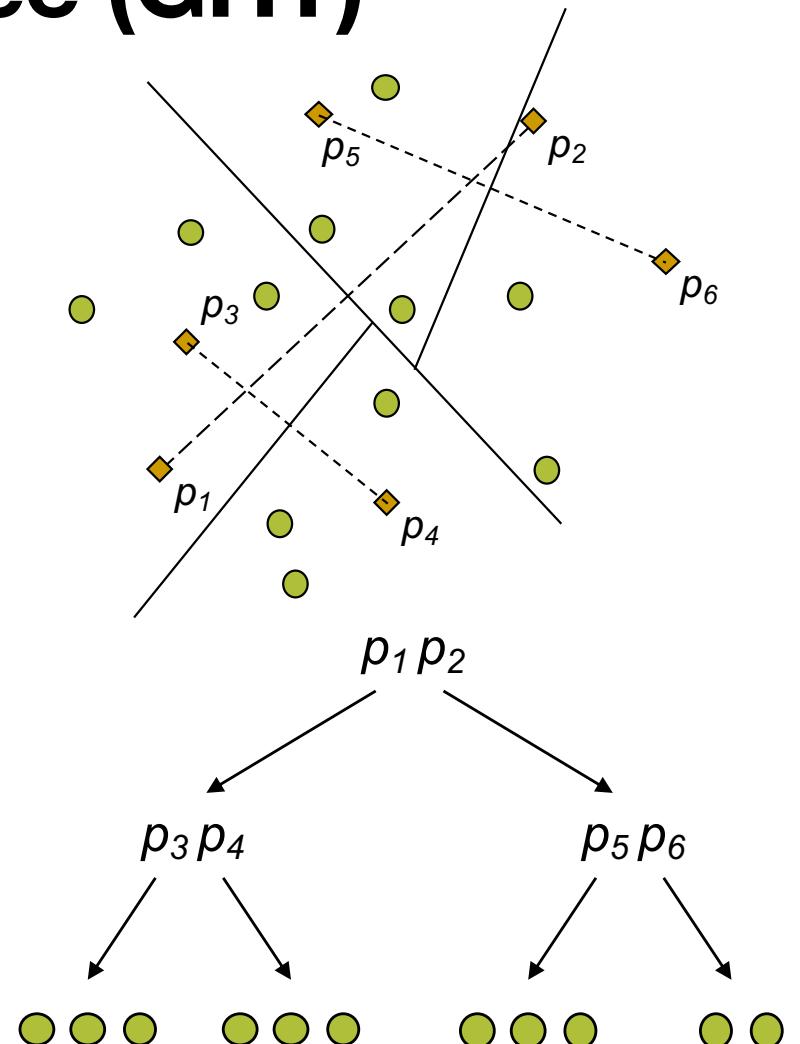
Partitionnement à l'aide d'un hyperplan

- Décomposition de l'espace entre deux sous-ensembles S_1 et S_2 en utilisant deux pivots $p_1, p_2 \in \mathbb{U}$
- $S_1 = \{o_j \mid \delta(p_1, o_j) \leq \delta(p_2, o_j)\}$
- $S_2 = \{o_j \mid \delta(p_1, o_j) \geq \delta(p_2, o_j)\}$
- Tous les objets de \mathbb{U} sont affectés à S_1 ou S_2 selon leur distance aux pivots
- Pas de garantie sur une division équilibrée contrairement au partitionnement par balle



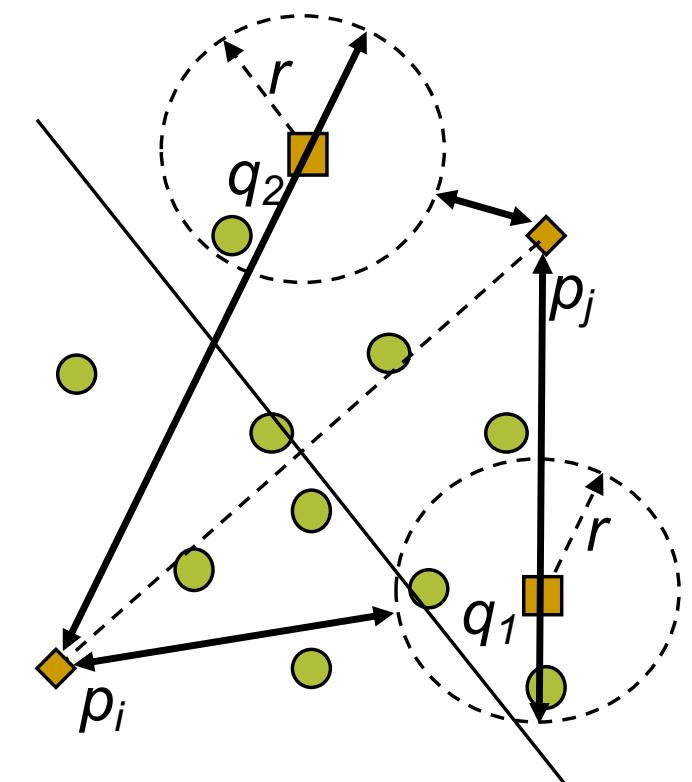
Generalized Hyperplane Tree (GHT)

- Application récursive du partitionnement par hyperplan pour construire un arbre binaire
- Chaque noeud intermédiaire référence deux pivots
- Noeuds fils du sous-arbre gauche : éléments de S_1
- Noeuds fils du sous-arbre droit : éléments de S_2
- Noeuds feuilles référence un ou plusieurs éléments selon la capacité souhaitée
- Les noeuds pivots font partie des candidats pour la recherche



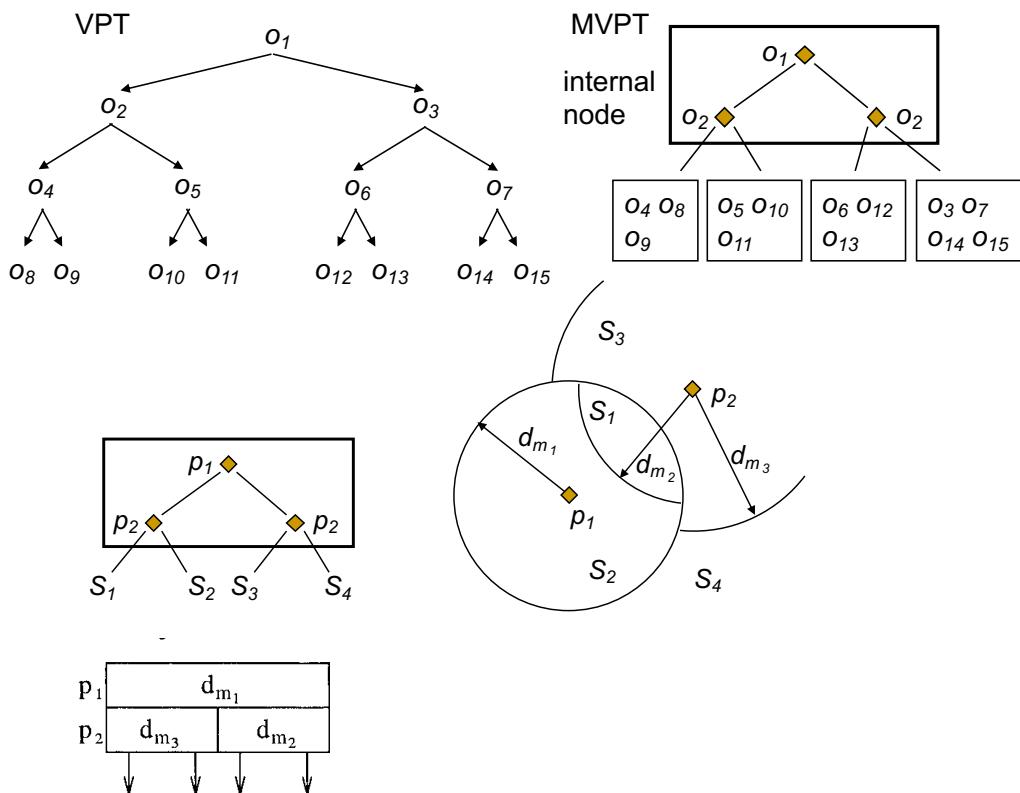
Recherche GHT : Voisins dans Région

- Soit une recherche $GHT(q, r)$
- Traversée de l'arbre depuis le noeud racine
- Pour chaque noeud interne (p_i, p_j)
 - si $\delta(q, p_i) \leq r$: ajout de p_i au résultat
 - si $\delta(q, p_j) \leq r$: ajout de p_j au résultat
 - si $\delta(q, p_i) - r \leq \delta(q, p_j) + r$: recherche dans le sous-arbre gauche
 - si $\delta(q, p_i) + r \geq \delta(q, p_j) - r$: recherche dans le sous-arbre droit

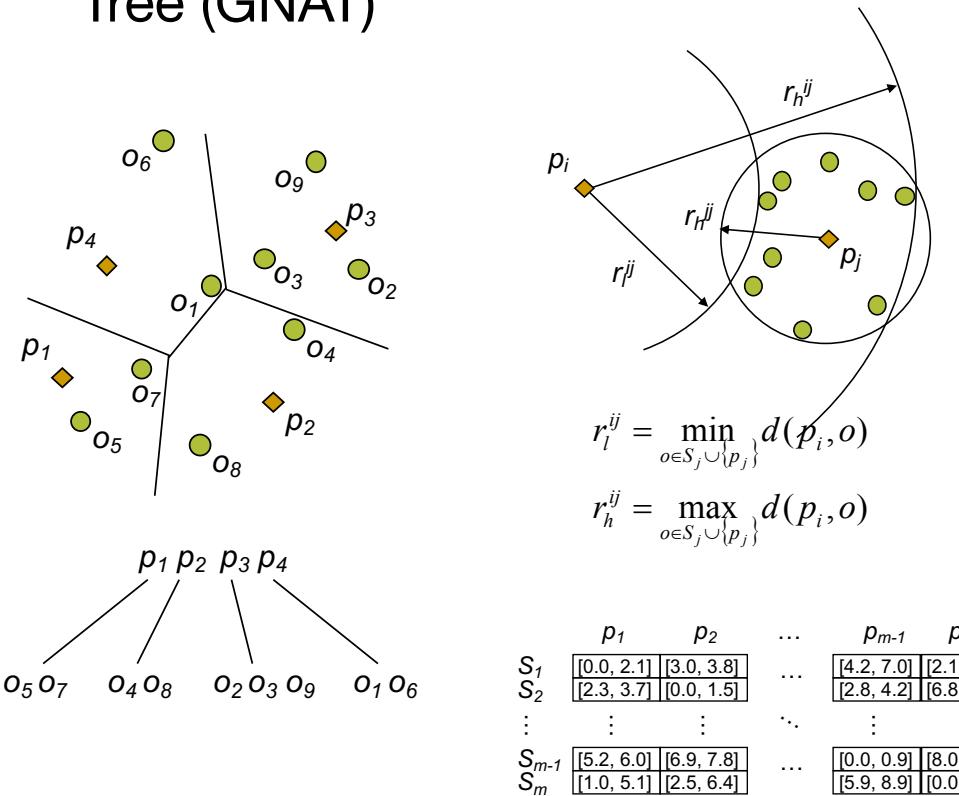


Variantes

- Multiple Vantage Point Tree (MVPT)



- Geometric Near-neighbor Access Tree (GNAT)

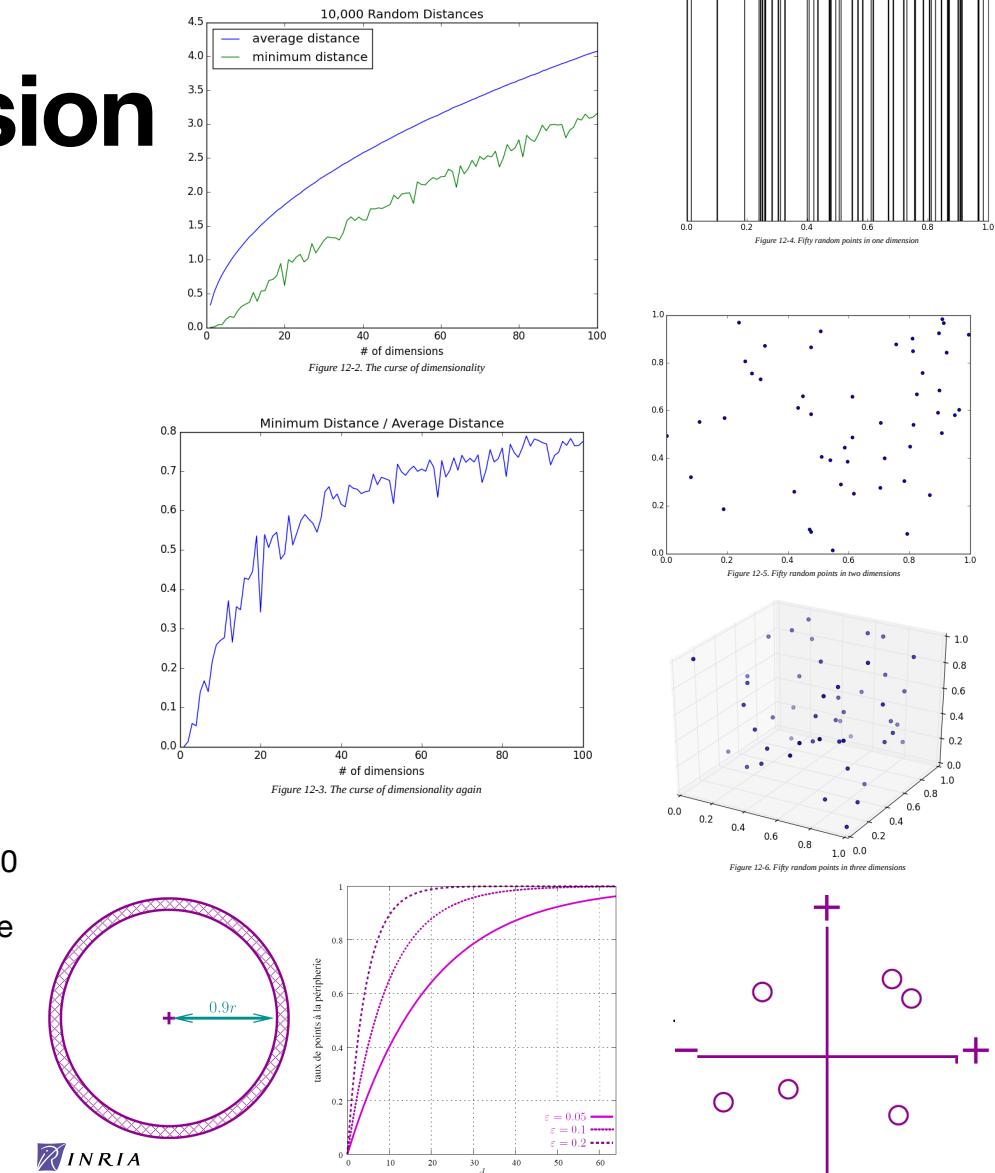


Evaluation des méthodes exactes

- Temps de calcul pour la recherche
 - n : nb d'éléments, d : nombre de dimensions
 - Force brute = $\Theta(nd)$
 - Ball Tree $\approx \Theta(d \log(n))$
 - Arbre kd
 - pour $d < 20$ $\approx \Theta(d \log(n))$
 - pour d plus grand $\approx \Theta(nd)$
- Le nombre de voisins n'affecte pas l'algorithme Force Brute mais rend les algorithmes Ball Tree et Arbre KD plus lents quand k augmente
- Le coût de la construction pour Ball Tree et Arbre kd peut-être amorti si le nombre de recherches est important

Malédiction de la dimension

- Les approches précédentes sont peu efficaces en grande dimension (ne filtre que peu de cellules)
 - Pour $k > 15$, il vaut mieux faire une recherche exhaustive ! (voir tp)
- Quelques problèmes
 - Vanishing Variance
 - La distance entre paires de points tend à être identique quand d augmente
 - Le point le plus proche est à une distance proche de la distance moyenne !
 - Phénomène de l'espace vide (sparsity)
 - Partition de l'espace vectoriel selon le signe des composants : $d = 100$
 - Bcp de cellules à créer (10^{30}), très peu de cellules sont remplis, bonne répartition difficile à obtenir
 - Proximité des frontières
 - Les objets sont proches des surfaces de séparation avec une très grande probabilité: le plus proche voisin d'un point appartient à une cellule différente



Recherche approchée par hâchage

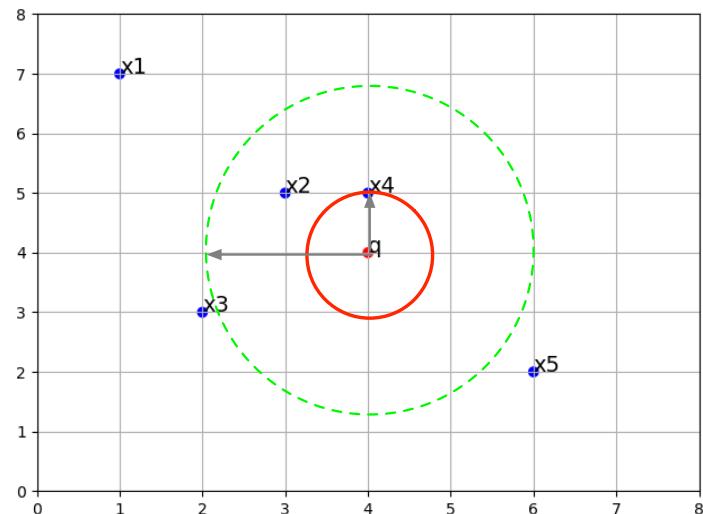
Recherche approchée

- Relâcher le problème de recherche pour trouver des voisins suffisamment proches
- Principe : Utiliser un facteur d'approximation c pour contrôler la précision de la recherche
 - $c \searrow$: plus précis, $c \nearrow$: moins précis
 - $c = 1 \Leftrightarrow$ recherche exacte
 - généralement $c \nearrow \Leftrightarrow$ complexité de recherche \searrow
 - compromis entre la qualité des résultats et la vitesse (complexité de la recherche)
- Les résultats ne vont plus être les voisins les plus proches mais les “presque” plus proches
 - Mais ce n'est pas forcément grave en pratique car : la similarité entre objets est souvent subjective et la recherche est itérative
 - Besoin d'évaluer la qualité de la recherche
- Comment trouver c ?
 - dépend des hyperparamètres de la méthode (ex: nombre max de feuilles visitées), mesurable empiriquement
 - dans certains cas, la théorie permet de connaître c grâce à une relation explicite avec les hyperparamètres

Recherche approchée

$$\delta(x, q) \leq c \cdot \max(k_{\min}_{x \in D} \delta(x, q)), c \geq 1$$

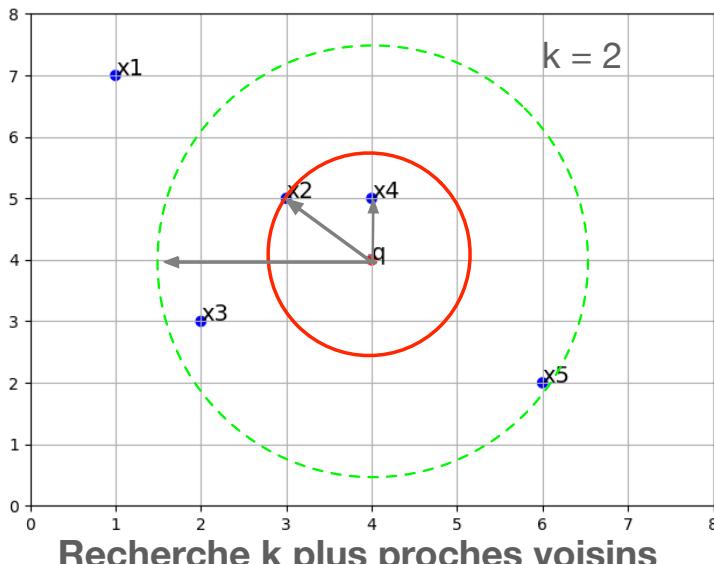
$$\delta(x, q) \leq c \cdot \min_{x \in D} \delta(x, q), c \geq 1$$



Recherche plus proche voisin

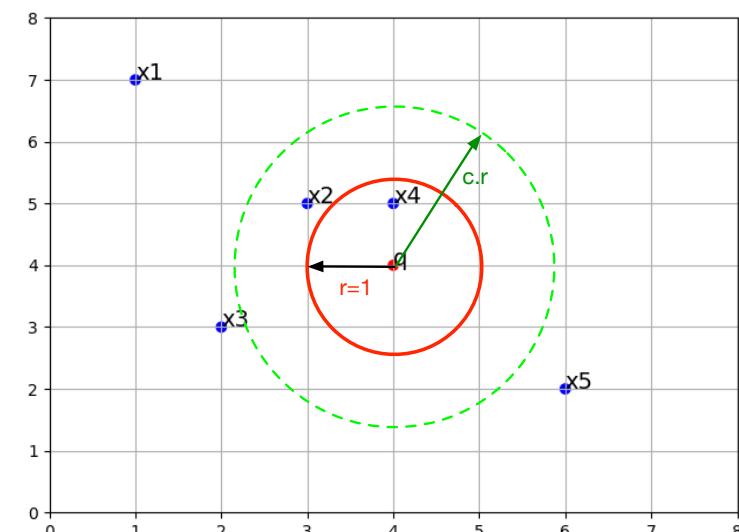
(utilisation de la distance au plus proche réel)

c : contrôle du “presque” plus proche



Recherche k plus proches voisins
(utilisation de la distance au plus éloigné des proches voisins)
 c : contrôle du “presque” plus proches

$$\{x \in \mathbb{U} \mid \delta(x, q) \leq c \cdot r, c > 1\}$$



Recherche dans région

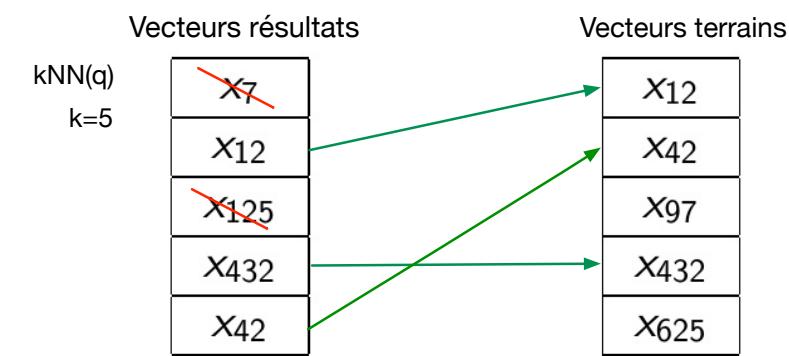
(utilisation de la distance au plus proche réel)
 c : contrôle du “presque” plus proche

Recherche approchée: Evaluation

- L'évaluation de performance d'une recherche approchée doit considérer
 - La qualité du résultat
 - La vitesse de la recherche
- Compromis entre les deux : un bon algorithme/méthode doit améliorer la vitesse avec une précision importante dans les résultats

Recherche approchée : Qualité du résultat

- Est-ce que les vecteurs retrouvés correspondent aux vrais voisins ?
- Utilisation des mesures classiques de la recherche d'informations
 - Evaluation sur un ensemble de requêtes pour garantir des résultats de façon statistique
- **Rappel** : Est-ce que j'ai retrouvé tous les vecteurs attendus ?
 - $$R = \frac{\text{nb vecteurs attendus retrouvés}}{\text{nb vecteurs attendus}}$$
- **Précision** : Est-ce que tous les vecteurs retrouvés sont ceux attendus ?
 - $$P = \frac{\text{nb vecteurs attendus retrouvés}}{\text{nb vecteurs retrouvés}}$$
- Erreur: $E = 1 - P$
- Pour une recherche (k)-NN, le “nb de vecteurs retrouvés = “nb de vecteurs attendus” = k donc $P = R$



Autre formulation:

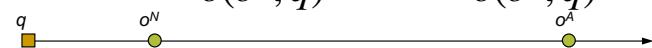
$$P = \frac{|S \cap S_a|}{|S_a|}$$

$$R = \frac{|S \cap S_a|}{|S|}$$

S : objets attendus, terrains ou résultat algorithme exact

S_a : résultats algorithme approchée

Recherche approchée: Qualité du résultat

- Existence d'autres mesures dont:
 - Erreur relatives sur les distances (ED)
 - Compare la distance entre la requête et les résultats exactes et approchées (o^n, o^a)
 - $$ED = \frac{\delta(o^a, q) - \delta(o^n, q)}{\delta(o^n, q)} = \frac{\delta(o^a, q)}{\delta(o^n, q)} - 1$$


- Erreur sur les positions (EP)
 - Mesure l'écart entre les rangs des résultats approximatifs et exacts.

- $$EP = \frac{\sum_{i=1}^{|S_a|} |S_n(o_i) - S_a(o_i)|}{|S_n| \cdot |X|}$$

- $S_i(o)$: position de o dans la liste ordonnée des résultats

Suppose $|X|=10,000$

Let us consider a $10\text{-NN}(q)$:

- $S=\{1,2,3,4,5,6,7,8,9,10\}$
- $S^{A1}=\{2,3,4,5,6,7,8,9,10,11\}$ the object 1 is missing
- $S^{A2}=\{1,2,3,4,5,6,7,8,9,11\}$ the object 10 is missing

As also intuition suggests:

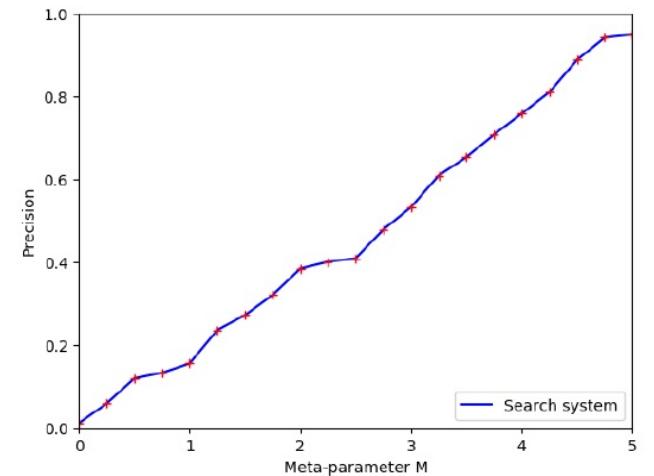
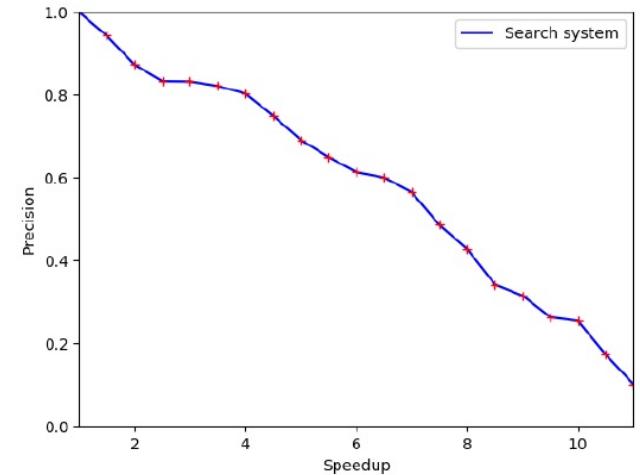
- In case of S^{A1} , $EP = 10 / (10 \cdot 10,000) = 0.0001$
- In case of S^{A2} , $EP = 1 / (10 \cdot 10,000) = 0.00001$

Recherche approchée : Vitesse de réponse

- On peut calculer l'accélération en temps d'exécution entre la recherche approchée et une recherche de référence (force-brute ou autre algo)
 - $acc = \frac{t_{base}}{t_s}$ (dépend de l'algorithme et de son implémentation)
- On peut aussi déterminer le ratio entre le nombre d'éléments inspectés et le nombre total de données
 - $f = \frac{nb\ elements\ inspectes}{n}$ (dépend uniquement de l'algorithme)

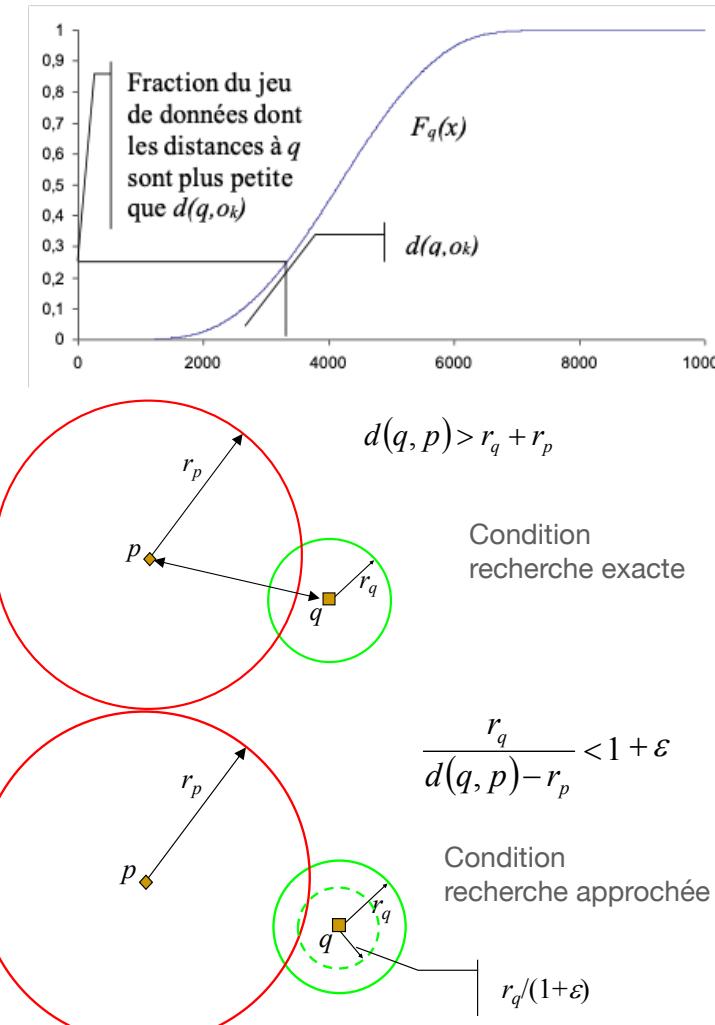
Recherche approchée : Procédure d'évaluation

- Sélection d'un ensemble de requêtes
- Calcul des voisins de chaque requête par une méthode exacte (groundtruth)
- Identification des hyperparamètres de la méthode approchée
- Pour chaque recherche et chaque hyperparamètre (ex: nb de feuilles visitées)
 - Exécution de la recherche
 - Calcul des mesures
- Moyenne des mesures sur l'ensemble des requêtes
- Visualisation des résultats de l'évaluation
 - Evaluation d'une mesure vs évaluation d'une autre mesure
 - Evaluation d'une mesure vs hyperparamètres



Recherche approchée

- Stratégie de base
 - Procéder à un arrêt précoce de l'algorithme de recherche
 - Arrêt après avoir exploré un certain pourcentage du jeu de donnée
 - Arrêt après un temps spécifié
 - Arrêt après avoir évaluer le résultat en cours : celui-ci contient 10% des objets les plus proches de la requête
 - ...
- Relâchement des conditions d'élagage
 - Elimination des régions ayant une faible probabilité de contenir des objets appartenant aux résultats
 - Utiliser une méthode de réduction de dimension (ACP)
- Méthodes spécifiques
 - à base de hachage, de quantification, de graphe de proximité



Fonction et table de hachage

- Fonction de hachage
 - à partir d'une donnée quelconque, calcule une empreinte numérique servant à identifier rapidement la donnée initiale
 - $h : \mathbb{U} \rightarrow [0, H] \subset \mathbb{N}$
- Table de hachage : structure de donnée
 - stockant des valeurs selon des indices (~tableau)
 - les indices des valeurs sont déterminés avec une fonction de hachage
 - Complexité (meilleur cas): $\Theta(1)$ pour l'accès, l'insertion et la suppression

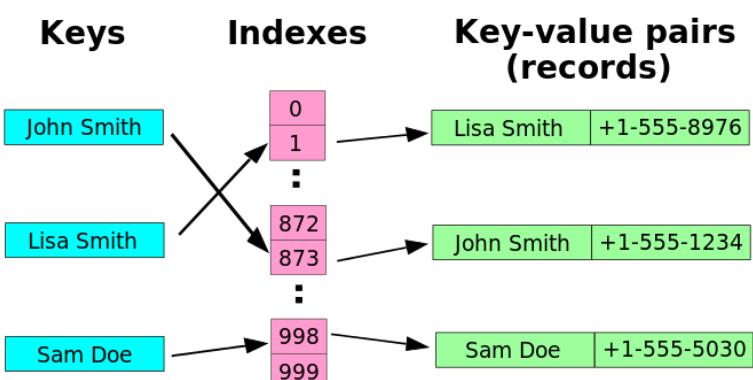
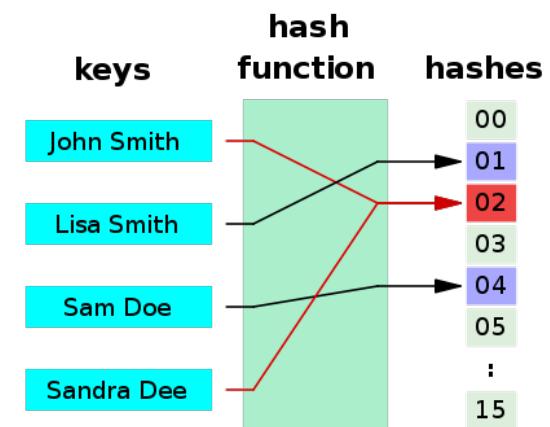
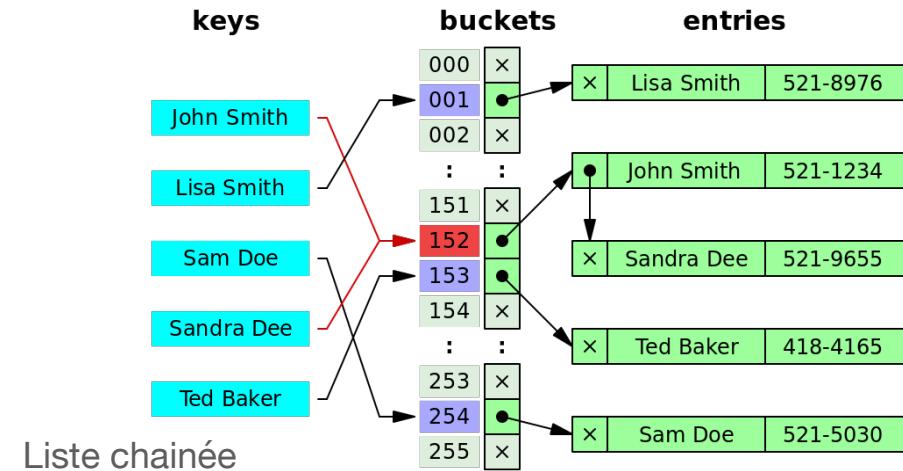
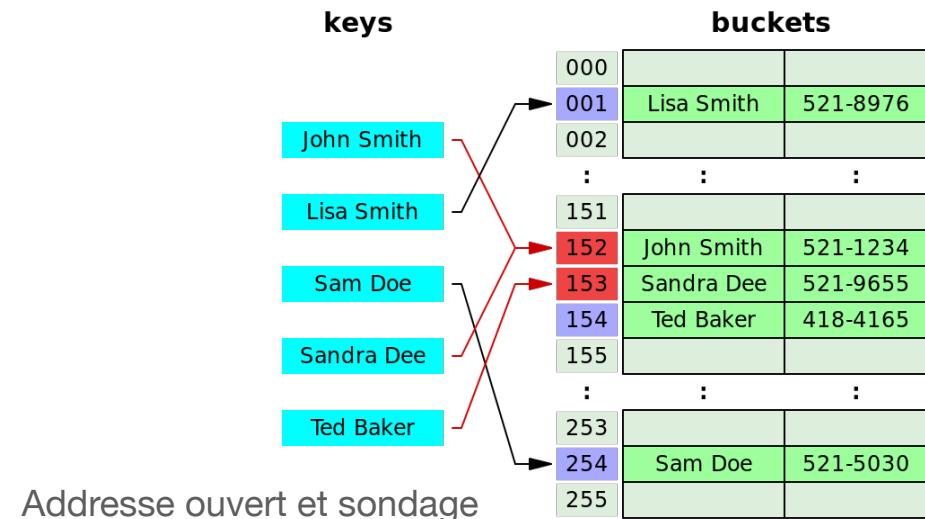


Table de hachage : Collisions

- Collisions
 - Plusieurs valeurs peuvent avoir la même valeur de hachage posant un problème pour le stockage et la récupération
 - Solutions pour résoudre le problème
 - Liste chainée
 - Adressage ouvert et sondage
 - La complexité n'est plus en $\Theta(1)$
- Objectif classique
 - Concevoir des fonctions de hachage qui éparpillent au mieux la donnée dans la table



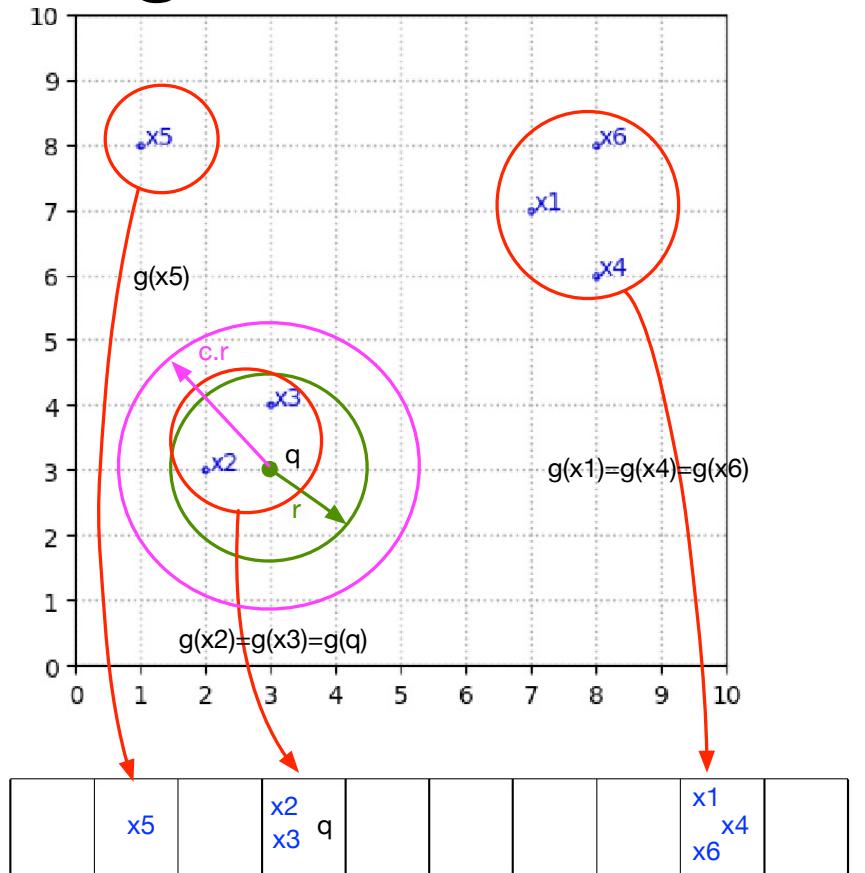
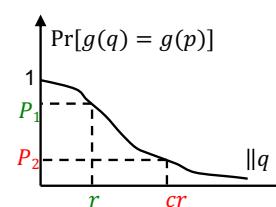
Liste chainée



Adresse ouvert et sondage

LSH: Locality Sensitive Hashing

- Idée de LSH
 - Exploiter les notions de fonctions et de table de hachage pour mettre en oeuvre la recherche par similarité
 - Mettre à profit le phénomène de collision pour regrouper les objets similaires
 - Conception d'une fonction de hachage adaptée: *Locality-Sensitive hash function*
- Définition formelle
 - Trouver une fonction de hachage $g : \mathbb{U} \rightarrow \mathbb{N}$ tel que $\forall x, q \in \mathbb{R}^d$:
 - si $\delta(x, q) < r$ alors $\Pr[g(q) = g(x)] \geq p_1$
 - si $\delta(x, q) \geq c \cdot r$ alors $\Pr[g(q) = g(x)] \leq p_2$
 - $p_1 \gg p_2$



La probabilité de collision des objets similaires est forte
La probabilité de collision des objets dissimilaires est faible

Fonction de hachage LSH

- Projection sur une droite tirée aléatoirement (distributions stables)

- coéficients $a = (a_1, a_2, \dots, a_d)$

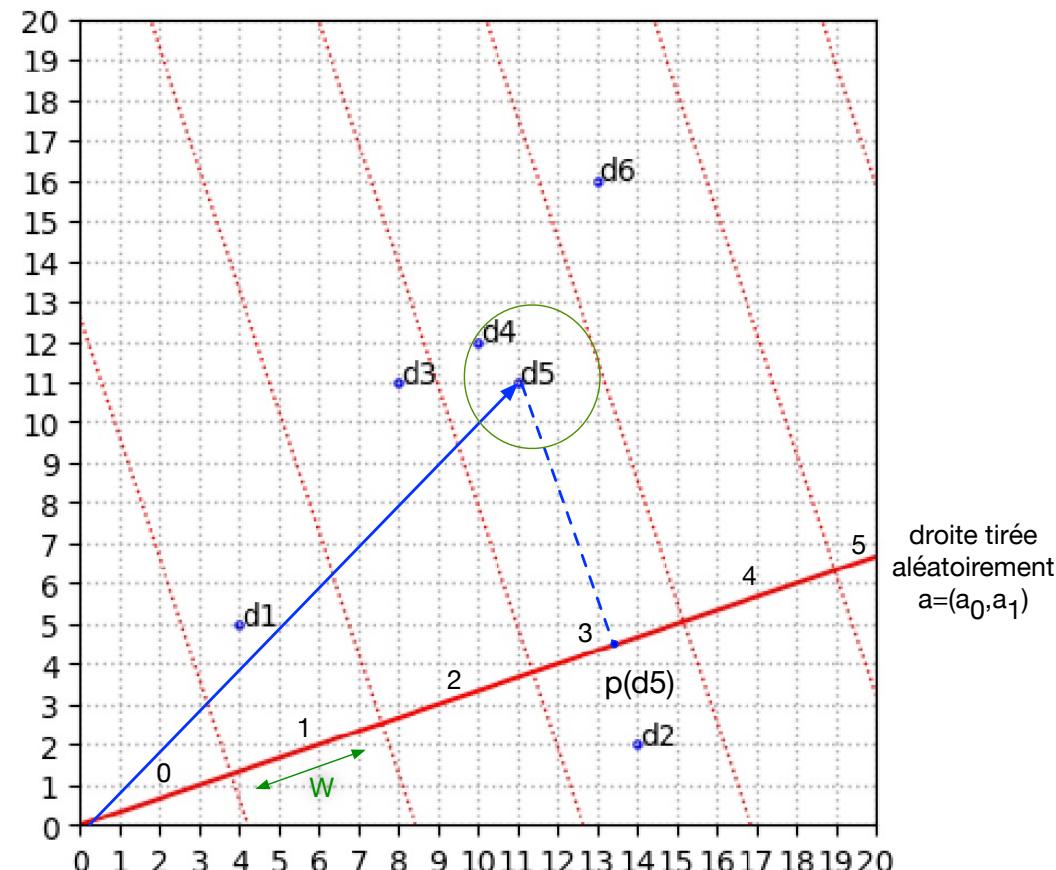
- $p(x) = \langle x, a \rangle = \sum_{i=1}^d x_i \cdot a_i$

- Quantification scalaire (valeur entière) avec un pas W correspondant à la taille des “bacs”

- $Q(x) = \left\lfloor \frac{x}{W} \right\rfloor$

- Fonction de hachage

- $g(x) = Q(p(x)) = \left\lfloor \frac{\langle x, a \rangle}{W} \right\rfloor = \left\lfloor \frac{\sum_{i=1}^d x_i \cdot a_i}{W} \right\rfloor$



Construction d'index avec LSH

Algorithm 1: Structure for a LSH index (v_0).

```

1 structure LSHIndex
2   a:  $\mathbb{R}^d$ 
3   W: Float
4   table: Array of lists
5 end

```

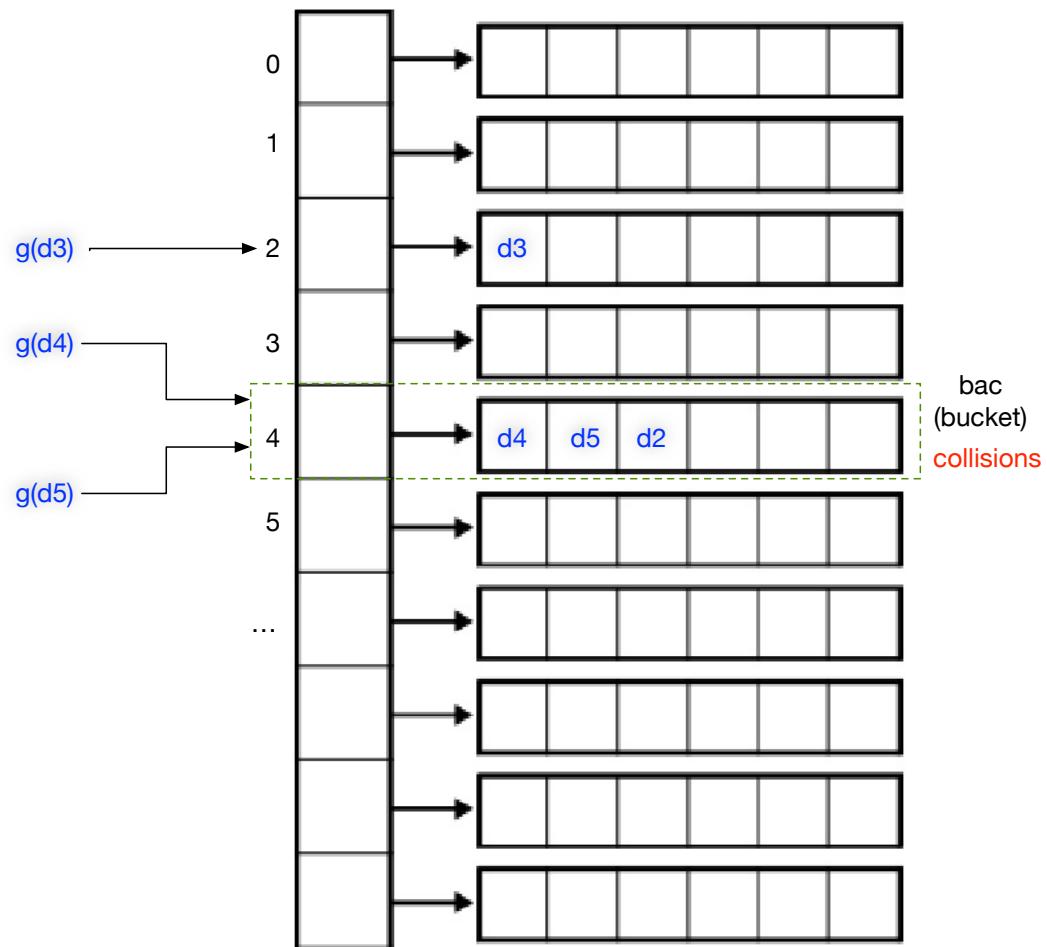
Algorithm 2: LSH index construction (v_0).

```

1 function build_index( $\mathcal{D}: \mathbb{R}^{n \times d}$ ,  $W: \text{Float}$ ): LSHIndex
2   index: LSHIndex
3   g: Integer
4   for  $i$  from 1 to  $d$  do
5     | index.a[i]  $\sim \mathcal{N}(0, 1)$            Tirage aléatoire
6   end                               du vecteur unitaire a
7   index.a  $\leftarrow \text{L2\_normalize(index.a)}$ 
8   index.W  $\leftarrow W$ 
9   for  $I \in \text{index.table}$  do
10    |  $I \leftarrow \emptyset$ 
11  end
12  for  $x \in \mathcal{D}$  do
13    |  $g \leftarrow \lfloor \frac{\langle x, \text{index.a} \rangle}{\text{index.W}} \rfloor$  Hachage de chaque objet
14    | index.table[g].append(x)           et placement dans la
15  end                                table de hachage
16 return index
17 end

```

Table de hachage 1

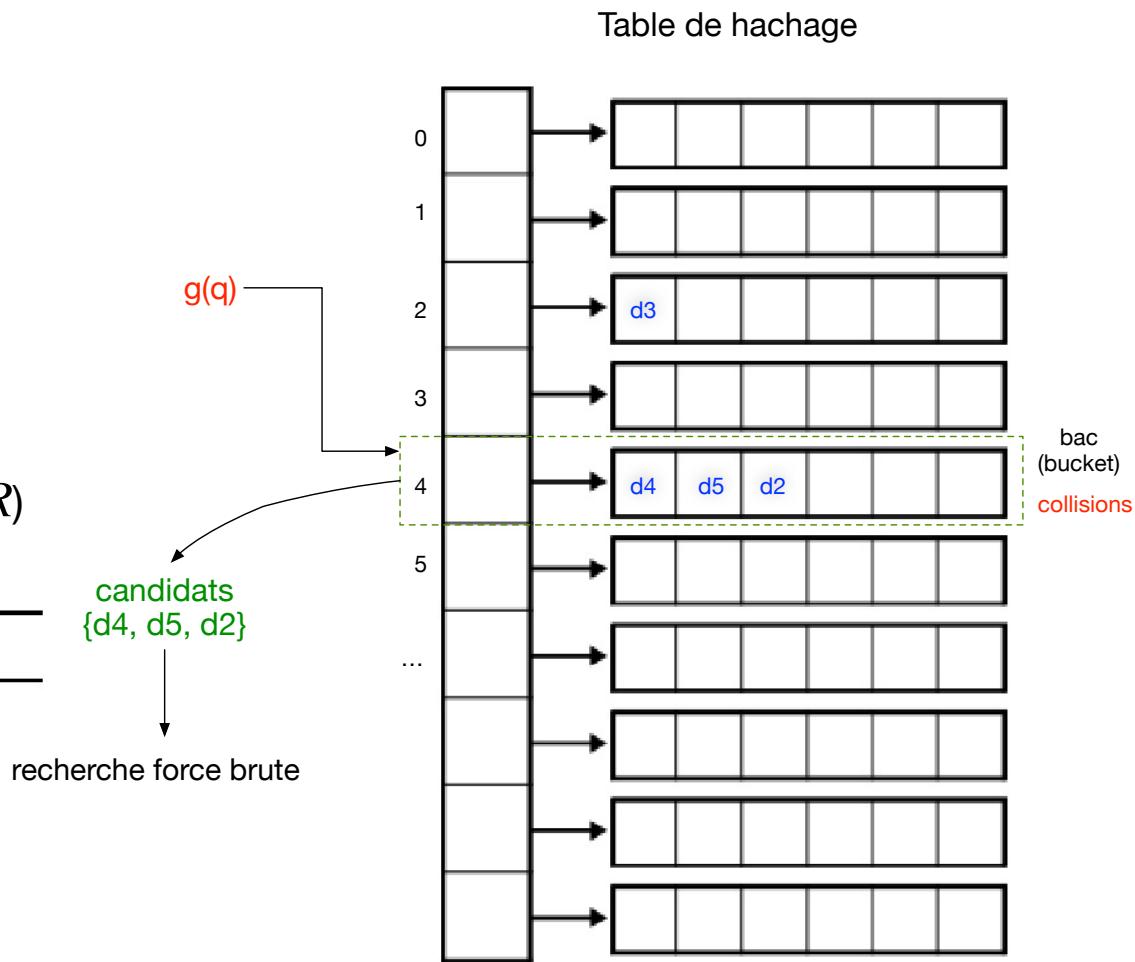


Recherche avec un index LSH

- Etapes
 - calcul de l'indice de hachage pour q
 - récupération des candidats du bac
 - force brute appliquée aux candidats
 - Arrêt dès que la condition de recherche approchée est satisfaite (ex: $\delta(x, q) < c \cdot R$)

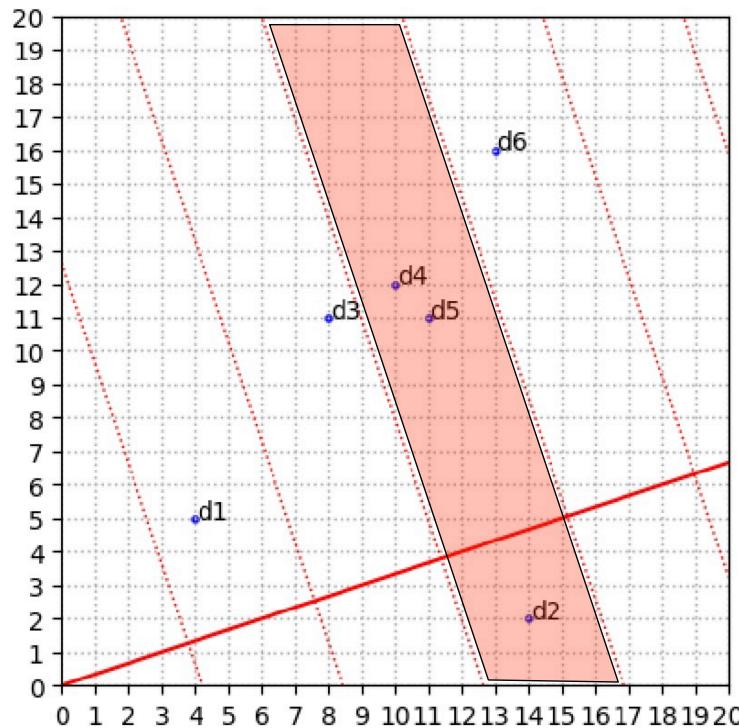
Algorithm 3: LSH index search (v0).

```
1 function search_index(index: LSHIndex, q:  $\mathbb{R}^d$ ): Set of vectors
2   short_list: Set of vectors in  $\mathbb{R}^d$ 
3   g: Integer
4    $g \leftarrow \lfloor \frac{\langle q, \text{index}.a \rangle}{\text{index}.W} \rfloor$  Hachage de q
5   short_list  $\leftarrow \text{index.table}[g]$ 
6   return short_list
7 end
```

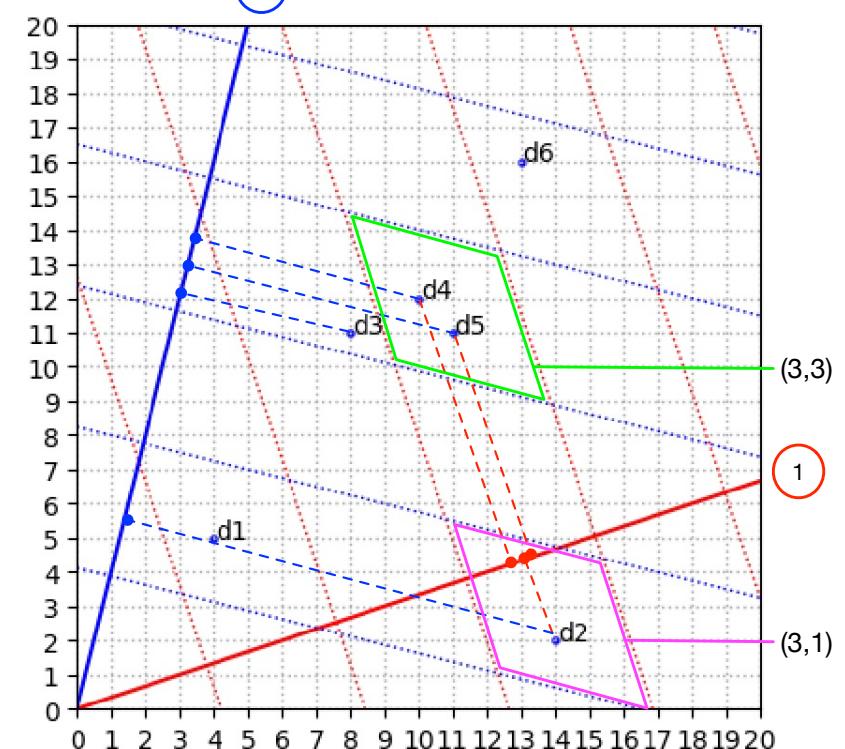
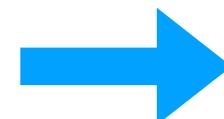


Utilisation de plusieurs projections

- $\delta(d1,d5) \nearrow \Rightarrow g(d1) \neq g(d5)$
- $\delta(d4,d5) \searrow \Rightarrow g(d4) = g(d5)$
- $\delta(d2,d5) \nearrow$ MAIS $g(d2) = g(d5)$



- $g(d4) = (3,3)$
- $g(d5) = (3,3)$
- $g(d2) = (3,1)$



Utilisation de plusieurs projections

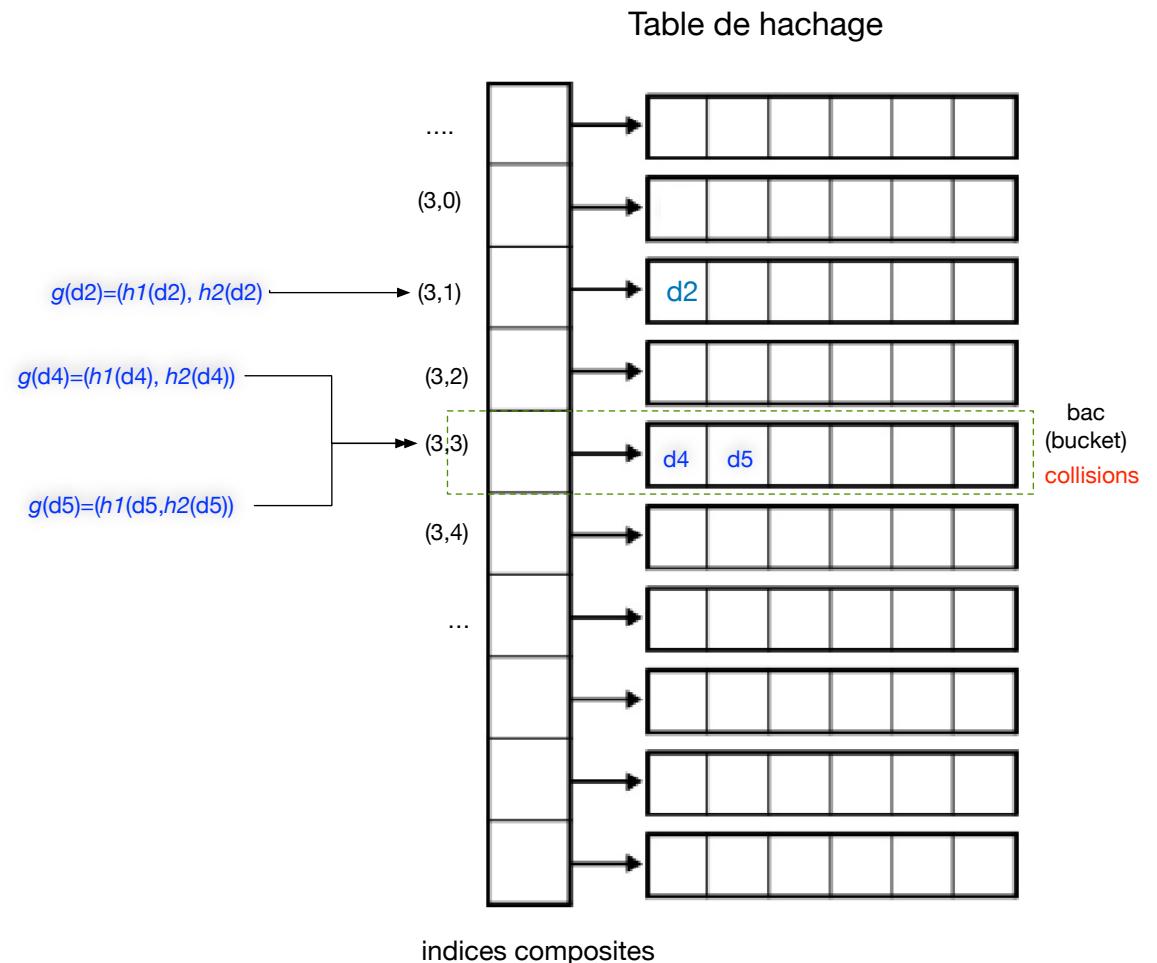
- Echantillonage de m fonctions de hachage

$$H = \{h_1(\cdot), h_2(\cdot), \dots, h_m(\cdot)\}$$

$$h_i(x) = \left\lfloor \frac{\langle x, a \rangle + b_i}{W} \right\rfloor$$

- Concaténation des fonctions

$$g(x) = (h_1(\cdot), h_2(\cdot), \dots, h_m(\cdot))$$



Construction d'index avec plusieurs projections

Algorithm 4: Structure for a LSH index (v0.5).

```
1 structure LSHIndex
2   |   a: array of  $m$  vectors in  $\mathbb{R}^d$ 
3   |   b: vector in  $\mathbb{R}^m$            - m droites
4   |   W: Float
5   |   table: Array of lists
6 end
```

Algorithm 5: LSH index construction (v0.5).

```
1 function build_index( $\mathcal{D}: \mathbb{R}^{n \times d}$ ,  $W: \text{Float}$ ): LSHIndex
2   |   index: LSHIndex
3   |   g: Vector in  $\mathbb{N}^m$ 
4   |   for  $i$  from 1 to  $m$  do
5   |     |   for  $j$  from 1 to  $d$  do
6   |     |     |   index.a[i][j] ~  $\mathcal{N}(0, 1)$ 
7   |     |   end
8   |     |   index.a[i] ← L2_normalize(index.a[i])
9   |     |   index.b[i] ~  $\mathcal{U}(0, W)$ 
10  |   end
11  |   index.W ←  $W$ 
12  |   for  $l \in \text{index.table}$  do
13  |     |   l ←  $\emptyset$ 
14  |   end
15  |   for  $x \in \mathcal{D}$  do
16  |     |   for  $i$  from 1 to  $m$  do
17  |     |     |   g[i] ←  $\left\lfloor \frac{\langle x, \text{index.a}[i] \rangle + \text{index.b}[i]}{\text{index.W}} \right\rfloor$ 
18  |     |   end
19  |     |   index.table[g].append(x)
20  |   end
21  |   return index
22 end
```

Tirage aléatoire des
m droites

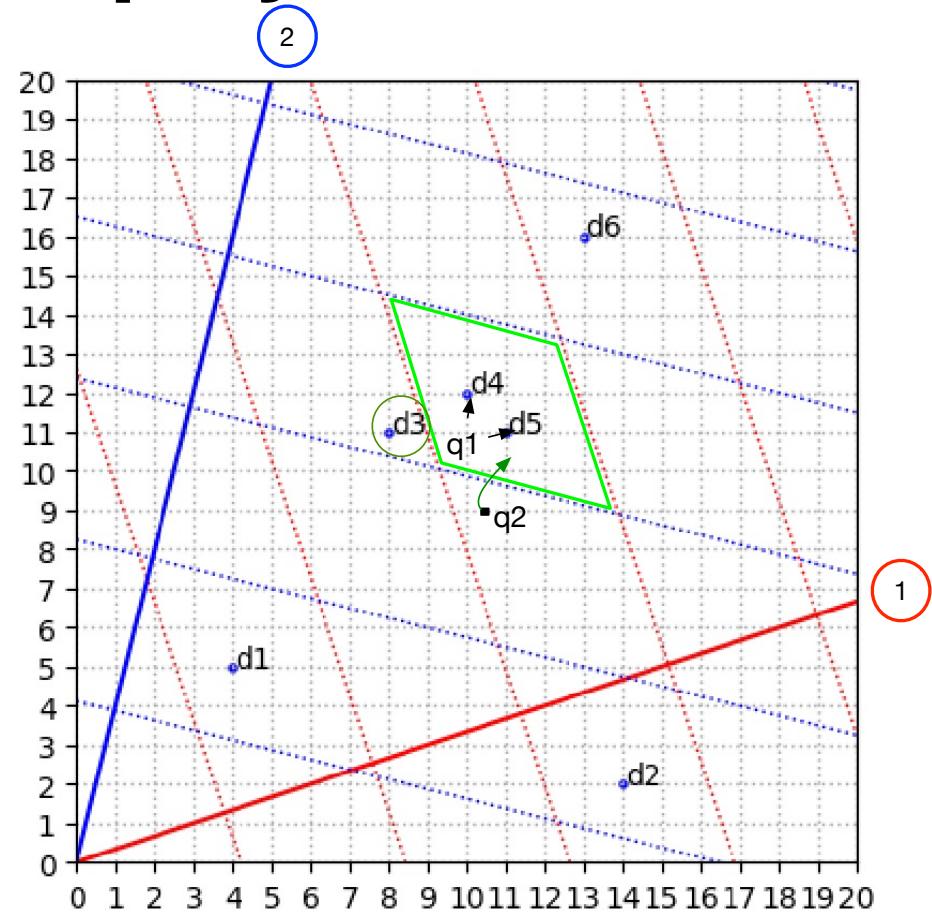
Calcul des m indices
de hachage pour chaque
objet

Recherche avec plusieurs projections

- Etapes
 - calcul des indices de hachage pour q
 - récupération des candidats du bac
 - force brute appliquée aux candidats

Algorithm 6: LSH index search (v0.5).

```
1 function search_index(index: LSHIndex, q:  $\mathbb{R}^d$ ): Set of vectors
2     short_list: Set of vectors in  $\mathbb{R}^d$ 
3     g: Vector in  $\mathbb{N}^m$ 
4     for i from 1 to m do
5          $g[i] \leftarrow \lfloor \frac{\langle q, \text{index.a}[i] + \text{index.b}[i] \rangle}{\text{index.W}} \rfloor$ 
6     end
7     short_list  $\leftarrow$  index.table[g]
8     return short_list
9 end
```



Bac plus restreint => meilleur précision mais rappel limité

Utilisation de plusieurs tables

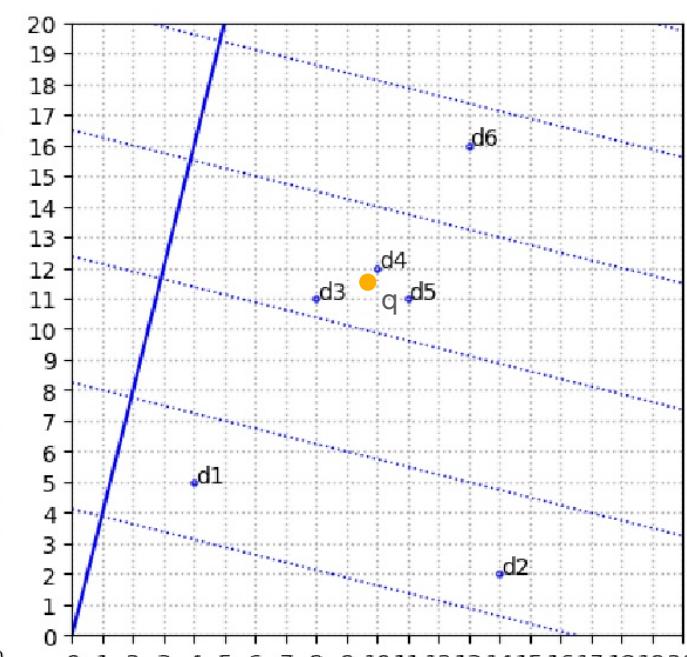
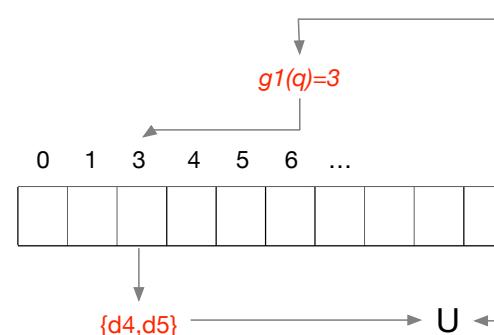
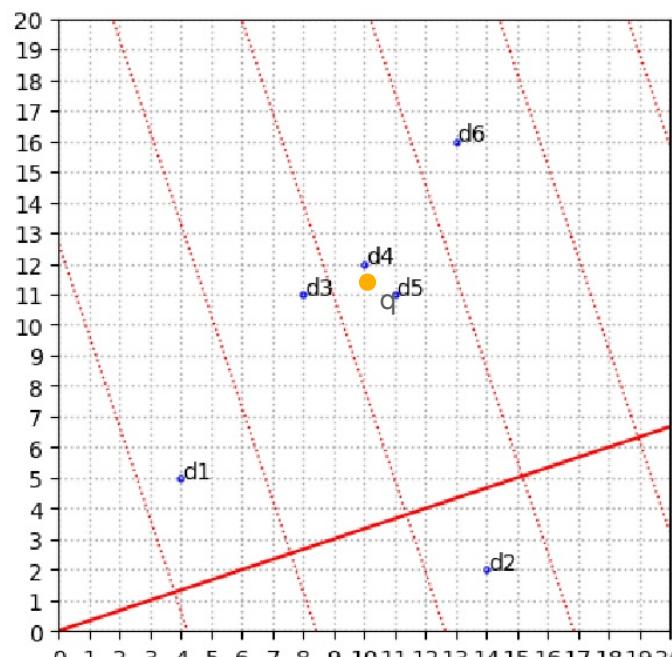
- Par table
 - Echantillonage de m fonctions de hachage

$$H^j = \{h_1(\cdot), h_2(\cdot), \dots, h_m(\cdot)\}$$

$$h_i^j(x) = \left\lfloor \frac{\langle x, a \rangle + b_i}{W} \right\rfloor$$

- Concaténation des fonctions h_i^j

$$g^j(x) = (h_1^j(\cdot), h_2^j(\cdot), \dots, h_m^j(\cdot))$$



Construction d'index avec plusieurs tables

Algorithm 7: Structures for LSH index (v1.0).

```

1 structure LSHTable
2   |   a: array of  $m$  vectors in  $\mathbb{R}^d$ 
3   |   b: vector in  $\mathbb{R}^m$            -  $m$  droites par table
4   |   table: Array of lists
5 end
6 structure LSHIndex
7   |   W: Float                  Même W pour toutes tables
8   |   tables: Array of  $l$  LSHTable    Tableau de tables
9 end

```

Algorithm 8: LSH index construction (v1.0).

```

1 function build_index( $\mathcal{D}: \mathbb{R}^{n \times d}$ ,  $W: \text{Float}$ ): LSHIndex
2   |   index: LSHIndex
3   |   index.W  $\leftarrow W$ 
4   |   for  $i$  from 1 to  $l$  do
5     |   |   index.tables[i]  $\leftarrow$  build_table( $\mathcal{D}$ ,  $W$ )
6   |   end
7   |   return index
8 end

```

Algorithm 9: LSH table construction (v1.0).

```

1 function build_table( $\mathcal{D}: \mathbb{R}^{n \times d}$ ,  $W: \text{Float}$ ): LSHTable
2   |   table: LSHTable
3   |   g: Vector in  $\mathbb{N}^m$ 
4   |   for  $i$  from 1 to  $m$  do
5     |   |   for  $j$  from 1 to  $d$  do
6       |   |   |   table.a[i][j]  $\sim \mathcal{N}(0, 1)$ 
7     |   |   end
8     |   |   table.a[i]  $\leftarrow$  L2_normalize(table.a[i])
9     |   |   table.b[i]  $\sim \mathcal{U}(0, W)$ 
10    |   end
11    |   for  $l \in \text{table.table}$  do
12      |   |   l  $\leftarrow \emptyset$ 
13    |   end
14    |   for  $x \in \mathcal{D}$  do
15      |   |   for  $i$  from 1 to  $m$  do
16        |   |   |   g[i]  $\leftarrow \left\lfloor \frac{\langle x, \text{table.a}[i] \rangle + \text{table.b}[i]}{\text{table.W}} \right\rfloor$ 
17      |   |   end
18      |   |   table.table[g].append(x)
19    |   end
20    |   return table
21 end

```

Recherche d'index avec plusieurs tables

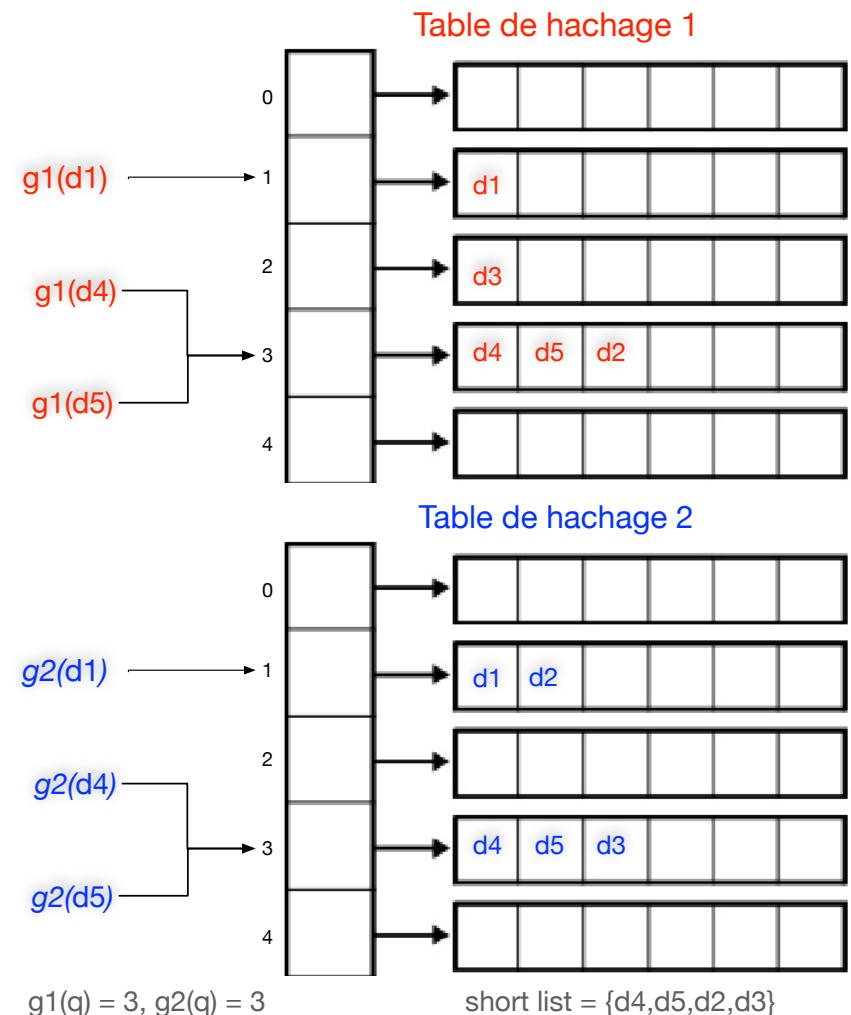
- Etapes
 - Calcul de l'indice de hachage pour q dans chaque table
 - Récupération des candidats associés de chaque table
 - Combinaison par union des candidats
 - Force brute appliquée au résultat de l'union

Algorithm 10: LSH index search (v1.0).

search_table() is the previous index search function (with additional parameter W).

```

1 function search_index(index: LSHIndex, q:  $\mathbb{R}^d$ ): Set of vectors
2   short_list: Set of vectors in  $\mathbb{R}^d$ 
3   for i from 1 to l do           Parcours des tables
4     | short_list  $\leftarrow$  short_list  $\cup$  search_table(index.tables[i], q, index.W)
5   end
6   return results
7 end
```



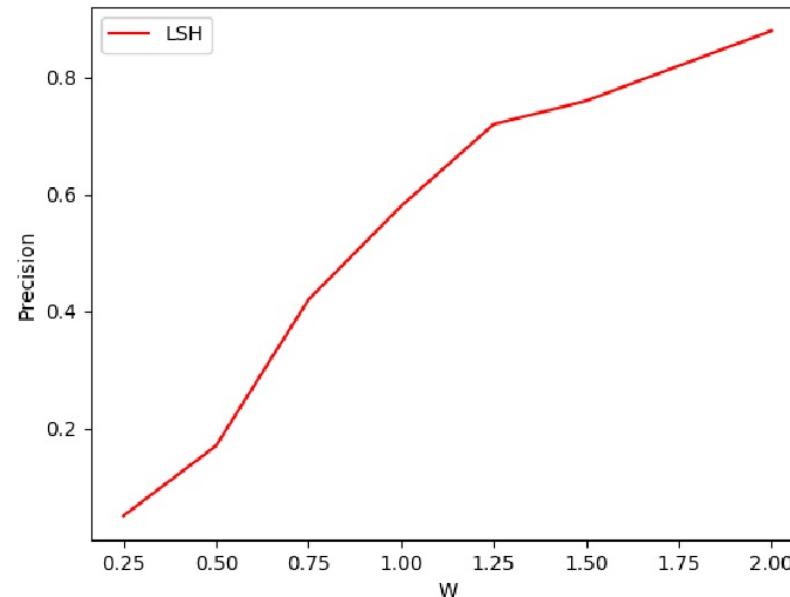
Complexité de LSH

- Etant donné les deux paramètres W (largeur des bacs) et L (nombre de tables)
 - Performances garanties théoriquement
 - Temps de construction : $\Theta(nLWt)$, t étant le temps d'évaluation d'une fonction de hachage sur un point
 - Espace : $\Theta(nL)$ plus l'espace pour stocker les points
 - Temps de recherche : $\Theta(L(Wt + dnP_2^k))$
 - Pour un facteur d'approximation donné $c = 1 + \epsilon$ et les probabilités P_1 et P_2 , on peut déterminer les paramètres
$$W = \left\lceil \frac{\log n}{\log 1/P_2} \right\rceil \text{ et } L = \left\lceil P_1^{-k} \right\rceil$$
 - Temps de construction : $\Theta(n^{1+\rho}P_1^{-1}kt)$, t étant le temps d'évaluation d'une fonction de hachage sur un point
 - Espace : $\Theta(n^{1+\rho}P_1^{-1})$ plus l'espace pour stocker les points
 - Temps de recherche : $\Theta(n^\rho P_1^{-1}(kt + d))$
 - avec $\rho = \frac{\log P_1}{\log P_2}$

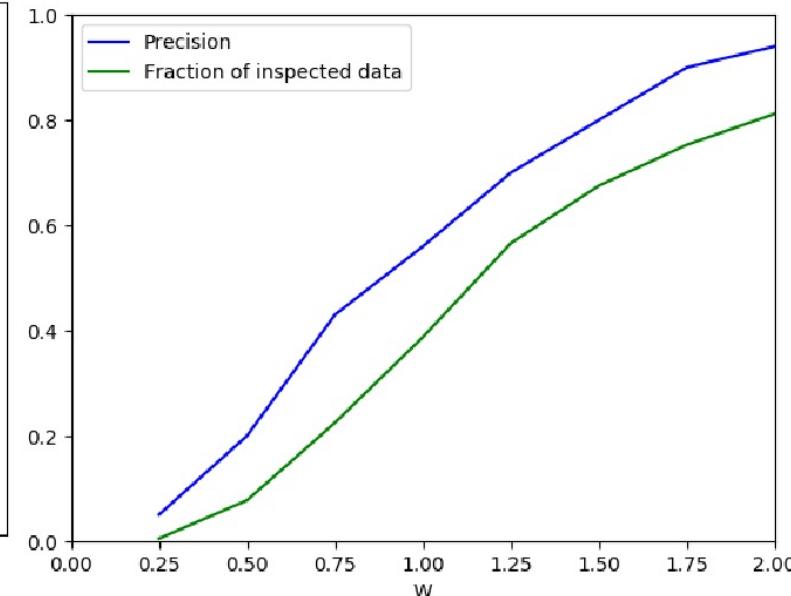
Performances de LSH

- Evaluation en fonction W (largeur des bacs)
 - Bac plus grand \Rightarrow Augmentation du Rappel
 - Rappel plus élevé \Rightarrow Plus de données inspectées
 - Plus de données inspectées \Rightarrow Ralentissement de la recherche

$n = 1.2 \times 10^6, |\mathbf{Q}| = 100, 2$ tables



$d = 25, m = 4$

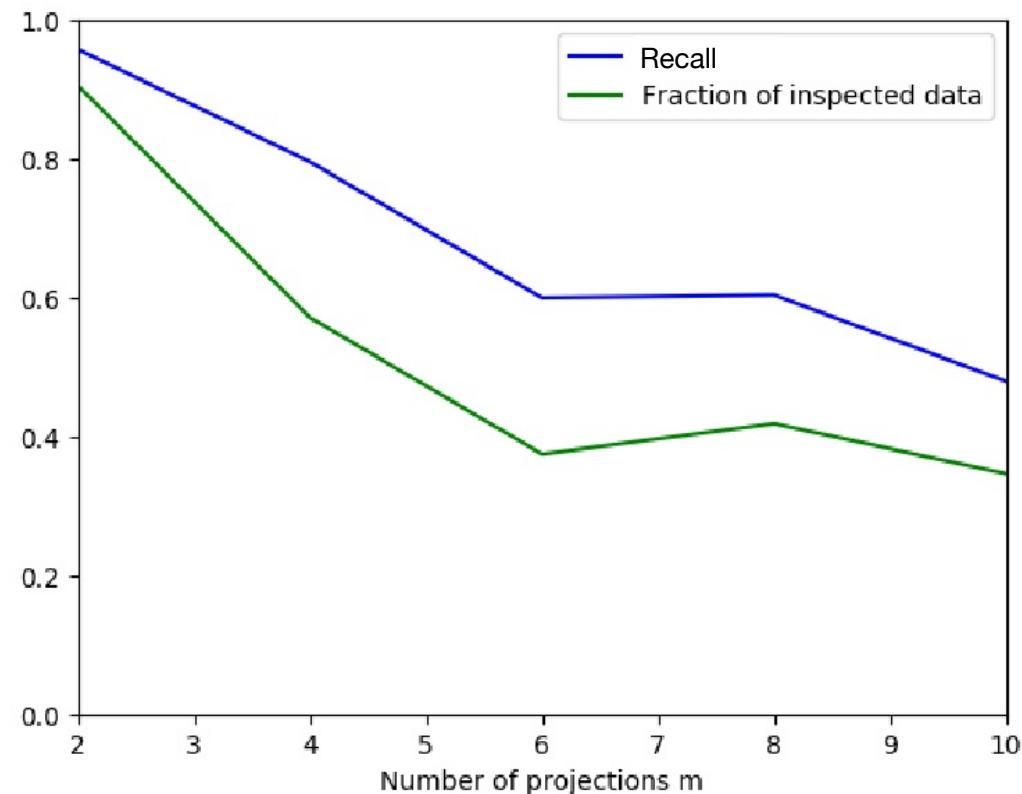


$d = 100, m = 10$

Performances de LSH

- Évaluation selon le nombre de projection m par table
 - Plus de projections \Rightarrow Augmentation de la précision mais diminution de rappel
 - Précision plus élevée \Rightarrow Moins de données inspectées
 - Moins de données inspectées \Rightarrow Accélération de la recherche
- Utilisation conjointe avec méthodes augmentant le rappel
 - Multi-probe LSH (recherche aussi dans les bacs adjacents)

$$n = 1.2 \times 10^6, |\mathbf{Q}| = 100, 2 \text{ tables}, W = 1.0$$

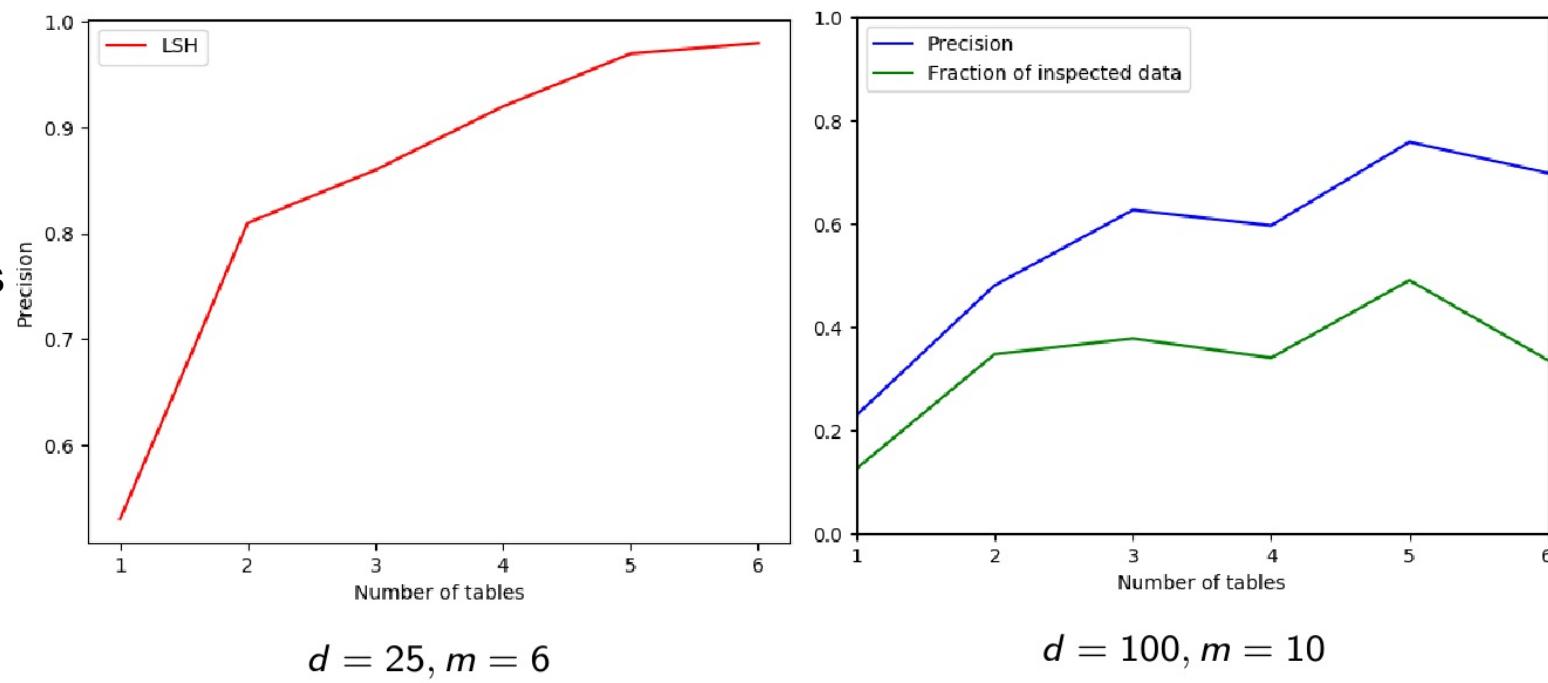


Performances de LSH

- Evaluation selon le nombre de tables

- Plus de table \Rightarrow Augmentation du Rappel
- Rappel plus élevé \Rightarrow Plus de données inspectées
- Plus de données inspectées \Rightarrow Ralentissement de la recherche

$$n = 1.2 \times 10^6, |\mathbf{Q}| = 100, W = 1.0$$



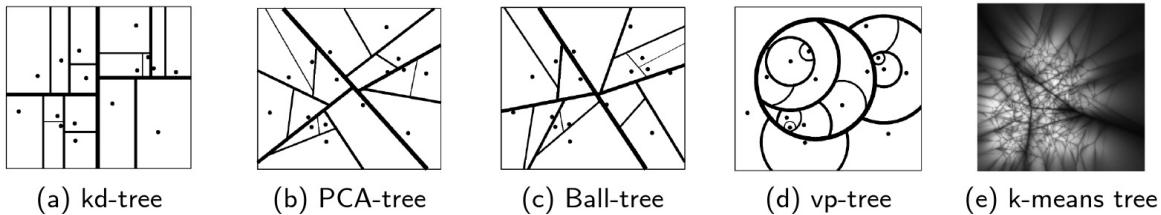
$$d = 25, m = 6$$

$$d = 100, m = 10$$

- Effet similaire à W

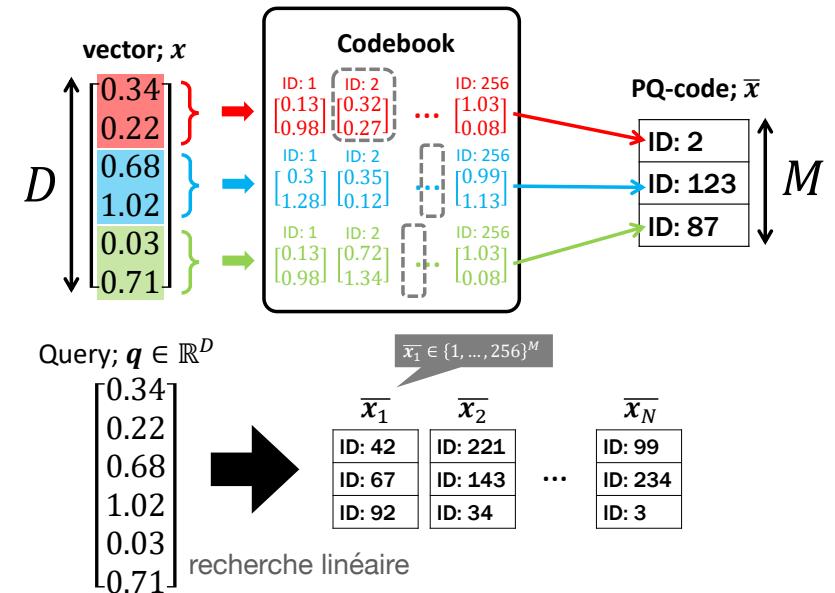
Autres méthodes

Méthode à base d'arbres

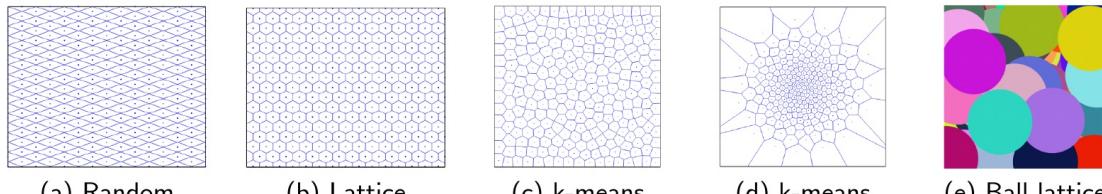


Kd tree (Sources: (a-d) [Kumar et al., 2008], (e) [Muja & Lowe, 2014])

Méthode à base de quantification de produits

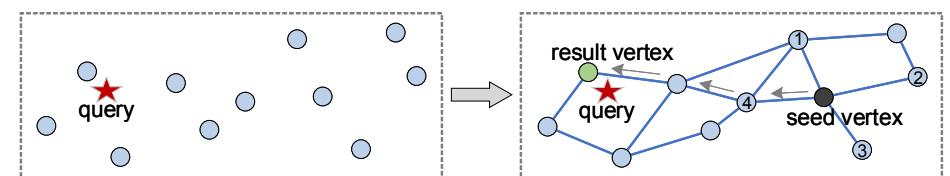


Méthode à base d'hachage



LSH (Source: (a-d) [Paulevé et al., 2010], (e) [Indyk, 2018])

Méthode à base de graphe de proximité



Conclusion

- Etude du problème générale de la recherche des plus proches voisins
- Nombreuses applications: recherche par similarité, recommandations, ...
- Notion de distances
- Algorithmes de base: Force brute, Arbres KD, Arbres VPT, LSH
- Evaluation des algorithmes, problèmes des grandes dimensions, compromis précision/vitesse
- Importance de l'expérimentation sur les données et les paramètres de l'indexation
- Champ de recherche actif