Locality-Sensitive Hashing

Francesco Corsi

Dipartimento degli studi Roma 3

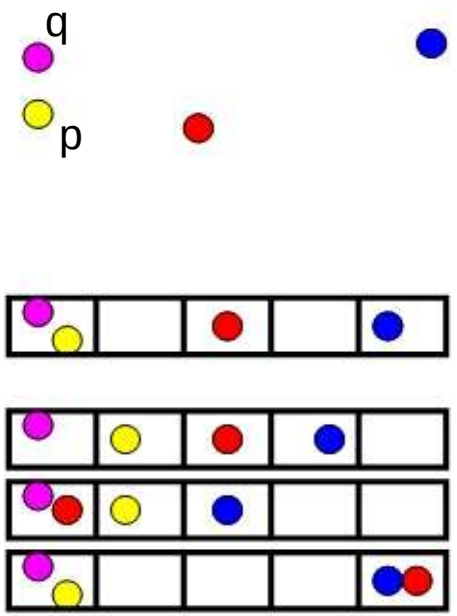
Facoltà di matematica

Indice

1. Obbiettivo
2. Locality-Sensitive Hashing - LSH
3. Ideal & actual LSH Family
4. Algoritmo
5. Il valore ρ
6. Applicazione della famiglia LSH
   (il problema del nearest neighbor)

Obbiettivo

L'obbiettivo dello studio è quello di costruire una funzione hash $g: R^d \rightarrow U$, dove R^d è uno spazio di dimensione d e U è uno spazio di dimensione fissata, tale che dati i punti p,q:



- Se $||p-q|| \leq r$,
  allora $Pr[g(p)=g(q)]$ è alta

- Se $||p-q||>cr$,
  allora $Pr[g(p)=g(q)]$ è bassa

Locality-Sensitive Hashing - LSH

- Una famiglia H di funzioni $h: R^d \rightarrow U$ è detta (P_1,P_2,r,cr)-sensibile, se dati i punti p,q:

  - Se $||p-q|| < r$ allora $Pr[\ h(p)=h(q)\ ] > P_1$
  - Se $||p-q|| > cr$ allora $Pr[\ h(p)=h(q)\ ] < P_2$

Famiglia LSH per λ_2

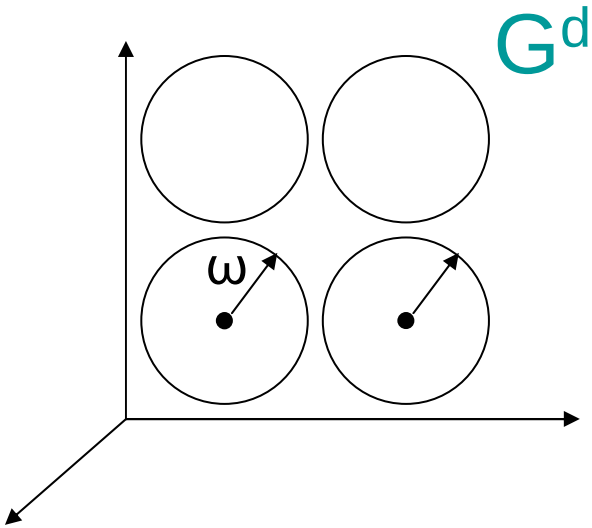
- Sarà descritta una famiglia “ideale” LSH per λ_2. Questo approccio ha alcune carenze e si mostrerà in quale modo possono essere migliorate.
- Sarà analizzato in che modo ottenere una buona famiglia di funzioni LSH, eliminando i problemi della famiglia ideale.
- Sarà descritto un algoritmo per costruire una famiglia di funzioni LSH ottimale.

- con λ_2^d indichiamo lo spazio R^d dotato di norma λ_2 : $||x_1,\ldots,x_d||_2=(|x_1|^2+\ldots+|x_d|^2)^{1/2}$

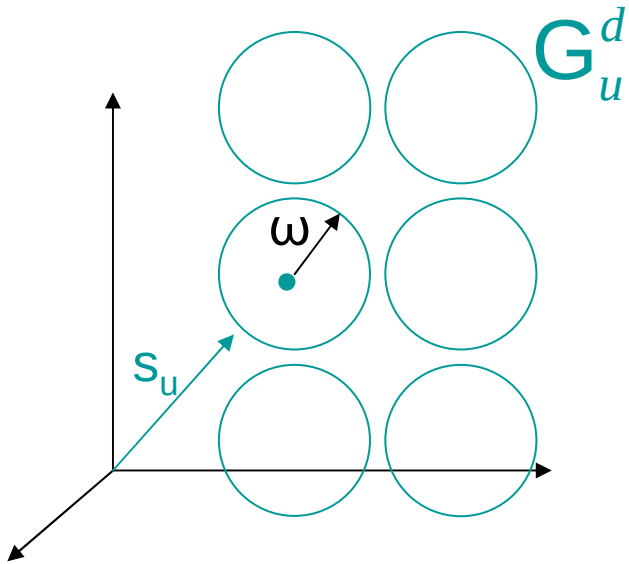
Ideal LSH Family

Una funzione hash $\hat{h}$ è costruita come segue:

- Sia G^d una griglia regolare infinita di palle in R^d:
  ogni palla ha raggio ω e ha centro in $4\omega \cdot Z^d$

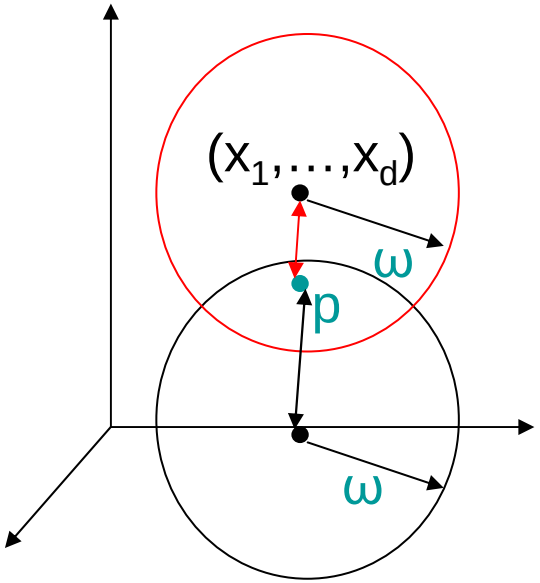


- Sia G_u^d per interi positivi u, la griglia G^d spostata in modo random uniformemente;
  ossia $G_u^d = G^d + S_u$ dove $S_u \in [0,4\omega]^d$.



…Ideal LSH Family

Occorre considerare quanti G_u^d sono necessari per ricoprire l'intero spazio R^d.
Si supponga che ne servano U per ricoprire l'intero spazio con una alta probabilità.



Si definisce $\hat{h}$ su un punto p come una d+1-upla: $(u,x_1,\ldots,x_d)$, con $u \in [1,U]$ ed $x_1,\ldots,x_d \in G_u^d$

La d+1-upla $(u,x_1,\ldots,x_d)$ specifica la palla che contiene il punto p: $p \in B((x_1,\ldots,x_d),\omega)$.
Se ci sono più palle che contengono p si sceglie quella con il più piccolo valore u.

…Ideal LSH Family

Il tempo necessario per calcolare $\hat{h}(p)$ è $\tau = O(U)$

Si itera su $G_1^d, \ldots, G_U^d$ e si trova il primo G_u^d tale che p è all'interno della palla con il centro in G_u^d.

…Ideal LSH Family

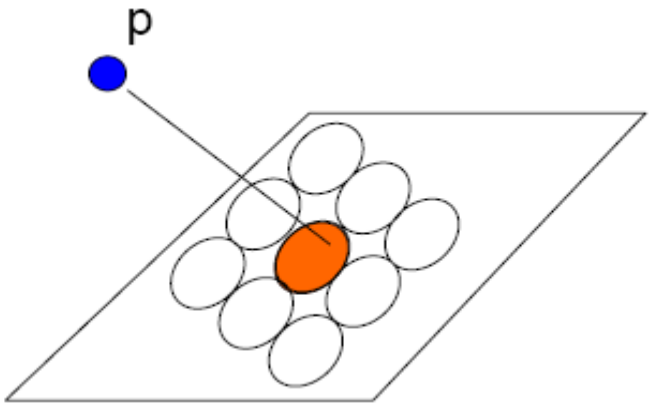
Intuitivamente, questa famiglia soddisfa la definizione di locality-sensitive data, in quanto più vicini sono p,q e più grande è la probabilità che p,q appartengono alla stessa palla.

Se $U = \Omega(2^d)$ allora $d=\Omega(\log n)$, quindi il tempo per calcolare $\hat{h}(p)$ può essere troppo grande.

Actual LSH Family

La famiglia che viene attualmente utilizzata è quella ideale descritta precedentemente a cui si aggiunge un ulteriore passo che premette di ridurre **U**, ossia il numero di griglie G_u^d per ricoprire lo spazio.

…Actual LSH Family



Per ridurre **U**, si fa una proiezione:
φ: $R^d \rightarrow R^t$ dove:

R^t è lo spazio di

dimensione **t < d**, ottenuto

tramite una riduzione dimensionale random.

Esempio:

si consideri una funzione f**:$R^3 \rightarrow R^2$,**

definita nel seguente modo $f(x,y,z)=(x+y+z, x-2y+3z)$

determinata dalla matrice $A= \begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & 3 \end{pmatrix}$

…Actual LSH Family

Il parametro **t** è **O(log n)**, così che i fattori esponenziali in **t** sono dell'ordine di **O(n)**.

Dopo la proiezione, si scelgono le griglie $G_1^t, \ldots, G_U^t$ nello spazio di dimensione minore $\mathbf{R^t}$.

Per calcolare **h(p)**, si analizza la proiezione di **p** nello spazio $\mathbf{R^t}$.

…Actual LSH Family

La funzione hash attuale è

$h(p) = \hat{h}(Ap)$ dove:

- **A** è una matrice random che rappresenta la mappa che riduce la dimensione da **d** a **t.**

- **$\hat{h}$** lavora nello spazio dimensionale t.

Algoritmo

Inizializzazione della funzione hash $h \in H$:

1. Per $u=1,\ldots,U$, si sceglie un vettore random $s_u \in [0,4\omega]^t$, che specifica la griglia $G_u^t = G^t + s_u$ nello spazio euclideo di dimensione t.
2. Si sceglie una matrice $\mathbf{A} \in \mathbf{M}_{t,d}$ dove ogni elemento $\mathbf{A}_{ij}$ è distribuito in accordo alla distribuzione normale standard $N(0,1)$ volte un fattore scalare $\frac{1}{\sqrt{t}}$. La matrice $\mathbf{A}$ rappresenta una proiezione random da $\mathbf{R}^d$ a $\mathbf{R}^t$

…Algoritmo

Calcolare $h()$ su un punto $p \in R^{\mathbf{d}}$:

1. Sia $p' = \mathbf{A}p$ la proiezione del punto p nel sottospazio di dimensione t dato da $\mathbf{A}$.
2. Per ogni $u=1,2,\ldots,U$:
   1. Si deve verificare se $B(p',\omega) \cap G_u^t \neq \emptyset$.
      i.e. se esiste qulache $(x_1,\ldots,x_t) \in G_u^t$ tale che $p \in B((x_1,\ldots,x_t),\omega)$.
   2. Una volta trovato tale $(x_1,\ldots,x_t)$, si definisce $h(p)= (u,x_1,\ldots,x_t)$.
3. Se non si trova alcuna palla così definita, si restituisce il valore 0^{t+1}.

Il valore ρ

Dato $c \geq 1$ e $p_2 \in (0,1)$ si definisce $\rho_x(c,p_2)$ la più piccola costante $\rho > 0$ tale che:

per ogni $r > 0$ esiste $p_1 \in (0,1)$ e una famiglia hash

(p_1,p_2,r,cr)-sensibile $H: X \to N$ con $\frac{\log(1/p_1)}{\log(1/p_2)} \leq \rho$, ossia:

$$\rho_x(c,p_2) = \sup_{r>0} \inf \left\{ \frac{\log(1/p_1)}{\log(1/p_2)} : \exists \text{ famiglia hash } (p_1,p_2,r,cr) \text{ - sensibile } H : X \to N \right\}$$

- Se $X = \lambda_s^d$ per qualche $s > 0$ e $d \in N$, definiamo:

$$\rho_s(c) = \sup_{0<q<1} \lim \sup_{d \to \infty} \rho_{\lambda_s^d}(c,q)$$

…Il valore ρ

Lemma:

Si consideri la funzione hash h, descritta nell'algoritmo, siano p,q due punti di R^d.

Sia p_1 la probabilità che h(p)=h(q) sapendo che ||p-q|| ≤1,

Sia p_2 la probabilità che h(p)=h(q) sapendo che ||p-q|| ≥c.

Allora per $\omega=O(\frac{1}{\sqrt[4]{t}})$ si ottiene:

$$\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = 1/c^2+O(\frac{\log(t)}{t^{1/2}}).$$

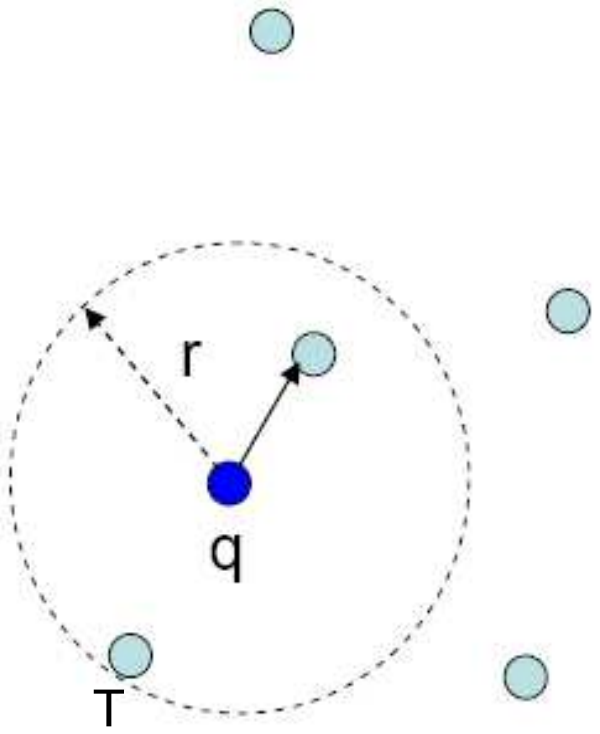
…Il valore ρ

L’importanza del parametro ρ deriva dall’applicazione delle funzioni LSH al problema di approssimazione della ricerca del nearest neighbor.

Nearest neighbor & R-near neighbor

- **Nearest neighbor:**

  Data un insieme P di N punti, in uno spazio di dimensione d, R^d, si costruisce una struttura di dati tale che, dato un query point q, restituisce il punto di P che è il più vicino a q.



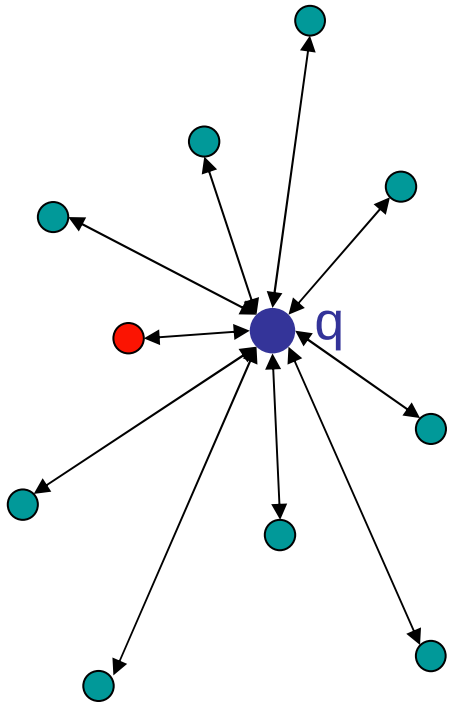
- **R-near neighbor:**

  Un punto T si dice un R-near neighbor di q se la distanza di T da q è al massimo R.

Algoritmo ingenuo

Dato un query point q, si calcola la distanza di q da ogni punto di P, e si riporta il punto con distanza minima.

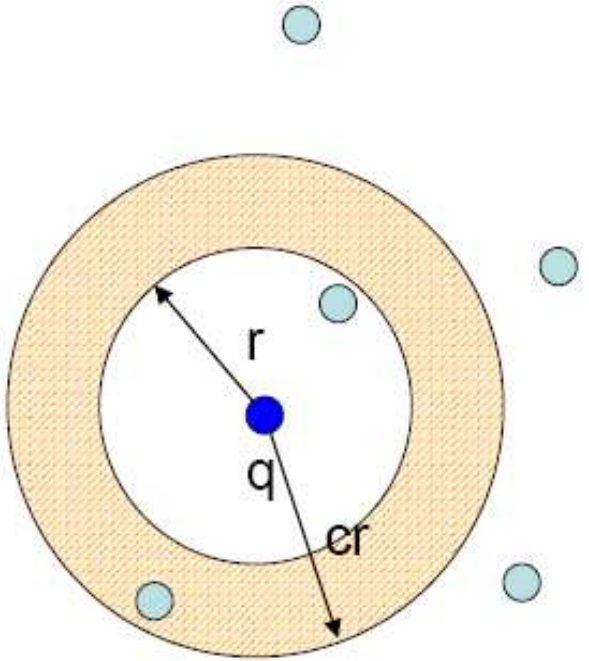
L'approccio dell'esplorazione lineare ha un query time dell'ordine O(dn), che è tollerabile per piccoli insiemi di dati, ma al loro crescere diventa inefficiente e dipende in modo esponenziale da d.



È possibile rimuovere la dipendenza esponenziale da d tramite l'approssimazione C-approximate near neighbor.

C-approximate near neighbor: c-NN

Dato un insieme P di punti in uno spazio di dimensione d, R^d, si costruisce una struttura di dati tale che dato un query point q, restituisce qualche punto con distanza al massimo c-volte la distanza di p da q, dove p è il punto di P più vicino a q.



Punto di vista random

- **Randomized C-approximate near neighbor:**

  Dato un insieme P di punti in uno spazio di dimensione d, R^d ed i parametri $R>0$, $\delta>0$, si costruisce una struttura di dati tale che:
  dato un query point q, se esiste un punto di P distante R da q, restituisce i punti di P distanti al massimo cR da q con probabilità $1-\delta$.

- **Randomized near-neighbor reporting:**

  Dato un insieme P di punti un uno spazio di dimensione d, R^d, ed i parametri $R>0$, $\delta>0$, si costruisce una struttura di dati tale che:
  dato un query point q, restituisce ciascun punto $p' \in P$ con distanza al massimo R da q con probabilità $1-\delta$.

nearest-neighbor & near-neighbor: collegamenti

Il problema di near-neighbor può essere visto come una versione di decisione del problema di nearest-neighbor:

Si crea una struttura di dati contenente varie R-near neighbor, per $R=R_1,\ldots,R_t$, dove R_t dovrebbe essere più grande della massima distanza di un query point rispetto al suo nearest neighbor.

Il nearest neighbor si recupera riordinando la struttura di dati rispetto all'ordine crescente del raggio, ci si ferma quando si trova il primo punto.

Conclusione

In conclusione si può dimostrare:

Teorema: Usando l'algoritmo descritto precedentemente è possibile risolvere il problema c-NN in λ_2^d che richiede un query time $O(dn^{\rho})$ e $O(dn^{1+\rho})$ per lo spazio e pre-elaborazione.

Dove il valore di ρ è dato dal lemma visto precedentemente:

$\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = 1/c^2 + O\left(\frac{\log(t)}{t^{1/2}}\right)$