Report LSH

Rocco Caruso

March 29, 2016

Chapter 1

Unspervised Learning

1.1 DBSCAN

L'idea "chiave"dell'algoritmo DBSCAN[2] è che ciascun elemento di un cluster debba avere un vicinato entro un certo raggio ϵ (Eps) che contenga almeno un numero minimo di punti (MinPts). Questo algoritmo, in altre parole, stima la *densità* di ciascun punto, per mezzo della cardinalità del suo "vicinato"entro un raggio predefinito Eps. Un punto sarà definito come *core-object* se la densità del suo vicinato supera una data soglia $MinPts$. Un punto q si dice "directly density-reachable"da un core-object p, se q ricade nell'eps-vicinato di p. La chiusura transitiva di questa proprietà, è detta density-reachability. Due punti p e q si diranno invece, *density-connected*, se esiste un terzo oggetto o dal quale entrambi sono density-reachable. Grazie a quest'ultima proprietà è possibile definire un cluster, ovvero come un insieme di oggetti densamente connessi che è massimale rispetto alla proprietà di density-reachability. Il noise, invece, sarà dato dall'insieme di punti che non appartengono a nessuno di questi cluster. In seguito verranno date le definizioni in maniera più formale:

definzione 1 (directly density-reachable)**.** Un oggetto p si dice *directly-density-reachable* da un oggetto q rispetto ai parametri Eps ed $MinPts$ nell'insieme di oggetti D se:

1. $p \in N_{eps(Q)}$
2. $|N_{eps(Q)}| \geq MinPts$

definzione 2 (density-reachable)**.** Un oggetto p si dice *"density-reachable"* da un oggetto q rispetto ai parametri Eps ed $MinPts$ nell'insieme di oggetti D, e si indica con $p >_D q$, se esiste una catena di oggetti: $p_1, \ldots, p_n$, $p_1 = pp_n = q$ tali che: $p_i \in D \wedge p_{i+1}$ è *directly-density-reachable* da p_i

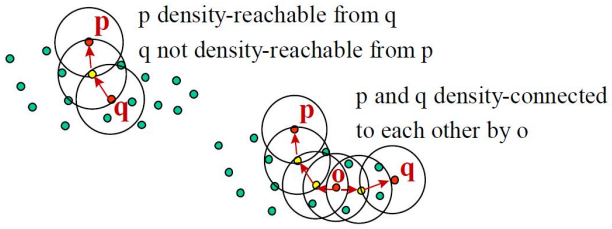


Figure 1.1: Density Reachability e Density Connectivity

Questa relazione è l'estensione canonica della relazione Directly-density-reachability, inoltre è una relaione transitiva ma non simmetrica. Sebbene non sia simmetrica, vi sono delle zone nell'isnieme D, in cui questa relazione è simmetrica ovvero per quegli oggetti o nell insieme per cui $|N_{eps(o)}| \geq MinPts$. Due oggetti *border*, ovvero due oggetti che giacciono sul confine di un cluser, probabilmente non saranno density-reachable fra loro in quanto non ci sono abbastanza oggetti nel loro vicinato. Tuttavia, ci sarà necessariamente, un terzo oggetto nel cluster dal quale entrambi questi oggetti border saranno density-reachable.

definzione 3 (density-connected)**.** Un oggetto p si dice *"density-connected"* da un oggetto q wrt Eps ed $MinPts$ nell'insieme di oggetti D se esiste un oggetto $o \in D$ tale che sia p che q siano *density reachable* da o. La relazione di density-connectivity è simmetrica, la figura 1.1 mostra la le definizioni su descritte in uno spazio bi-dimensionale.

Proprio grazie alla proprietà di density connectivity, si potrà dare la definizione di cluster ovvero un insieme massimale di oggetti "densamente connessi".

definzione 4 (cluster)**.** Sia D un insieme di oggetti. Si defnisce *"cluster"* C, wrt Eps ed $MinPts$ nell'insieme di oggetti D, un sottoniseme non vuoto di D, che soddisfa le seguenti condizioni :

1. *Massimalità*: $\forall p, q \in D$, se $p \in C \wedge q >_D p$ wrt MinPts e Eps, allora anche $q \in C$

2. *Connettività*: $\forall p, q \in C$, p è *"density-connected"* a q wrt MinPts e Eps

definzione 5 (noise)**.** Siano $C_1, \ldots, C_k$ tutti i cluster wrt Eps e , ovvero come un insieme di oggetti densamente connessi che è massimale rispetto MinPts in D. Si definisce *noise* l'insieme di oggetti D che non appartengono a nessun cluster C_i, noise=$\{p \in D | \forall i : p \notin C_i\}$.

Questo algoritmo presenta due proprietà fondamentali che ne permettono una computazione efficiente:

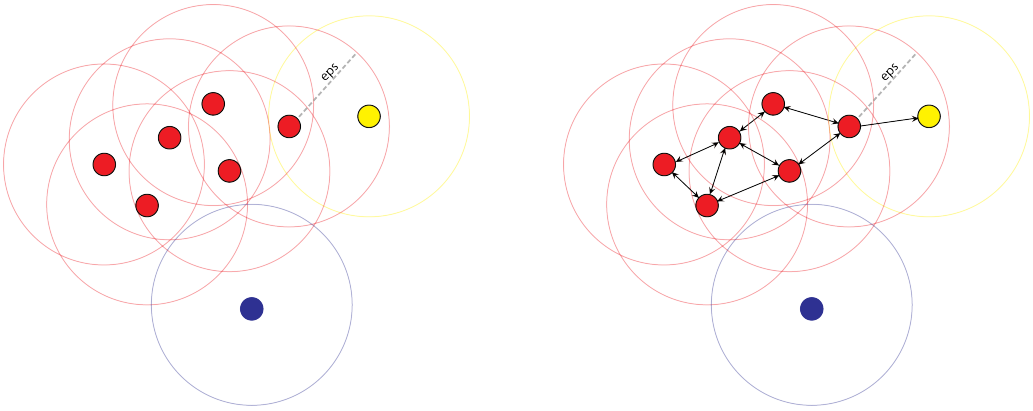
1. Dato un qualsiasi core-object, l'insieme dei punti density reachable da tale punto (w.r.t Eps,MinPts) costituirà un cluster.

2. Si consideri un cluster C, ciacun elemento di C è density-reachable da un ogni core-object in C. Ciascun cluster sarà quindi, univocamente identificato da uno qualsiasi dei suoi core-object.

L'algoritmo di DBSCAN, al fine di creare un cluster,parte da un punto arbitrario p e ritrova tutti i punti da esso density-reachable rispetto ai parametri Eps e $MinPts$, effettuando region-query [1] prima per p ed eventualmente per i vicini di p diretti ed indiretti. Se l'oggetto p è un core-object tale procura ci darà un cluster, altrimenti l'oggetto p sarà considerato come NOISE, e si passerà al oggetto successivo. Se p è un border-point sarà successivamente ri-etichettato quando, verrà considerato un core object dal quale p è density reachable. Le implementazioni classiche di DBSCAN fanno uso di indici spaziali come R-tree o X-tree, che consentono di ritrovare il viciniato di ciacun punto (region-query) in maniera efficiente $O(logn)$. Nel peggiore dei casi, DBSCAN visita ogni punto del dataset, ovvero esegue una region query per ciascun punto, ciò determina una complessità $0(n^2)$ dove n rappresenta il numero di punti nel dataset. Se si utilizza una struttura indicizzata come un R-tree, è possibile passare ad una coplessità $0(nlogn)$. Tali strutture indicizzate però, non riescono a scalare all'aumentare della dimensionalità dei dati: la performance delle region query passa da O(logn)to O(n), facendo degenerare, cioè la complessità temporale di dbscan in $O(n^2)$ rendendo poco adatto questo algoritmo al problema del clustering dei tweets. Nel lavoro di Guo et al. [7] vien proposto una variante dell'algoritmo DBSCAN che fa uso dell'LSH, al fine di ritrovare i nearest-neighbors di un punto, riducendo sensibilmente la complessità dell'algoritmo.

L'algoritmo DBSCAN riesce a generare dei cluster anche con un elevata presenza di rumore. Inoltre grazie alla definizione di "density-connected"3, riesce ad identificare cluster di forme arbitrarie. Inoltre, a differenza di altri algoritmi come k-means, non necessita di conoscere a priori il numero di cluster. Tutte queste caratteristiche rendono questo algoritmo particolarmente interessante nella scoperta di cluster in questo lavoro di tesi poiché: i messaggi di un microblog come Twitter contengono un'alta percentuale di rumore, e nel task di event-detection non è possibile sapere a priori il numero di eventi.

[1] una region-query di un punto p è la ricerca dei punti che ricadono nel viciniato di p



(a) MinPts=3 (b) Esempio di grafo di densità

Figure 1.2: rossi=core-objects, gialli=border, blue=noise

1.1.1 DBSCAN come ricerca di componeti connesse

Il problema del clustering, può essere ricondotto al problema della ricerca di componenti connesse in un grafo. In un grafo, una *componente connessa* è un sotto-grafo in cui ogni coppia di vertici è connessa da un cammino, e il sottografo non è connesso a nessun vertice del grafo di partenza. Per ottenere una corrispondenza fra una componente ed un cluster secondo dbscan, sarà sufficiente far sì che per ogni coppia di punti density-connected, esista un cammino che li colleghi. In figura 1.2b è mostrato come è possibile rappresentare un grafo-diretto a partire da connessioni di densità: per ogni coppia di oggetti di directly density reachable a, b con a core-object, verrà creato un arco (a, b), se anche b è un core object sarà creato un ulteriore arco da b ad a : (b, a). Talvolta però, un border-object può essere nel vicinato di più di un core-object anche appartenenti a cluster distinti, come mostrato in figura 1.3. Se fossero lasciati entrambi gli archi verso il border-object, verrebbe identificata un'unica componente connessa. In tali situazioni l'algoritmo DBSCAN non è deterministico: tale oggetto border potrà essere assegnato ad uno qualsiasi dei cluster. Sarà sufficiente fare in modo che ogni border object abbia un solo arco in entrata.

1.2 Sensitive Hashing

Un problema fondamentale che molti task di Data Mining devono affrontare è quello di esaminare i dati per trovare item "simili". Molto spesso questo problema è risolto cercando il Nearest Neighbor (o i primi k-nearest neighbor) di un oggetto in qualche spazio metrico R^d . Nel caso di poche dimen-

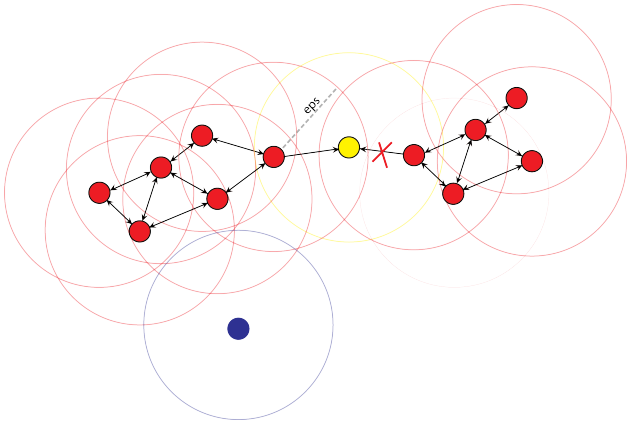


Figure 1.3: border object density reachable da due core-object

sioni (d=2, d=3) questo problema, si riesce a risolvere in maniera piuttosto efficiente (O(log n)) mediante l'utlizzo di strutture dati come K-D-Tree o R-tree che partizionano lo spazio delle feature. Tuttavia al crescere delle dimensioni (curse of dimensionality), queste strutture basate sul partizionamento dello spazio, degradano in una ricerca lineare [6] passando quindi ad una complessità quadratica. Per rendere meglio l'idea supponiamo di voler calcolare i documenti più simili in una collezione composta da un milione di documenti, si avrebbero quindi $\binom{100000}{2}$ coppie di cui calcolare la similarità. Se ci si impiega un microsecondo per ogni calcolo di similarità, sarebbero necessari quasi sei giorni per valutare tutte queste coppie. Se l'obiettivo è proprio il calcolo delle similarità di ogni possibile coppia, l'unico modo per ridurre il tempo necessario è usare una qualche forma di parallelismo. Tuttavia spesso, come nel caso del NNS-problem, si è interessati solo alle coppie più simili o più in generale a quelle coppie la cui similarità è al di sopra di una determinata soglia. É quindi sufficiente considerare solo quelle coppie che "probabilmente" sono simili, piuttosto che considerarle tutte. Proprio a tale scopo, se la dimensionalità dei dati è piuttosto alta, è possibile usare la tecnica del *Local Sensitive Hashing* (LSH) [5] [3]. L'idea chiave dell'LSH è quella di eseguire l'hasing dei punti attraverso molte funzioni di hashing, in modo tale che per ogni funzione la probabilità di collisione sia più alta per i punti vicini fra loro, rispetto ai punti più distanti. Dopo aver eseguito l'hashing tutte le coppie che sono state mappate nello stesso "bucket" possono essere considerate come *coppie candidate*, solo queste coppie saranno valutate per determinare quali sono realmente simili. Quello che ci si augura è che, le coppie di item fra loro dissimili non vegano mai mappate verso lo stesso bucket,tali coppie rappresentano infatti dei *false positive*. Allo stesso tempo ci si auspica che tutte (o la maggior parte) delle coppie realmente

simili siano mappate nel medesimo bucket, le coppie che simili che non sono ricadute nello stesso bucket rappresentano i *false-negative*

1.2.1 Teoria dell'LSH

L'LSH è un framework per costruire strutture dati che permettono di ricercare i "near neighbor" all'interno di una collezione di vettori altamente dimensionali. Dato un insieme P contenente vettori D-dimensionali, l'obiettivo è di costruire una struttura che consenta di ritrovare, per un qualsiasi query point q, tutti quei punti che giacciono in un raggio di distanza cR (dove c rappresenta un fattore di approssimazione > 1), o più formalmente gli R-*near neighbors* di q. Data una funzione di hashing h diremo che una coppia di oggetti (x, y) sarà una *candidate* se $h(x) = h(y)$. Sia $d(\cdot, \cdot)$ una qualche funzione definita su un insieme di oggetti S

definzione 6 (Locality-sensitive hashing)**.** Una famiglia di funzioni H si dice $(d_1, d_2, p_1, p_2) - Sensitive$ se $\forall (x, y) \in S, \forall h \in H$

- $if \quad d(x, y) \leq d_1 \Rightarrow P_H[h(x) == h(y)] \geq p_1,$
- $if \quad d(x, y) \geq d_2 \Rightarrow P_H[h(x) == h(y)] \leq p_2$

dove $p_1 > p_2$

A differenza di algoritmi di hashing classici dove si cercano di evitare delle collsioni per item diversi, in questo caso si cerca di massimizzare la probabilità di collisione per item vicini. Dato un oggetto q come query, verranno restituiti gli oggetti che giacciono nello stesso bucket $h(q)$

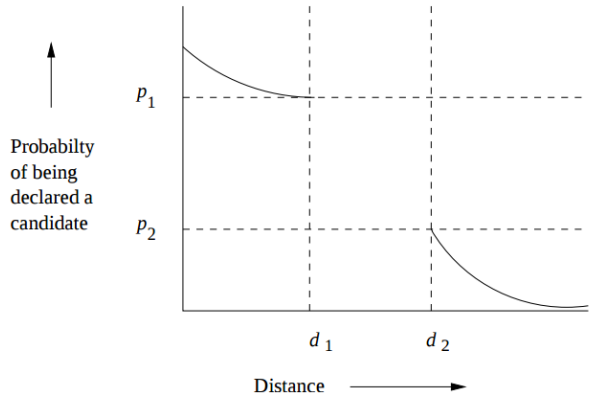
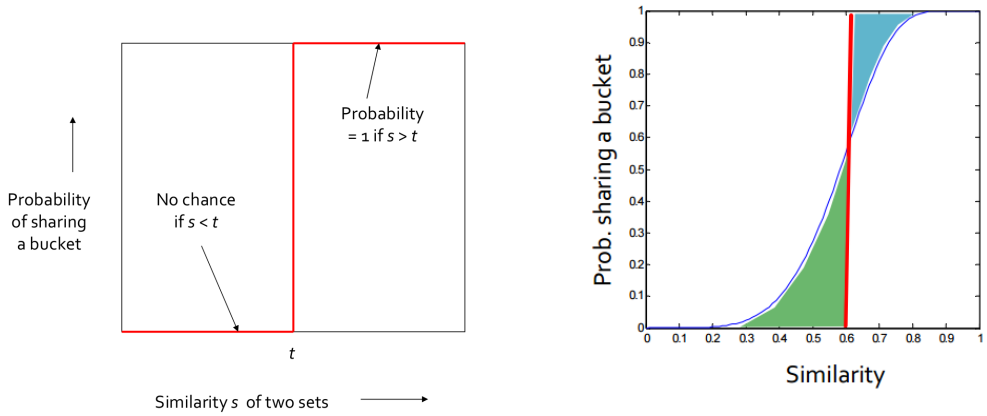


Figure 1.4: esempio di una funzione $(d_1, d_2, p_1, p_2) - sensisitve$

La figura 1.4 mostra un la probabilità che una possibile funzione di una famiglia $(d_1, d_2, p_1, p_2) - sensisitve$ dichiari una coppia come candidata.

Data una famiglia di funzioni hash $(d_1, d_2, p_1, p_2) - sensisitve$ H si può costruire una nuova famiglia di funzioni *H'* applicando la *AND-construction* su H come segue: ciascuna funzione di H' sarà data dalla concatenazione da r funzioni di H : $\{h_1....h_r\}$. Allora per ogni funzione h in *H'* diremo che: $h(x) = h(y)$ se $h_i(x) = h_i(y)$ per ogni $i...r$. Dato che le funzioni h_i sono scelte in maniera indipendente da H, possiamo asserire che *H'* è una famiglia $(d_1, d_2, p_1^r, p_2^r) - sensisitve$. Se invece costruiamo una nuova famiglia di funzione, per mezzo di una disgiunzione di b elementi di H (*OR-construction*), passeremo da una famiglia $(d_1, d_2, p_1, p_2) - sensisitve$ ad una $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b) - sensisitve$ *H'*. In questo caso diremo che: $h(x) = h(y)$ se esiste un valore di i tale che $h_i(x) = h_i(y)$ Se p è la probabilità che elemento di H dichiari una coppia (x, y) come candidata, allora $1 - p$ è la probabilità che non lo sia. Avendo b funzioni $h_1, ..., h_b$ avremo che la probabilità che nessuna di esse dichiari la coppia come candidata, pari a $(1-p)^b$, e di conseguenza $1-(1-p)^b$ sarà la probabilità che almeno una fra le b funzioni dichiarerà la coppia come candidata. É facile notare che la *And-Construction* fa decrescere tutte le probabilità, mentre la *OR-construction* ha l'effetto opposto.É però possibile, applicare in cascata queste due tecniche AND,OR in modo tale che la probabilità p_2 sia più bassa possibile, mentre p_1 sia quanto più possibile vicino ad 1, ottenendo a una famglia $(d_1, d_2, 1-(1-p_1^r)^b, 1-(1-p_2^r)^b) - sensisitve$. Scegliendo in maniera accurata i valori di r e di b sarà possibile controllare l'andamento della funzione di probabilità $1-(1-p^r)^b$. Tale funzione, una volta fissati i parametri r e b, descrive una *S-curve*. Come per qualsiasi *S-curve*, è possibile determinare il punto fisso, ovvero quel valore di t che rimane inalterato dopo aver applicato, la funzione $p = 1-(1-p^r)^b$. Tale valore può essere approssimato come $t \approx (1/b)^{\frac{1}{r}}$. Al di sotto di tale soglia, il valore di probabilità decresce,al di sopra cresce.Quindi, se se scegliamo la probabilità più alta p_1 al di sopra della soglia t, e p_2 al di sotto, avremo che il valore p_2 è ridotto e al contempo il valore p_1 viene innalzato. Bisogna quindi impostare i valori di b, r in modo tale che il punto fisso della funzione, corrisponda proprio al valore di similarità tale per cui due elementi li possiamo considerare "simili". La figura 1.5a mostra il caso ideale: dopo un valore di soglia la propabilità che due una coppia diventi candidata diviene immediatamente pari a 1, al di sotto pari a zero. Nella figura 1.5b viene mostrato invece il caso in cui $r = 5, b = 10$, l'area verde indica il false-positive rate, mentre l'area blue indica il false negative rate. Bisogna quindi, effettuare un tuning dei parametri b, r per cercare di ritrovare la maggior parte delle coppie di elementi realmente simili, e al contempo avendo pochi false-positive. Bisogna tenere a mente, però , che i false-positive possono



(a) s-curve ideale (b) s-curve r=5,b=10

Figure 1.5: esempi di S-Curve

essere filtrati con una fase di post-processing valutando la similarità reale fra le coppie restituite, i false negative, invece non possono essere in alcun modo recuperati, poiché rappresentano quelle coppie di item la cui distanza è bassa, ma che non sono state mappate nello stesso bucket.

1.2.2 LSH-Cosine

In questo lavoro di tesi poiché si stanno analizzando documenti testuali, verrà utilizzata una famiglia di funzioni di hashing local-sensitive per la distanza del coseno. In particolare sarà adottato lo schema di lsh proposto da Charikar[1] in cui la la probabilità che due punti collidano (ovvero che siano mappati verso lo stesso bucket), è proporzionale al coseno dell'angolo fra di loro. Data una collezione di vettori in R^d viene definita una famiglia di funzioni hashing come segue: si sceglie un vettore random $\vec{r}$ le cui componenti sono presse da una distribuzione Gaussiana. Proprio grazie a questo vettore random verrà definita una funzione di hashing come segue:

$$h_{\vec{r}}(\vec{u}) := \begin{cases} 1 & \text{se } \vec{r} \cdot \vec{u} \geq 0, \\ 0 & \text{se } \vec{r} \cdot \vec{u} \leq 0, \end{cases} \tag{1.1}$$

La figura 1.7 mostra l'interpretazione geometrica dell'equazione 1.1, ovvero valutare il segno del prodotto interno fra il vettore che definisce la funzione di hashing, e un un vettore item, equivale a stabilire se l'item è al di sotto o al di sopra dell'iperpiano identificato dalla funzione. Costruendo in questa

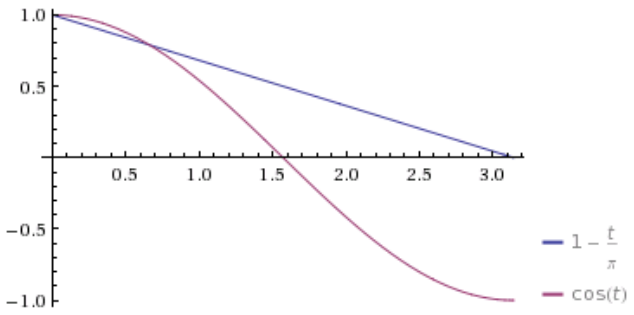


Figure 1.6: approsimazione sim. coseno

maniera una funzione di hashing si avrà che, per due vettori $\vec{u}, \vec{v}$,

$$P[h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v}))] = 1 - \frac{\theta(\vec{u}, \vec{v})}{\pi} \tag{1.2}$$

Secondo il teorema1.2 [4], la probabilità che un iperpiano random, separi due vettori è direttamente proporzionale all'angolo fra di essi. In figura 1.6 si può notare che per piccoli angoli (non vicini all'angolo retto), $1 - \frac{\theta}{\pi}$ è una buona approssimazione per $\cos(\theta)$. Inoltre, dall'equazione 1.2 si ha che

$$cos(\theta(u, v)) = cos((1 - P[h_r(u) = h_r(v)])\pi) \tag{1.3}$$

Grazie a questa equazione, è possibile stimare la probabilità del coseno attraverso il risultato dell'applicazione delle funzioni di hashing. Guardando con più attenzione si può notare che

$$P[h_r(u) = h_r(v)]) = g1 - hammingDistance(h_r(u), h_r(v))/r$$

In pratica, questa uguaglianza permette di passare dal problema del calcolo della similarità del coseno fra due vettori altamente dimensionali, al problema del calcolo della distanza di hamming fra due stringhe.

Anche questa famiglia di funzioni di hashing può essere aplificata con la tecnica AND-OR precedentemente descritta.

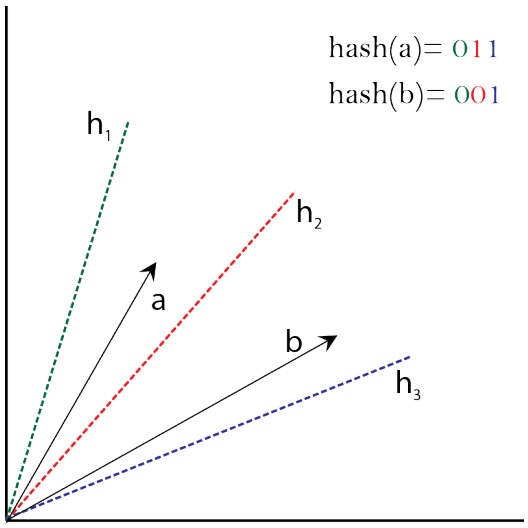


Figure 1.7: funzione di hashing

Bibliography

[1] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[2] Martin Ester. A density-based algorithm for discovering clusters in large spatial databases with noise. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 226–231. AAAI Press, 1996.

[3] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[4] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995.

[5] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[6] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[7] Y. P. Wu, J. J. Guo, and X. J. Zhang. A linear dbscan algorithm based on lsh. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 5, pages 2608–2614, Aug 2007.