Recherche d'information par le contenu II

Master2 - Partage de données à grande échelle - GMIN307

Alexis JOLY - alexis.joly@inria.fr

November 1st 2012

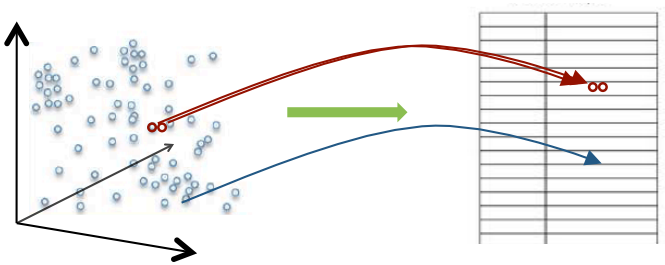
Etude détaillée de LSH

LSH: D

LSH (Localité Sensitive Hashing), méthode de hachage introduite par Gionis, Indyk et Motwani en 1998 pour la recherche approximative dans les espaces de grande dimensions

Sensible à la localité = la probabilité de collision des hash code est forte si les vecteurs sont proches et faible si les vecteurs sont éloignés

Formellement, selon Gionis et al. en 1998



$\mathcal{F}$ est une famille de fonctions $h : \mathbb{R}^d \rightarrow S$ satisfaisant les conditions suivantes pour deux points quelconques $p, q \in \mathbb{R}^d$ et une fonction h choisie aléatoirement parmi la famille $\mathcal{F}$:

- si $d(p,q) \leq R$, alors $Pr_{h \in H}[h(p) = h(q)] \geq P_1$
- si $d(p,q) \geq cR$, alors $Pr_{h \in H}[h(p) = h(q)] \leq P_2$

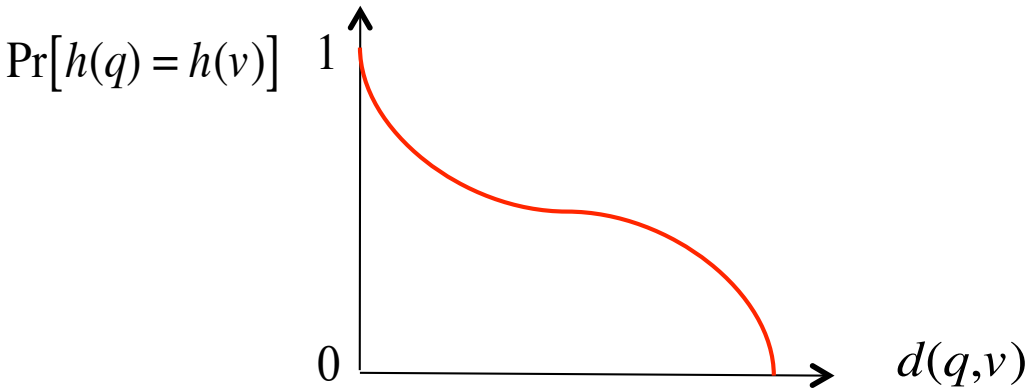
LSH: Intro

LSH (Localité Sensitive Hashing), méthode de hachage introduite par Gionis, Indyk et Motwani en 1998 pour la recherche approximative dans les espaces de grande dimensions

Pour certaines familles LSH, la probabilité de collision peut même s'exprimer comme une fonction décroissant avec la distance

$$\Pr\left[h_\theta(q) = h_\theta(v)\right]_{p_\theta} = f(d(q,v))$$

- θ is i.i.d drawn from a known **data independent distribution**
- f(d) is the sensitivity function (monotonically increasing from 0 to 1)



LSH sensible au produit scalaire

Exemple: famille LSH sensible au produit scalaire

$$h(x) = \mathrm{sgn}(w^T x) \qquad p_w = N(0,I)$$

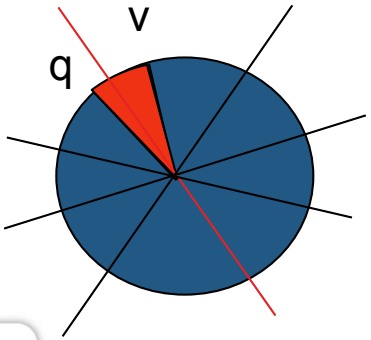
C'est le cas que l'on a déjà rencontré = projections aléatoires + quantification scalaire binaire

On peux montrer que la fonction de sensibilité vaut

Pour tous q, v $\in \mathbb{R}^d$

$$\Pr\left[h_w(q) = h_w(v)\right]_w = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{q^T v}{\|q\|\|v\|}\right)$$

Interprétation dans le cas normé:



$$\Pr\left[h_w(q) \neq h_w(v)\right]_w = angle(q,v)/\pi = \frac{1}{\pi}\cos^{-1}\left(q^T v\right)$$

$$= \frac{1}{\pi}\cos^{-1}\left(1 - \frac{\left(d(q,v)\right)^2}{2}\right)$$

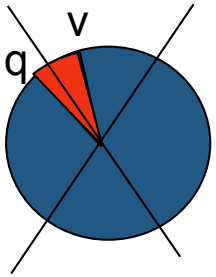
LSH sensible au produit scalaire

Exemple: famille LSH sensible au produit scalaire

$$h(x) = \mathrm{sgn}(w^T x) \qquad p_w = N(0,I) \qquad \Pr\left[h_w(q) = h_w(v)\right]_w = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{q^T v}{\|q\|\|v\|}\right)$$

On forme un hash code de D bits en utilisant D fonctions de hachage de la famille $\mathcal{F}$

$$z(x) = [h_1(x),...,h_D(x)] = sign(Wx)$$



z(q) = 1 hash code = [0 1 0 1 0 1 1 0 …0 0 1 0]

z(v) = 1 hash code = [1 0 1 1 1 1 0 1 …0 1 0 0]

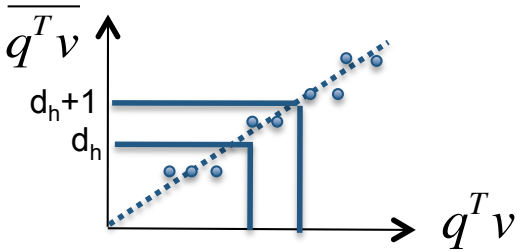
$$\approx \frac{D}{\pi}\cos^{-1}\left(\frac{q^T v}{\|q\|\|v\|}\right)$$

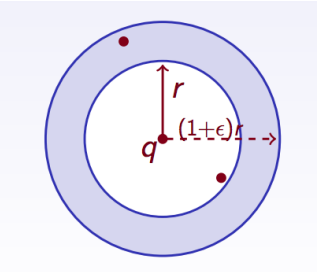
LSH sensible au produit scalaire

La distance de Hamming entre deux hash code est un estimateur de la probabilité de collision et donc du produit scalaire entre q et v

$$\overline{q^T v} = \|q\|\|v\|\cos\left(\frac{\pi}{D} d_h\left(z(q), z(v)\right)\right)$$

Variance





La variance $\sigma(k, q^T v)$ de l'estimateur décroit avec le nombre de bits et converge vers le produit scalaire exact

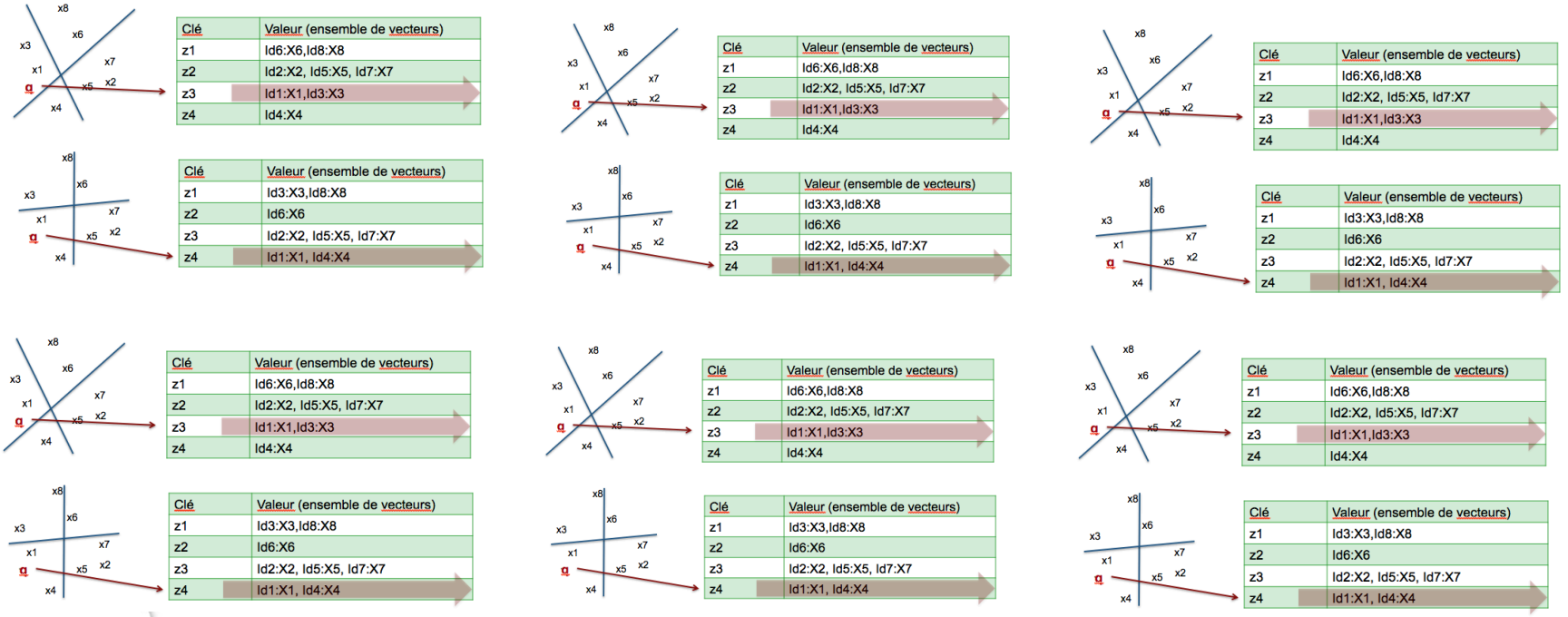
$$\lim_{k \to \infty} \overline{q^T v} = q^T v$$

Les k plus proches voisins dans l'espace de Hammoing convergent vers les k plus proches voisins exactes

LSH: algorithme de base pour l'indexation et la recherche

Tables multiples: création de L tables dont les clé sont générées par k fonctions de hachage (hash code de taille k pour chaque table). Taille index O(Ln)

Recherche = accès simple dans chaque table O(L)



LSH: algorithme de base

Probabilité de collision dans une table de k bits:

$$\Pr\left[h_w(q) = h_w(v)\right]_w = f(d(q,v)) \Rightarrow \Pr\left[z(q) = z(v)\right]_W = \left(f(d(q,v))\right)^k$$

Fonction de sensibilité

Très faible

Probabilité de non collision dans une table:

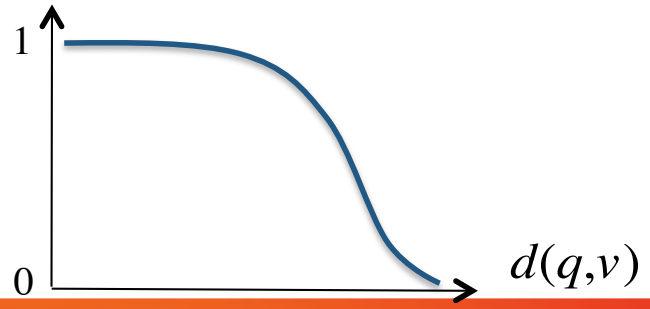
$$\Pr\left[z(q) \neq z(v)\right]_W = 1 - \left(f(d(q,v))\right)^k$$

Probabilité de non collision dans L tables:

$$\left(1 - \left(f(d(q,v))\right)^k\right)^L$$

Probabilité de collision dans au moins une table:

$$1 - \left(1 - \left(f(d(q,v))\right)^k\right)^L$$

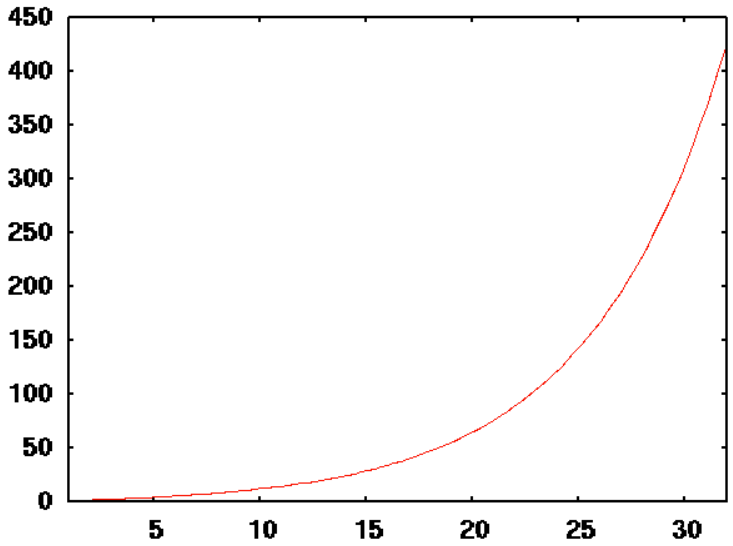


LSH: algorithme de base

Si on veux retrouver α (e.g 95%) des v tels que d(q,v)<r, il faut que la probabilité de collision dans au moins une table soit supérieure à α

$$1-\left(1-\left(f(r)\right)^{k}\right)^{L} > \alpha \quad \Rightarrow \quad L > \frac{\ln(1-\alpha)}{\ln(1-f(r)^{k})}$$

L (nombre de tables)



A qualité α constante et rayon r constant, le nombre de tables doit croitre pour compenser l'augmentation de k

k (nombre de bits par table)

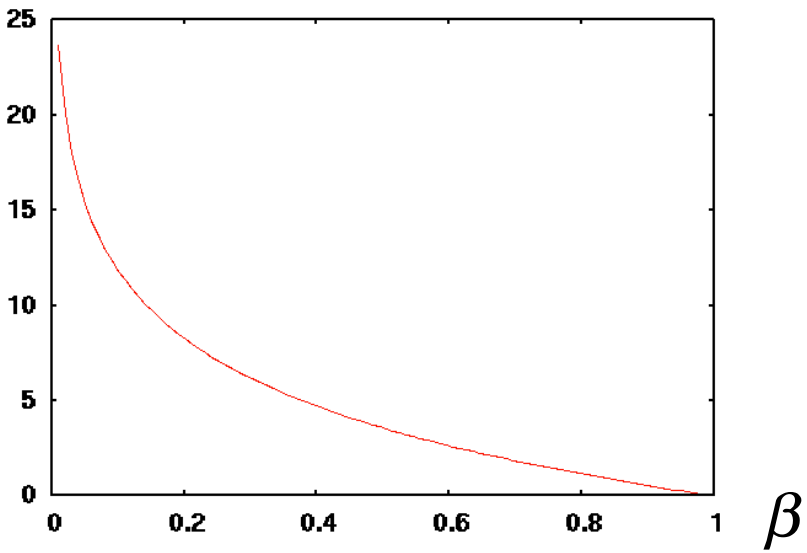
LSH: algorithme de base

D'un autre côté, l'augmentation de k permet de réduire la probabilté de fausses collisions dans chaque table. Si on veux que la probabilité de collision soit faible lorsque d(q,v)>r+ε

Proba de collision dans une table

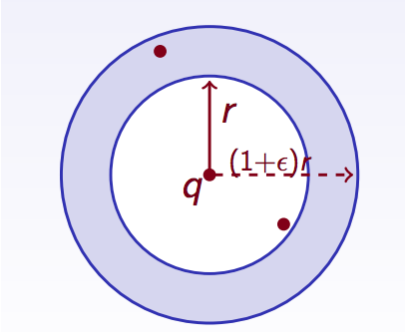
$$n\left(f((1+\varepsilon)r)\right)^k < \beta \quad \Rightarrow \quad k > \frac{\ln(\beta)}{\ln(f((1+\varepsilon)r))}$$

k (nombre de bits par table)



Propriété fondamentale de LSH dans le cas général

Requête à un rayon près (1+ε)-approximatives



- Given: radius R, error parameter ϵ and query point $\boldsymbol{q}$:
  - if there exists data point $\boldsymbol{p}$ s.t. $\boldsymbol{q} \in B(\boldsymbol{p}, R)$, return YES and a point (or all points) p' s.t. $\boldsymbol{q} \in B(\boldsymbol{p'}, (1+\epsilon)R)$,
  - if $\boldsymbol{q} \notin B(\boldsymbol{p}, R)$ for all data points p, return NO,
  - if closest data point to q is at distance between R and $R(1+\epsilon)$ then return YES or NO

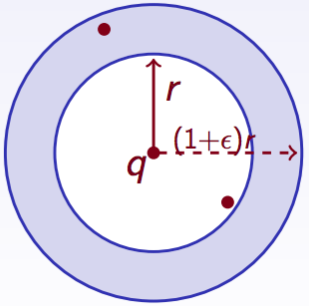
Propriété fondamentale de LSH dans le cas général

Famille de fonctions de hachage locality-sensitive
$\mathcal{H}$ avec paramètres (ϵ, r, P_1, P_2):

- Si $\|p - q\| \leq r$ alors $\mathcal{P}r_{\mathcal{H}}[h(p) = h(q)] \geq P_1$
- Si $\|p - q\| \geq (1 + \epsilon)r$ alors $\mathcal{P}r_{\mathcal{H}}[h(p) = h(q)] \leq P_2$

Theorem

Toute famille de fonctions de hachage (ϵ, r, P_1, P_2)-locality-sensitive conduit à un algorithme pour la recherche à un rayon r près $(1 + \epsilon)$-approximative avec $\sim n^{\rho}$ de temps de recherche et $n^{1+\rho}$ de taille d'index



$$\rho = \frac{\log(1/P_1)}{\log(1/P_2)} < 1$$

Propriété fondamentale de LSH dans le cas général

Pour avoir une probabilité d'erreur au plus égale à δ il faut prendre k, L tels que

$$P_2^k n \approx 1 \qquad\qquad L \approx (1/P_1)^k \log(1/\delta)$$

Avec ces contraintes on obtient en effet:

$$k = \frac{\log n}{\log(1/P_2)}$$

$$L = (1/P_1)^{\frac{\log n}{\log(1/P_2)}} \log(1/\delta) = n^{\frac{\log(1/P_1)}{\log(1/P_2)}} \log(1/\delta) = n^{\rho} \log(1/\delta)$$

Propriété fondamentale de LSH dans le cas général

L'espérance du nombre d'objets rencontrés à moins de $(1+\epsilon)r$ de la requête est $P_2^k Ln \approx L$

Pour les vrais voisins à r-près la probabilité d'être hachés dans une même case que q est au moins

$$1-(1-(1/P_1)^k)^L \geq 1-(1/e)^{\frac{L}{(1/P_1)^k}} \geq 1-\delta$$

Taille de l'index $\mathcal{O}(Ln) \approx n^{1+\rho+o(1)}$

Recherche $\mathcal{O}(L) \approx n^{\rho+o(1)}$

LSH sensible aux distances L_p

p–Stable distributions

- **p–stable distribution ($p \geq 0$)**: A distribution $\mathcal{D}$ over $\Re$ s.t
  - n real numbers $v_1 \ldots v_n$,
  - i.i.d. variables $X_1 \ldots X_n$ with distribution $\mathcal{D}$,
  - r.v. $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{1/p} X = l_p(\boldsymbol{v})X$, where X is a r.v. with distribution $\mathcal{D}$

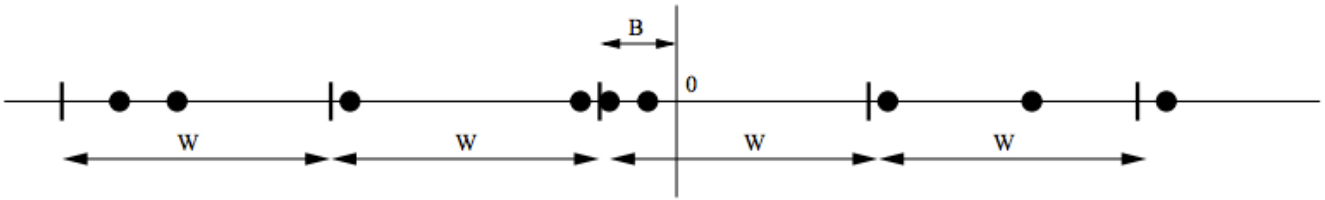
- E.g. p–Stable distr for $p = 1$ is Cauchy distr, for $p = 2$ is Gaussian distr

LSH sensible aux distances L_p

How are p–Stable distributions useful?

- Consider a vector $\boldsymbol{X} = \{X_1, X_2, \ldots, X_N\}$, where each X_i is drawn from a p–Stable distr
- For any pair of vectors $\boldsymbol{a}, \boldsymbol{b}$ $\boldsymbol{a} \cdot \boldsymbol{X} - \boldsymbol{b} \cdot \boldsymbol{X} = (\boldsymbol{a} - \boldsymbol{b}) \cdot \boldsymbol{X}$ (by linearity)
- Thus $\boldsymbol{a} \cdot \boldsymbol{X} - \boldsymbol{b} \cdot \boldsymbol{X}$ is distributed as $(l_p(\boldsymbol{a} - \boldsymbol{b}))X'$ where X' is a p–Stable distr r.v.
- Using multiple independent $\boldsymbol{X}$'s we can use $\boldsymbol{a} \cdot \boldsymbol{X} - \boldsymbol{b} \cdot \boldsymbol{X}$ to estimate $l_p(a - b)$ [Ind'01]

LSH sensible aux distances L_p



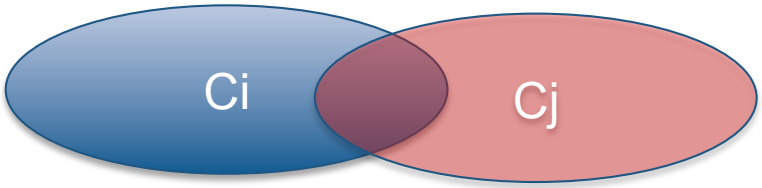
- Consider $h_{\boldsymbol{a},b} \in \mathcal{H}^w$, $h_{\boldsymbol{a},b}(\boldsymbol{v}) : \mathcal{R}^d \rightarrow \mathcal{N}$
- $\boldsymbol{a}$ is a d dimensional random vector whose each entry is drawn from a p-stable distr
- b is a random real number chosen uniformly from $[0, w]$ (random shift)
- $h_{\boldsymbol{a},b}(\boldsymbol{v}) = \lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{v} + b}{w} \rfloor$

LSH sensible à la distance de Jaccard

MinHash: the min-wise independent permutations locality sensitive hashing scheme

Jaccard distance = mesure de similarité entre des ensembles d'objets

$$sim_J(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$$



- View sets as columns of a matrix; one row for each element in the universe. a_{ij} = 1 indicates presence of item i in set j
- Example

C_1	C_2
0	1
1	0
1	1
0	0
1	1
0	1

$sim_J(C_1, C_2) = 2/5 = 0.4$

Hash function=

- Randomly permute rows
- Hash $h(C_i)$ = index of first row with 1 in column C_i
- Suprising Property

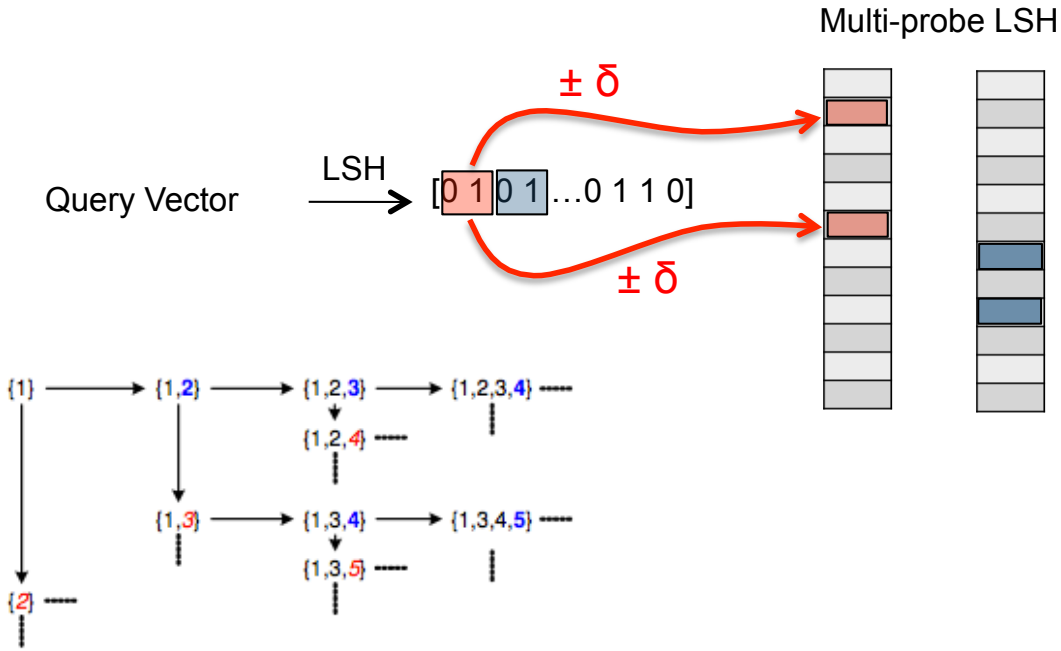
$$P[h(C_i) = h(C_j)] = sim_J(C_i, C_j)$$

Limitations de LSH et variantes

Problème de LSH: espace mémoire

LSH peux nécessiter un grand nombre de tables pour atteindre des qualités suffisantes ce qui rend la méthode problématique lorsque les données sont grandes

→ Utilisation d'accès multiples pour réduire le nombre de tables



Exemple: LSH pour vecteurs binaires de grande dimension

Vecteurs d'entrée déjà binarisés (souvent le cas en audio fingerprinting)

- Cube de Hamming: $\{0,1\}^d$ avec distance de Hamming

Fonction de hachage = une réduction de $\{0,1\}^d$ dans $\{0,1\}^k$. Le choix des composantes se fait selon une loi binomiale:

- Choisir aléatoirement un ensemble de composantes selon une loi binomiale de paramètre $\frac{1}{2l}$, (espérance du nombre de composantes $\frac{d}{2l}$)
- Affecter aléatoirement 0 ou 1 à toutes ces positions, affecter les autres à 0. Soit r le vecteur résultat.
- $h_r(p) = r \cdot p$

= Projection dans des sous espaces aléatoires de dimension avec k<<d typiquement:

k=log(d)

Exemple: LSH pour vecteurs binaires de grande dimension

Recherche à (δ_1,δ_2)-près = conserver les vecteurs à moins de d.δ_1 bits de la requête, rejeter ceux à plus de d.δ_2 bits de la requête

Indexation/pré-traitement:

- Projeter tous les p dans $A(p) = h_{r_1}(p) \dots h_{r_k}(p)$
- Pour tout $v \in \{0,1\}^k$ précalculer tous les $(\frac{\delta_1+\delta_2}{2})k$-voisins

= Buckets voisines à un rayon près

Recherche:

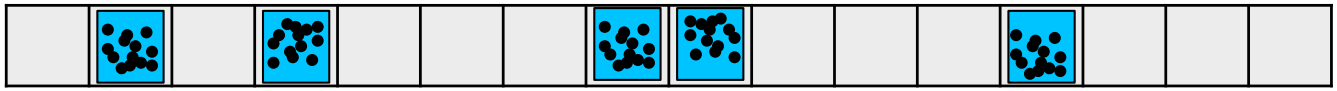
- Calculer $A(q) = h_{r_1}(q) \dots h_{r_k}(q)$
- Retrouver et vérifier explicitement les $(\frac{\delta_1+\delta_2}{2})k$-voisins de $A(q)$

= Visite des buckets voisines de la requête dans chaque table.

- Temps de recherche logarithmique, taille de l'index polynomiale

Autre problème de LSH: mauvais balancement

Sur certaines données réelles, LSH peux conduire à des partitions loin de l'uniformité. Certaines bucket peuvent concentrer presque tous les points:



→ Augmentation du nombre de fausses collisions
→ Augmentation du temps de raffinement
→ Problématique dans les contexte distribués (load balancing)

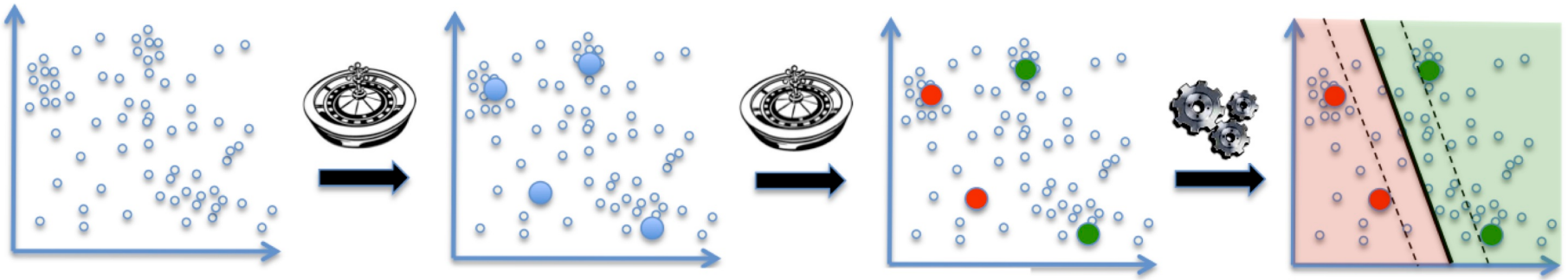
Solution: utiliser des fonctions de hâchage dépendant des données. De nombreuses fonctions « data dependent » ont été proposées récemment dans la littérature (spectral-hashing, KLSH, RMMH, etc.)

Random Maximum Margin Hashing

RMMH = une famille de fonctions de hachage basée sur le **partitionnement aléatoire des données**

Avantage = fonctions indépendantes + adaptabilité aux données

Basé sur **un concept théorique nouveau** = **l'apprentissage aléatoire de classifieurs** (1 par fonction de hachage = 1 par bit)



M échantillons aléatoires

Labels positifs et négatifs purement aléatoires !!

SVM

$$h(\mathbf{x}) = sgn(\mathbf{w}_m.\mathbf{x} + b_m)$$

Random Maximum Margin Hashing

Pourquoi ça marche ?

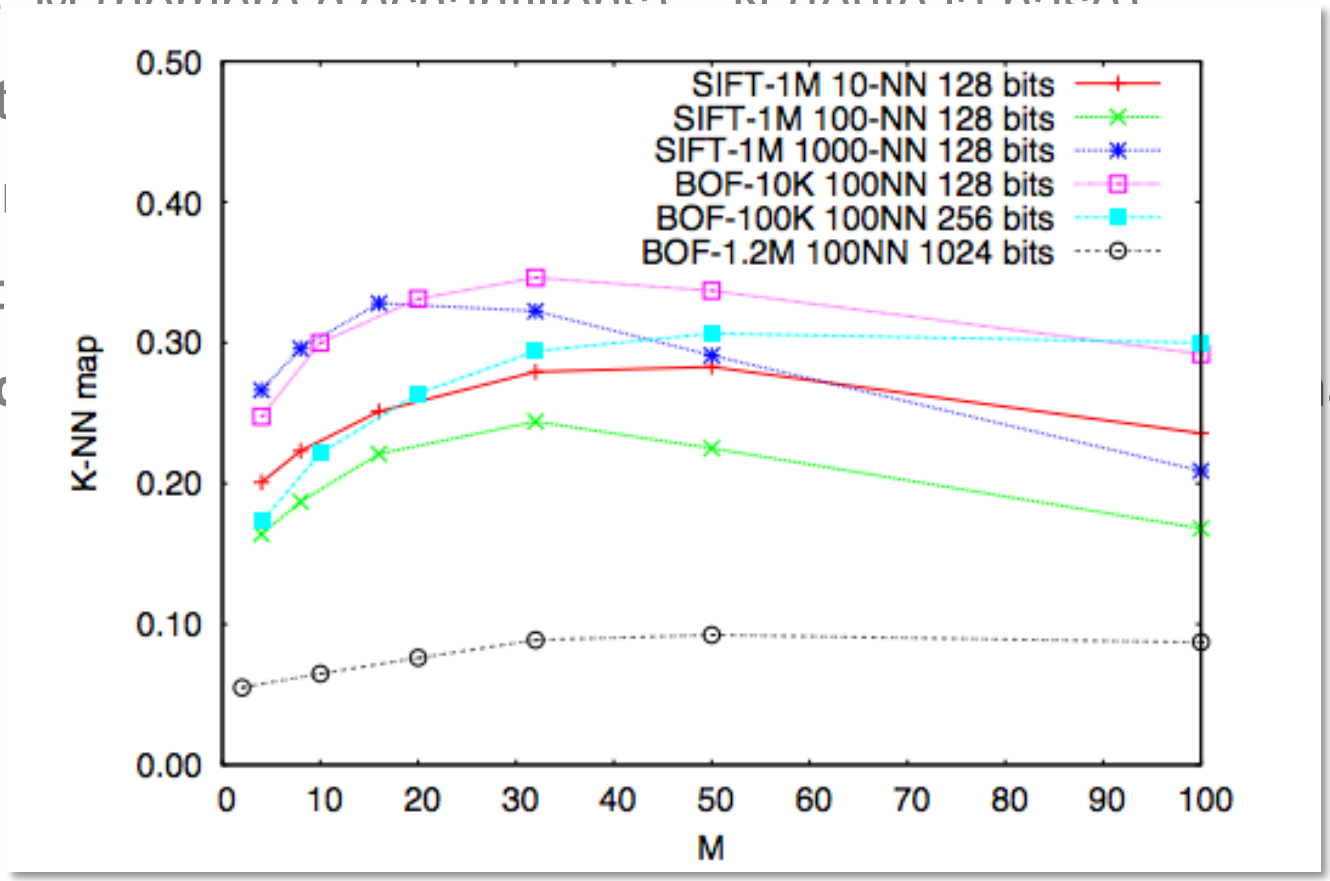
Cas extrême: M (nombre d'échantillons) = N (toute la base)

Uniformit

Overfitti

M trop faible:

M optimal = **c**

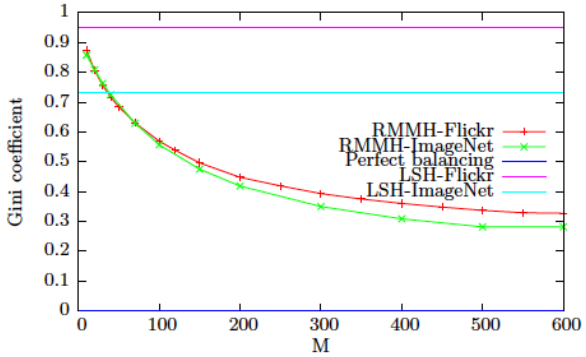


Random Maximum Margin Hashing

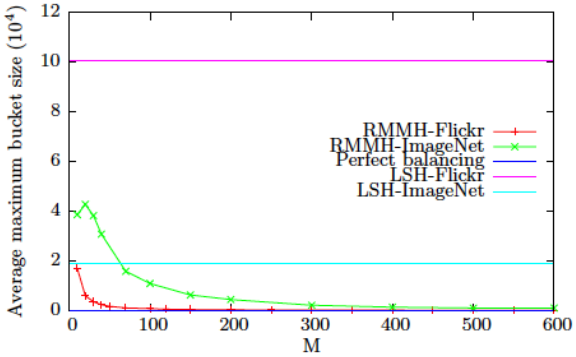
Pourquoi ça marche ?

Cas extrême: M (nombre d'échantillons) = N (toute la base)

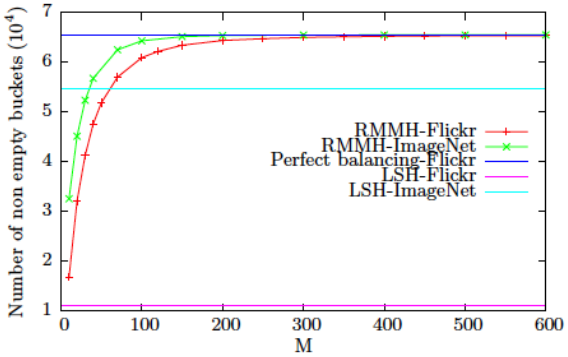
Uniformité parfaite de la partition apprise (≈Voronoï)



(a) Gini coefficient



(b) Avg. Max. bucket size



(c) Number of non empty buckets

Random Maximum Margin Hashing

Evaluation on ImageNet

1.2M images described by BovW featrues (dim 1000)

Supervised classification across 1000 semantic categories, 1000-NN