

Next step: LSH Tuning

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures

Regolare con attenzione i parametri M, b ed r

Obiettivo specifico :

Identificare quasi tutte le copie di elementi che hanno firme simili

Eliminare il maggior numero possibile di coppie che NON presentano firme simili

Ridurre falsi positivi

Processo di ottimizzazione dei parametri di un algoritmo di LSH.

M = numero totale di tabelle hash che utilizziamo nell'algoritmo LSH. Ogni tabella hash è una struttura che raggruppa gli elementi in base alle loro firme.

Aumentare M = creare più tabelle

Migliora prob. Copie simili MA
Aumenta complessità computazionale

b = numero di bande in cui dividiamo la firma di ogni elemento. In LSH firma viene spezzata in b bande e ogni banda viene usata per calcolare un valore hash.
Valore b alto = bande piccole = algoritmo più selettivo, potrebbe far perdere coppie simili

Valore b basso = Aumenta prob di trovare coppie simili
Più falsi positivi

r = numero righe per banda = lunghezza di banda

Direttamente legato a b

Prodotto b ed r = lunghezza totale della firma

Valore r grande = bande più lunghe

Aumenta prob. Che due elementi simili finiscano nello stesso bucket

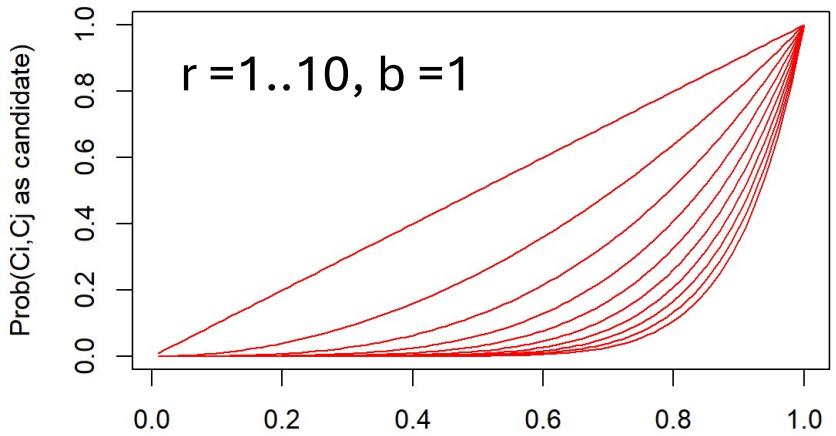
Può anche aumentare falsi positivi

Valore r piccolo = algoritmo restrittivo

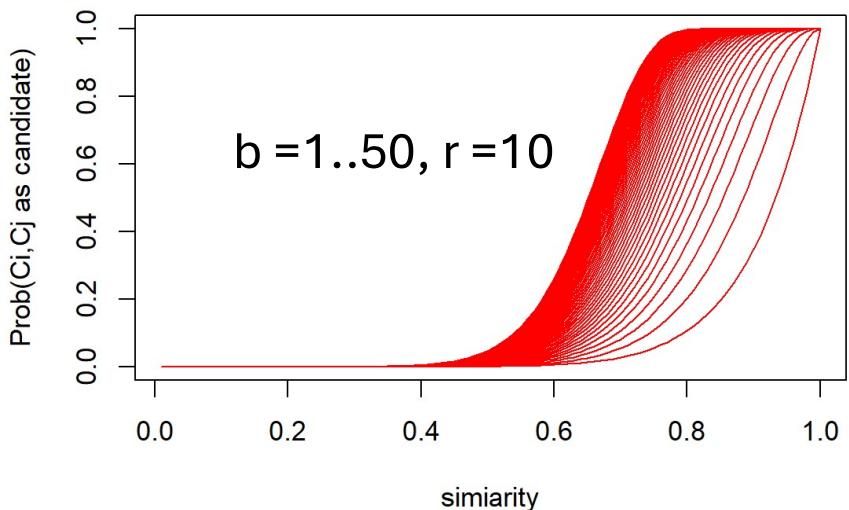
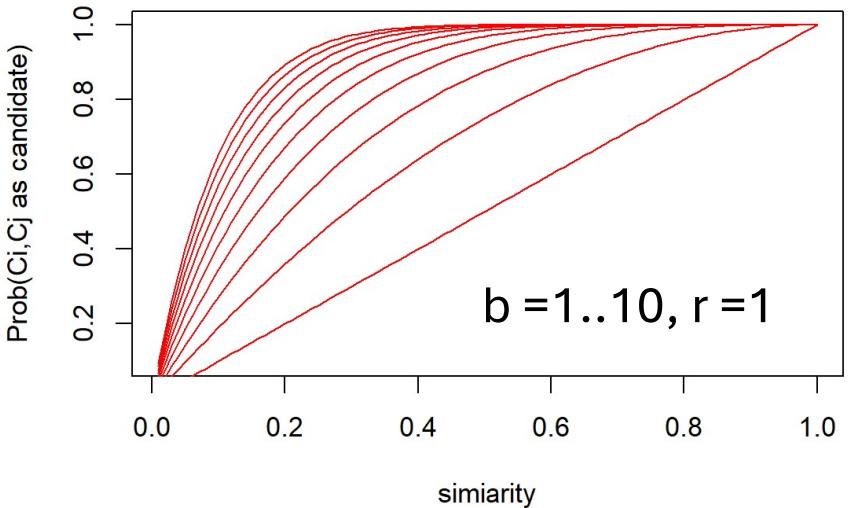
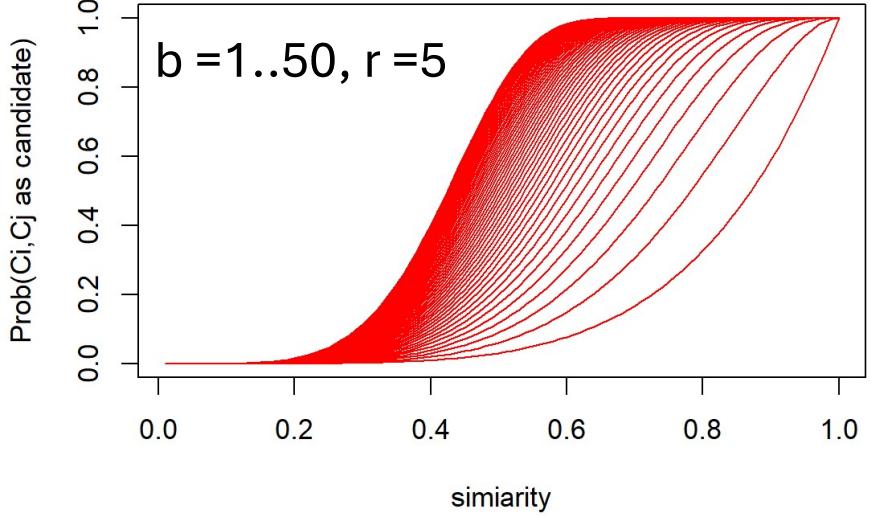
Probabilità che due elementi siano identificati come **candidate** in LSH in base alle loro similitudini

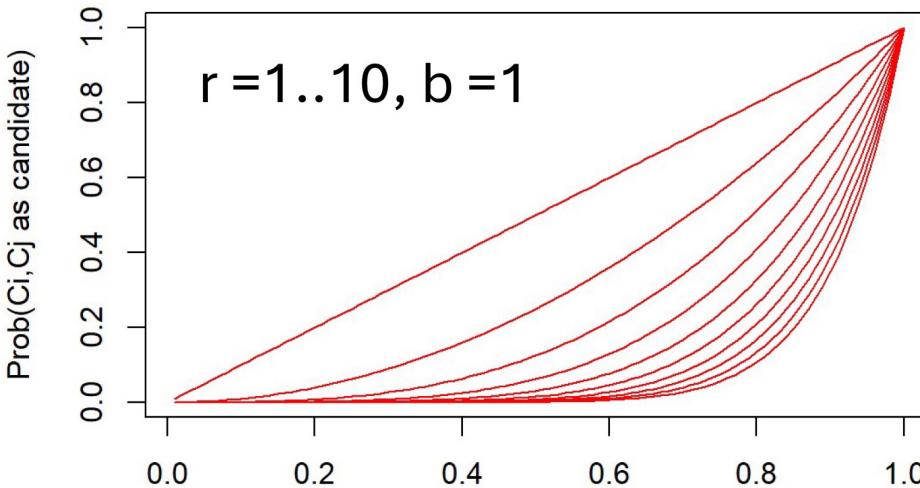
The S function

similitudine
di elementi



Similitudine
di elementi





La similarità tra due elementi (ad esempio, la similarità di Jaccard) determina la probabilità che una singola riga della firma sia uguale per entrambi. Se la similarità è, ad esempio, 0.8, c'è una probabilità dell'80% che una riga della firma coincida tra i due elementi.

Per una banda con r righe, tutte le r righe devono coincidere affinché i due elementi finiscano nello stesso bucket. La probabilità che ciò accada è la probabilità che una singola riga coincida, elevata alla potenza r . Ad esempio, se la similarità è 0.8 e $r = 2$, la probabilità che entrambe le righe coincidano è $0.8^2 = 0.64$.

Se r aumenta a 5, la probabilità diventa 0.8^5 circa 0.33 che è molto più bassa.

Quindi, più r è grande, più è difficile che tutte le righe coincidano, anche per elementi simili, riducendo la probabilità che finiscano nello stesso bucket.

Abbiamo b fisso a 1 (una sola banda) e r che varia da 1 a 10 (ogni banda ha una lunghezza che va da 1 a 10 righe).

La probabilità di identificare due elementi come candidati cresce lentamente con la similarità = Curva molto graduale.

Perchè questo comportamento?

Comportamento dovuto al fatto che con $b=1$ abbiamo solo una banda quindi la probabilità dipende interamente dalla lunghezza della banda (r). Quando r aumenta la probabilità che due elementi simili finiscano nello stesso bucket diminuisce, perché diventa più difficile che tutte le r righe della banda coincidano.

es.

→ Doc1 : [5, 2, 8, 1, 9, 3, 7, 4, 6, 2, 1, 8]

Doc2 : [5, 2, 8, 1, 9, 3, 7, 4, 6, 5, 3, 9]

$$\text{Divisione in bande} \\ b=4 \Rightarrow r = \frac{\text{# righe}}{b} = \frac{12}{4} = 3$$

Doc1 : Banda 1 (righe 1-3): [5, 2, 8]
Banda 2 (righe 4-6): [1, 9, 3]
Banda 3 (righe 7-9): [7, 4, 6]
Banda 4 (righe 10-12): [2, 1, 8]

Doc2 : Banda 1 (righe 1-3): [5, 2, 8]
Banda 2 (righe 4-6): [1, 9, 3]
Banda 3 (righe 7-9): [7, 4, 6]
Banda 4 (righe 10-12): [5, 3, 9]

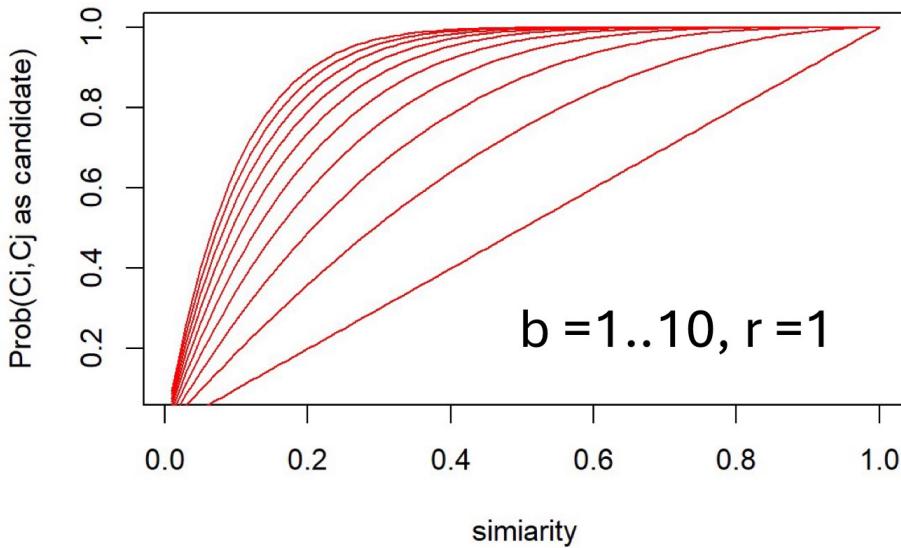
Ogni banda contiene 3 righe

Calcolo valore hash A bande

Banda 1
Doc1 : [5, 2, 8] → hash = bucket X
Doc2 : [5, 2, 8] → hash = bucket X
Doc2 : [5, 2, 8] → hash = bucket X

Banda 2

Identificare coppie candidate
Poi Doc1 e Doc2 coincidono stessa bucket in ALMENO 1 banda => coppie candidate!

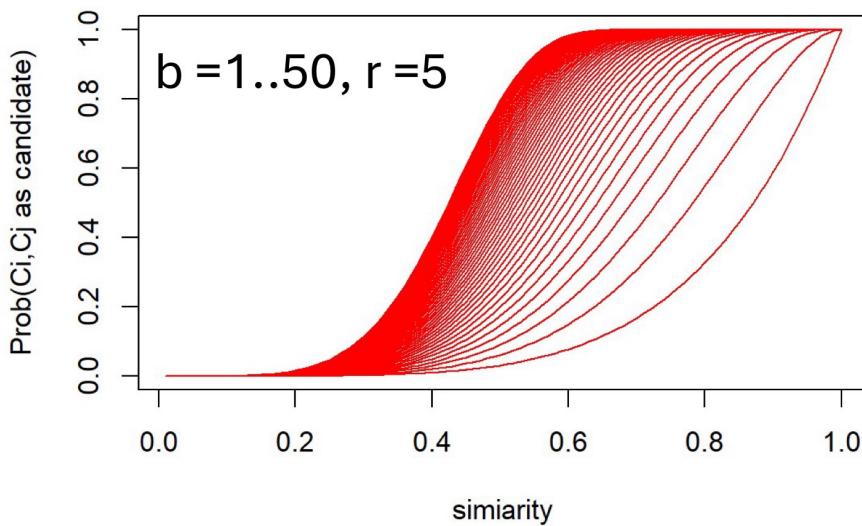


Qui abbiamo r fisso a 1 (ogni banda ha una sola riga) e b varia da 1 a 10 (quindi il numero di bande aumenta).

Curva più ripida rispetto al grafico precedente. La probabilità cresce rapidamente con la similarità: per similarità basse la probabilità è quasi 0, mentre per similarità alte si avvicina rapidamente a 1.

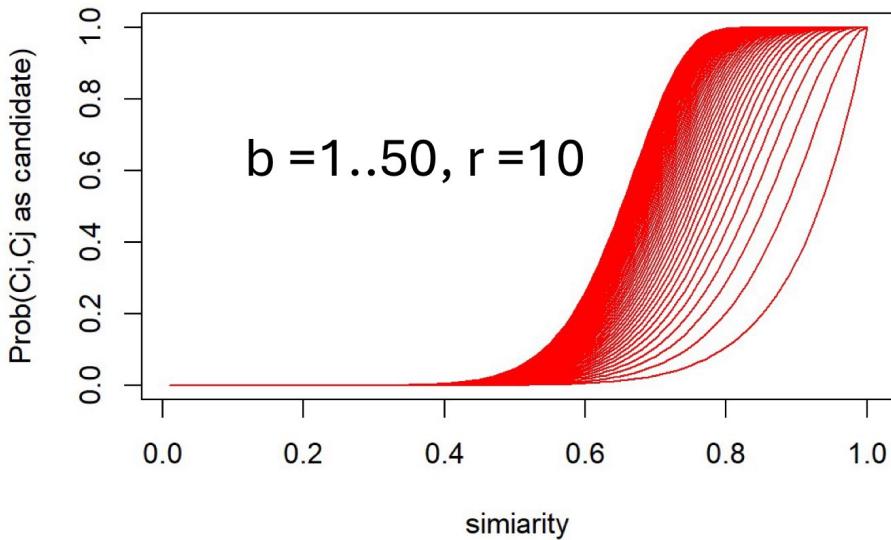
Perchè ciò accade?

Con $r=1$ ogni banda è molto piccola (una sola riga) quindi è più facile che due elementi simili abbiano lo stesso valore hash in una banda. Aumentando b , ci sono più bande, il che aumenta la probabilità che due elementi simili coincidano in almeno una banda, ma riduce la probabilità per elementi poco simili, rendendo la curva più selettiva.



Qui r è fisso a 5 (ogni banda ha 5 righe) e b varia da 1 a 50. Curva S ancora più ripida. La probabilità è quasi 0 per similarità inferiori a 0.5, ma cresce bruscamente e si avvicina a 1 per similarità superiori a 0.7.

Con $r=5$ ogni banda è più lunga rispetto al caso con $r=1$ quindi è più difficile che due elementi poco simili abbiano lo stesso valore hash in una banda. Aumentando b , ci sono più bande per trovare una corrispondenza, ma la selettività rimane alta grazie a r più grande. Questo rende l'algoritmo più efficace nel distinguere tra elementi simili e non simili.



SE VOGLIAMO CATTURARE PIÙ COPIE SIMILI (ANCHE AL COSTO DI QUALCHE FALSO POSITIVO) POSSIAMO SCYBIENE r PICCOLA, b GRANDE
SE INVECE VOGLIEMO ESSERE SELETTIVI r GRANDE, b GRANDE

Qui r è fisso a 10 (ogni banda ha 10 righe) e b varia da 1 a 50. La curva è la più ripida di tutte. La probabilità rimane vicina a 0 per similarità inferiori a 0.7 ma schizza a 1 per similarità superiori a 0.8.

Con $r=10$ le bande sono ancora più lunghe quindi è estremamente improbabile che due elementi poco simili finiscano nello stesso bucket.

Questo rende l'algoritmo molto selettivo: solo elementi con similarità molto alta (vicino a 1) hanno una probabilità significativa di essere identificati come candidati.

Aumentare b amplifica ulteriormente questo effetto, perché con più bande ci sono più possibilità di trovare una corrispondenza per elementi molto simili.

Ecco la traduzione in italiano del testo fornito:

- L'esempio che abbiamo visto in precedenza è un esempio di famiglia di funzioni che possono essere combinate per distinguere tra coppie a bassa distanza da coppie ad alta distanza
- Qui descriveremo altre famiglie di funzioni, oltre a quella del minhash
- Ci sono 3 condizioni che dobbiamo garantire:
 - Le coppie vicine devono essere candidate più probabili rispetto a quelle distanti
 - Statisticamente indipendenti: stimare la probabilità che due o più funzioni diano una certa risposta
 - Efficienti:
 - Il tempo necessario per identificare le coppie candidate deve essere molto inferiore al calcolo della distanza tra tutte le coppie
 - Combinabili per costruire funzioni che siano migliori nell'evitare falsi positivi e negativi

[Il testo è stato generato con l'aiuto dell'IA]

LSH: families of functions

Idea = vogliamo distinguere coppie di elementi che sono “vicini” (simili) da coppie che sono “lontane” senza dover calcolare la distanza tra tutte le possibili coppie

Operazione troppo costosa in termini computazionali

Locality-Sensitive functions

- The example we have previously seen is an example of **family of function** that can be combined to distinguish between pair at a low distance from pairs at a high distance
- Here we will describe other families of functions, beside the minhash one.
- There are **3 conditions** that we need to ensure:
 - Close pair must be more likely candidates than distant ones
 - Statistically independent: estimate probability that two or more function will give a certain response
 - Efficient:
 - Time needed to identify candidate pair much less than computing distance between all pairs
 - Combinable to build functions that are better at avoiding false positive and negatives

1) Le coppie vicine devono avere più probabilità di essere candidate rispetto a quelle lontane

Vicinanza definita da una metrica di distanza (distanza euclidea, somiglianza di Jaccard...)

if (x) e (y) vicini then

funzione deve avere una probabilità più alta di segnarli come coppia candidata

LSF sensibile alla località : reagiscono alla vicinanza tra gli elementi

2) Indipendenza statistica

Quando usiamo più funzioni della stessa famiglia vogliamo che siano indipendenti tra loro.

Possiamo combinare le loro decisioni e stimare meglio la probabilità che (x) e (y) siano una coppia candidata

Se due funzioni dessero sempre la stessa risposta, sarebbero ripetitive. Noi vogliamo l'indipendenza ogni funzione deve offrire un punto di vista nuovo.

3) Efficienza

Il tempo necessario per identificare le coppie candidate con queste funzioni deve essere molto inferiore rispetto al calcolo della distanza tra tutte le coppie possibili.

Se LSF fosse più lento di un metodo brute-force che testa ogni possibile coppia, la sua implementazione non avrebbe senso.

Funzioni devono essere combinabili.

Più precisione, meno falsi positivi o falsi negativi

Locality-Sensitive functions

- What is a Locality-Sensitive function?
 - It is a function that takes in input two items and render a decision about whether these items should be a candidate pair.
 - Shorthands:
 - $f(x) = f(y)$ x and y are candidate pair
 - $f(x) \neq f(y)$ x and y are **not** candidate pair Scattered copies
- A collection of such functions is called a family of Locality-Sensitive functions
- We say that a family F is (d_1, d_2, p_1, p_2) – *sensitive* if:
 - $\forall f \in F:$ $d(x, y) \leq d_1 \Rightarrow \Pr[f(x) = f(y)] \geq p_1$
 - If $d(x, y) \geq d_2$ the probability that $f(x) = f(y)$ is at most p_2

Una famiglia di funzioni F è definita **(d₁,d₂,p₁,p₂)-sensitive** se soddisfa due condizioni legate alla distanza tra (x) e (y) e alla probabilità che una funzione (f) scelta a caso dalla famiglia (F) dice che (x) e (y) sono "simili" (vale $f(x) == f(y)$).

Queste condizioni si basano su due soglie di distanza (d_1 e d_2) e due probabilità (p_1 e p_2):

Per ogni f appartenente alla famiglia di funzioni F :

1) Per coppie "vicine" (distanza minore o uguale di d_1):

- If $d(x, y) \leq d_1$ the probability that $f(x) = f(y)$ is at least p_1

$f(x) = f(y)$ ALMENO p_1

2) Per coppie "lontane" (distanza maggiore o uguale di d_2):

- If $d(x, y) \geq d_2$ the probability that $f(x) = f(y)$ is at most p_2

$f(x) = f(y)$ MASSIMO p_2

Assumiamo che $p_1 > p_2$

Coppie vicine hanno una probabilità più alta di essere considerate simili rispetto alle coppie lontane.

LSH for min-hashing

- We have
 - S = space of all sets
 - d = Jaccard distance,
*Freake indiki
to be printed rest indiki*
 - H is family of Min-Hash functions for all permutations of rows
- For any hash function $h \in H$:
$$P[h(x) = h(y)] = 1 - d(x, y)$$
- Simply restates theorem about Min-Hashing in terms of distances rather than similarities

Primo punto: Definizione del contesto

Spazio degli insiemi (S)

Definiamo formalmente:

$$S = \text{spazio di tutti gli insiemi}$$

S rappresenta la collezione di insiemi da confrontare. Ad esempio, ogni elemento di S può essere un insieme di parole rappresentante un documento:

$$x = \{a, b, c\} \quad \text{e} \quad y = \{b, c, d\}$$

Distanza di Jaccard (d)

La distanza di Jaccard è definita come:

$$d(x, y) = 1 - \text{Somiglianza di Jaccard}(x, y)$$

dove la somiglianza di Jaccard è:

$$\text{Somiglianza di Jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|}$$

Esempio numerico

Per $x = \{a, b, c\}$ e $y = \{b, c, d\}$:

- Intersezione: $x \cap y = \{b, c\} \Rightarrow |x \cap y| = 2$
- Unione: $x \cup y = \{a, b, c, d\} \Rightarrow |x \cup y| = 4$
- Somiglianza: $\frac{2}{4} = 0.5$
- Distanza: $d(x, y) = 1 - 0.5 = 0.5$

Famiglia di funzioni Min-Hash (H)

H contiene tutte le funzioni Min-Hash generate da permutazioni di righe. Esempio di matrice caratteristica:

Elemento	x	y
a	1	0
b	1	1
c	1	1
d	0	1

Una funzione $h \in H$ si ottiene:

1. Scegliendo una permutazione casuale delle righe (es: d, c, b, a)
2. Per ogni insieme, prendendo il primo elemento (nella permutazione) con valore 1

Secondo punto: Proprietà del Min-Hashing

Teorema fondamentale

Per ogni $h \in H$:

$$P[h(x) = h(y)] = 1 - d(x, y)$$

Questa probabilità equivale alla somiglianza di Jaccard.

Esempio

Per $x = \{a, b, c\}$ e $y = \{b, c, d\}$:

$$P[h(x) = h(y)] = 0.5$$

Spiegazione intuitiva

La probabilità dipende dall'intersezione: la prima riga con 1 comune a entrambi gli insiemi è proporzionale alla loro sovrapposizione.

Terzo punto: Collegamento con LSH

Definizione di famiglia LSH

H è (d_1, d_2, p_1, p_2) -sensitive se:

$$\begin{cases} d(x, y) \leq d_1 \implies P[h(x) = h(y)] \geq p_1 \\ d(x, y) \geq d_2 \implies P[h(x) = h(y)] \leq p_2 \end{cases}$$

Esempio numerico

Con $d_1 = 0.2$, $d_2 = 0.6$, $p_1 = 0.8$, $p_2 = 0.4$:

- Se $d(x, y) \leq 0.2$: $P \geq 0.8$
- Se $d(x, y) \geq 0.6$: $P \leq 0.4$

Locality-Sensitive functions

- The family of **minhash functions**, assuming that we are using a **Jaccard Distance**, is
$$(d_1, d_2, 1 - d_1, 1 - d_2) - \text{sensitive}$$
- For any d_1 and d_2 such that $0 < d_1 < d_2 \leq 1$
- For other distances, there is **no guarantee** that it might have a locality-sensitive family of hash functions.
- Min-hash H is a $(1/3, 2/3, 2/3, 1/3)$ -sensitive family for S and d.

Famiglie di funzioni Min-Hash è locality-sensitive rispetto alle distanze di Jaccard.

Noi sappiamo che

$$\begin{cases} d(x, y) \leq d_1 \implies P[h(x) = h(y)] \geq p_1 \\ d(x, y) \geq d_2 \implies P[h(x) = h(y)] \leq p_2 \end{cases}$$

Nel caso del Min-Hashing sappiamo che:

$$P[h(x) = h(y)] = 1 - d(x, y) \rightarrow \text{Distanza di Jaccard}$$

Quindi sappiamo che $P[h(x) = h(y)]$ = similarità di Jaccard

Allora noi sappiamo che:

Se $d(x, y) \leq d_1$:

$$P[h(x) = h(y)] = 1 - d(x, y) \geq 1 - d_1$$

$$P[h(x) = h(y)] \geq 1 - d_1 \quad / \quad 6 \leq 7$$

Se $d(x, y) \geq d_2$:

$$P[h(x) = h(y)] = 1 - d(x, y) \leq 1 - d_2$$

Min-Hash è $(d_1, d_2, 1-d_2, 1-d_2)$

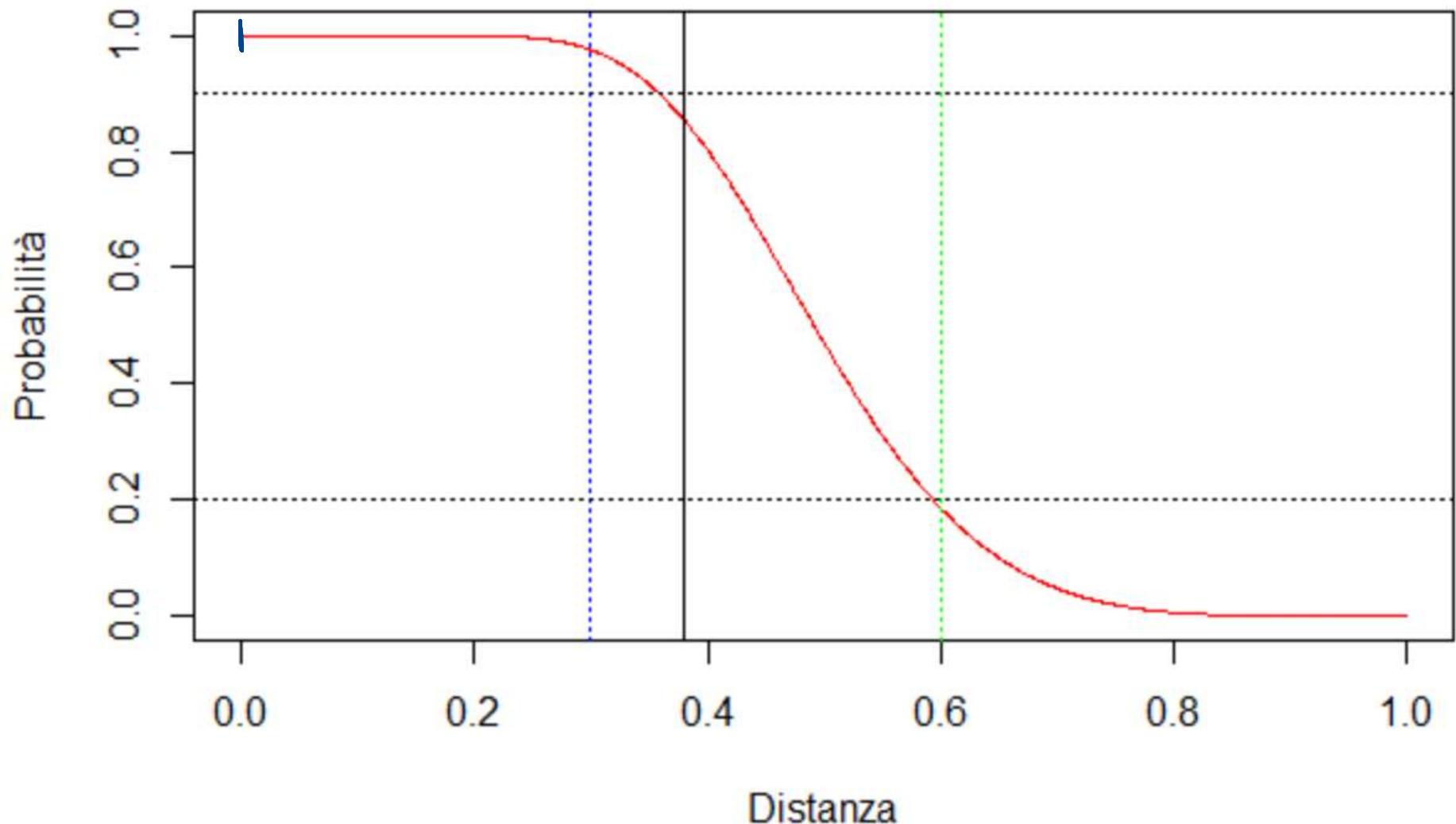
$$0 < d_1 < d_2 \leq 1$$

Min-Hashing è locality-sensitive specificato per le distanze di Jaccard.

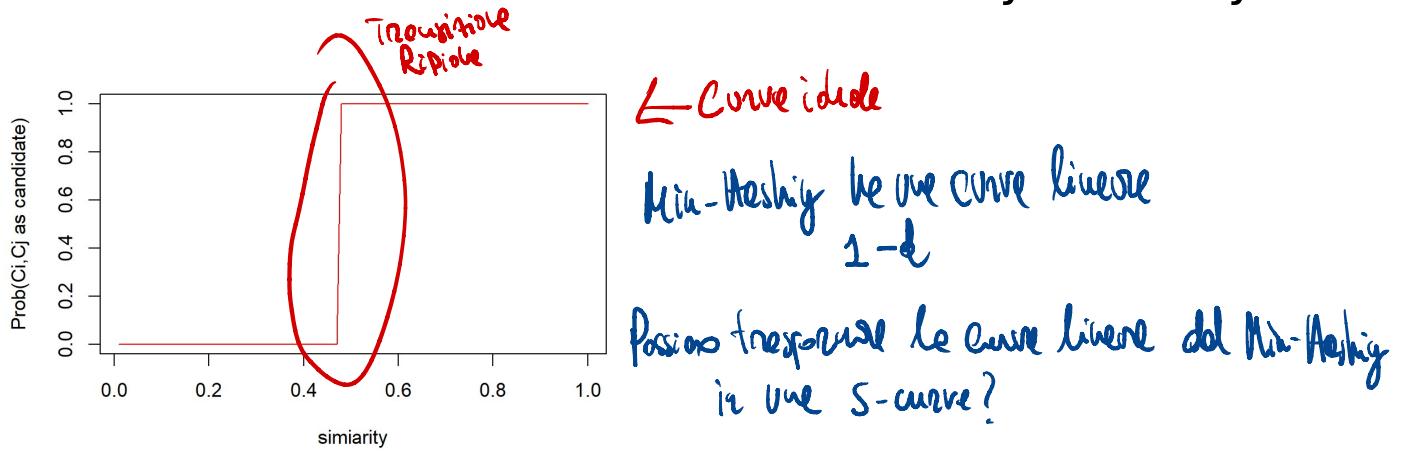
Se usessimo altre metriche, non i garantire che Min-Hashing funzioni come funzione locality-sensitive

$$P[h(x) = h(y)] = 1 - d(x, y)$$

Non va bene



- Can we reproduce the “S-curve” effect we saw before for any LS family?



- The “bands” technique we learned for signature matrices carries over to this more general setting Tecnica delle bande può essere applicata a questo contesto più generale
- Can do LSH with any (d_1, d_2, p_1, p_2) -sensitive family!
- Two constructions:
 - AND construction like “rows in a band”
 - OR construction like “many bands”

Amplifying Locality-Sensitive functions

- Lets build a new family F' where each $f \in F'$ is built from r members of a family F , for some fixed r .
 - f is built from $\{f_1, \dots, f_r\}$ such that $f(x) = f(y)$ if and only if $f_i(x) = f_i(y)$ for all $i = 1, 2, \dots, r$
- If F is (d_1, d_2, p_1, p_2) – *sensitive*
 F' will be $(d_1, d_2, (p_1)^r, (p_2)^r)$ – *sensitive*
- We are lowering all probabilities if we choose F and r judiciously
- This process is called **AND-construction**

Dati due punti (x) e (y) la funzione f e F' restituisce lo stesso valore per (x) e (y) ($f(x)=f(y)$) se e solo se tutte le r funzioni f_1, f_2, \dots, f_r che compongono (f) restituiscono lo stesso valore per (x) e (y)

$$r=3 \quad \{f_1, f_2, f_3\} \quad f(x) = f(y) \Rightarrow \begin{cases} f_1(x) = f_1(y) \\ f_2(x) = f_2(y) \\ f_3(x) = f_3(y) \end{cases}$$

AND - CONSTRUCTION

Costruisce una famiglia F' combinando r funzioni di F

Definisce la nuova funzione f e F' in modo che $f(x)=f(y)$ solo se tutte le r funzioni sono d'accordo.

Modifica la probabilità di collisione, portandole da (p_1, p_2) a $((p_1)^r, (p_2)^r)$ amplificando la differenza tra punti vicini e lontani

Richiede una scelta oculata di F e di r per ottimizzare i risultati

Esempio

$$F(d_1, d_2, p_L, p_r) \text{ - sensitive}$$

$$p_L = 0.9 \quad p_r = 0.1$$

$$r=3 \text{ per costruire } F'$$

$$F' \text{ sarebbe } (d_1, d_2, (p_L)^r, (p_r)^r) \text{ - sensitive quindi}$$

$$(p_L)^r = 0.729$$

$$\frac{p_L}{p_r} = 9$$

$$(p_r)^r = 0.001$$

$$\frac{p_L^r}{p_r^r} = 729$$

AND construction

Amplificat rapporto tra probabilità

P' molto più efficace nel distinguere
punti vicini da punti lontani

F' molto selettiva \Rightarrow Punti buoni: prob collisione molto più alta

Hash function independencies

- Independence of hash functions (HFs) really means that the prob. of two HFs saying “yes” is the product of each saying “yes” But two particular hash functions could be highly correlated
- For example, in Min-Hash if their permutations agree in the first one million entries
- However, the probabilities in definition of a LSH-family are over all possible members of H, H' (i.e., average case and not the worst case)

Le funzioni hash sono ampiamente utilizzate in informatica per applicazioni come la ricerca di dati simili, la compressione e la crittografia. Un aspetto cruciale delle funzioni hash è la loro **indipendenza**, ovvero la capacità di garantire che i risultati prodotti da diverse funzioni hash siano statisticamente indipendenti l'uno dall'altro. Questo documento esplora il concetto di indipendenza delle funzioni hash, con particolare attenzione al caso pratico del Min-Hash e alla sua applicazione nei Local Sensitive Hashing (LSH).

1 Definizione di Indipendenza delle Funzioni Hash

Due funzioni hash h_1 e h_2 si dicono **indipendenti** se la probabilità che entrambe producano lo stesso risultato per due input x e y è pari al prodotto delle loro probabilità individuali. Formalmente, se p_1 è la probabilità che $h_1(x) = h_1(y)$ e p_2 è la probabilità che $h_2(x) = h_2(y)$, allora:

$$P(h_1(x) = h_1(y) \wedge h_2(x) = h_2(y)) = p_1 \times p_2.$$

Tuttavia, in pratica, due funzioni hash specifiche potrebbero non essere completamente indipendenti. Questo accade quando esiste una correlazione tra le funzioni, ovvero quando il comportamento di una funzione influenza o è simile a quello dell'altra.

2 Correlazione nelle Funzioni Hash: Esempio con Min-Hash

Il **Min-Hash** è una tecnica utilizzata nei Local Sensitive Hashing (LSH) per stimare la similarità di Jaccard tra insiemi. Ad esempio, può essere usato per trovare documenti simili basandosi sui loro contenuti. Il Min-Hash funziona generando permutazioni casuali degli elementi di un insieme e utilizzando il primo elemento di ogni permutazione per calcolare il valore hash.

Un problema che può verificarsi è che due funzioni Min-Hash condividono le stesse permutazioni per un gran numero di elementi. Ad esempio, se due funzioni utilizzano "le prime un milione di voci" dello stesso insieme di elementi, i loro valori hash saranno identici per molti input. Questo rende le due funzioni altamente correlate, violando l'assunzione di indipendenza.

3 Probabilità nel Caso Medio

Nella definizione di una famiglia LSH, le probabilità di collisione (ad esempio, p_1 per punti vicini e p_2 per punti lontani) sono calcolate come una media su tutte le possibili funzioni hash nella famiglia H o H' . Questo significa che le proprietà di una famiglia LSH sono garantite **nel caso medio**, non nel caso peggiore.

In altre parole, anche se alcune funzioni hash specifiche possono essere correlate, in media, scegliendo funzioni a caso dalla famiglia, l'indipendenza è approssimativamente rispettata. Questo approccio statistico permette di ottenere risultati affidabili pur non garantendo l'indipendenza assoluta per ogni coppia di funzioni hash.

4 Conclusione

L'indipendenza delle funzioni hash è un requisito fondamentale per molte applicazioni pratiche, come il Local Sensitive Hashing. Tuttavia, in pratica, l'indipendenza perfetta è difficile da ottenere a causa di fattori come la correlazione tra funzioni hash. Nonostante ciò, l'approccio basato sul caso medio garantisce che le proprietà desiderate siano soddisfatte in modo approssimativo, rendendo le famiglie LSH uno strumento efficace per la gestione di grandi quantità di dati.

Amplifying Locality-Sensitive functions

- Now lets build **a family F'** where each $f \in F'$ is built from **b** members of a family F , for some fixed b.
 - f is built from $\{f_1, \dots, f_b\}$ such that $f(x) = f(y)$ if and only if $f_i(x) = f_i(y)$ for one or more values i
- If F is (d_1, d_2, p_1, p_2) – *sensitive*
 F' will be $(d_1, d_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b)$ – *sensitive*
- We are increasing all probabilities if we choose F and b judiciously
- This process is called **OR-construction**

Spiegazione Dettagliata delle Slide

1. Costruzione della Famiglia F' :

- Si parte da una famiglia di funzioni F e si costruisce una nuova famiglia F' .
- Ogni funzione $f \in F'$ è composta da b funzioni $\{f_1, f_2, \dots, f_b\}$ estratte da F .
- La funzione f è definita in modo che $f(x) = f(y)$ se e solo se almeno una delle funzioni f_i soddisfa $f_i(x) = f_i(y)$. Questo corrisponde a un'operazione logica **OR**: basta una corrispondenza tra le b funzioni per considerare x e y "simili".

2. Sensibilità della Famiglia F' :

- Se F è (d_1, d_2, p_1, p_2) -sensibile, allora F' sarà $(d_1, d_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b)$ -sensibile.
- **Significato dei parametri:**
 - d_1 : Soglia di "vicinanza" (se due punti hanno distanza $\leq d_1$, sono considerati vicini).
 - d_2 : Soglia di "lontananza" (se due punti hanno distanza $\geq d_2$, sono considerati lontani).
 - p_1 : Probabilità che una funzione $f_i \in F$ collassi punti vicini ($\leq d_1$).
 - p_2 : Probabilità che una funzione $f_i \in F$ collassi punti lontani ($\geq d_2$).
- **Formula delle probabilità per F' :**
 - La probabilità che **almeno una** delle b funzioni collassi punti vicini è $1 - (1 - p_1)^b$.
 - Analogamente, per punti lontani: $1 - (1 - p_2)^b$.

3. Amplificazione delle Probabilità:

- L'OR-construction aumenta sia p_1 che p_2 , ma l'obiettivo è massimizzare il divario tra p'_1 e p'_2 .
- **Esempio numerico:**
 - Supponiamo F abbia $p_1 = 0.5$ e $p_2 = 0.2$.
 - Con $b = 2$:

$$p'_1 = 1 - (1 - 0.5)^2 = 0.75,$$
$$p'_2 = 1 - (1 - 0.2)^2 = 0.36.$$

- Il divario $p'_1 - p'_2$ aumenta da 0.3 a 0.39, migliorando la discriminazione.

4. Scelta Ottimale di b :

- Aumentando b , p'_1 e p'_2 crescono, ma il tasso di crescita dipende da p_1 e p_2 .
- Se $p_1 > p_2$, p'_1 cresce più rapidamente di p'_2 , almeno inizialmente. Tuttavia, oltre un certo b , p'_2 potrebbe avvicinarsi a p'_1 , riducendo l'efficacia.
- **Obiettivo:** Scegliere b in modo che p'_1 sia significativamente maggiore di p'_2 , bilanciando falsi positivi e negativi.

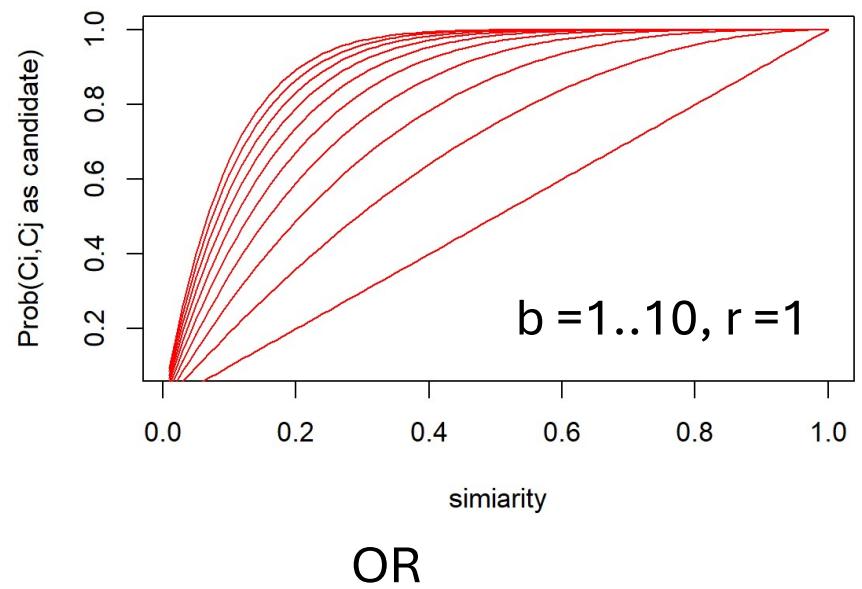
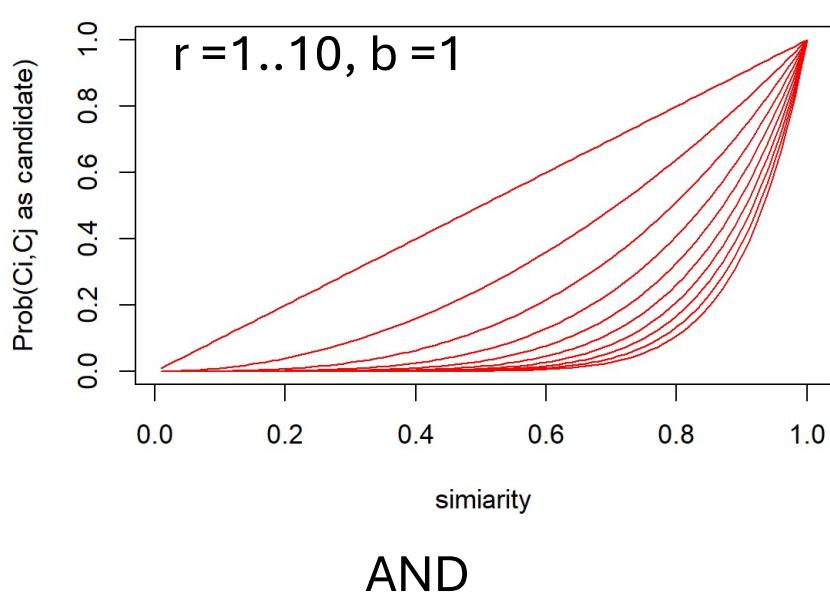
5. Conclusione (OR-Construction):

- Questo metodo permette di **ridurre i falsi negativi** (punti vicini non riconosciuti) aumentando p'_1 , ma introduce più **falsi positivi** (punti lontani riconosciuti come vicini) a causa di p'_2 .
- La chiave è selezionare b e F in modo che p'_1 sia sufficientemente alta e p'_2 rimanga accettabilmente bassa, migliorando le prestazioni complessive dell'algoritmo LSH (Locality-Sensitive Hashing).

Formula di Sintesi:

$$F' \text{ è } (d_1, d_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b) \text{-sensitive}$$

- AND and OR constructions can be **cascaded** in order to make the low probability close to 0 and the high probability close to 1.



- By choosing b and r properly we can make the lower probability close to 0 and the higher close to 1.
- As for the signature matrix, we can use the AND construction followed by the OR construction or vice-versa
- Or any sequence of AND's and OR's alternating

Example

- $(0.2, 0.8, 0.8, 0.2)$ -sensitive
- $B=4$ $r = 4$.
 - AND OR $(0.2, 0.8, 0.878, 0.0064)$ -sensitive
 - OR AND $(0.2, 0.8, 0.9936, 0.1215)$ -sensitive
 - Apply again OR and next AND
 - $(0.2, 0.8, 0.9999996, 0.0008715)$ -sensitive

Cascata di Costruzioni AND e OR

L'obiettivo è combinare le operazioni **AND** e **OR** per avvicinare:

- La probabilità **bassa** (collisione di punti lontani) a **0**.
- La probabilità **alta** (collisione di punti vicini) a **1**.

Esempio di parametrizzazione: Una famiglia di funzioni iniziale (d_1, d_2, p_1, p_2) -sensibile, ad esempio $(0.2, 0.8, 0.8, 0.2)$:

- $d_1 = 0.2$: Soglia di vicinanza.
- $d_2 = 0.8$: Soglia di lontananza.
- $p_1 = 0.8$: Probabilità di collisione per punti vicini.
- $p_2 = 0.2$: Probabilità di collisione per punti lontani.

Come Funziona la Cascata

AND-Construction

- Richiede che **tutte** le r funzioni in una "banda" collassino x e y .
- Formula:

$$p_{\text{AND}} = p^r$$

- **Effetto:** Riduce p_1 e p_2 , ma p_2 diminuisce più rapidamente.
Esempio: $0.2^4 = 0.0016$.

OR-Construction

- Richiede che **almeno una** delle b funzioni collassi x e y .

- Formula:

$$p_{\text{OR}} = 1 - (1 - p)^b$$

- **Effetto:** Aumenta p_1 e p_2 , ma p_1 cresce più rapidamente.
Esempio: $1 - (1 - 0.8)^4 = 0.9984$.

Esempi di Cascate

Caso 1: AND seguito da OR

Parametri iniziali: $(0.2, 0.8, 0.8, 0.2)$.

- **Step 1 (AND con $r = 4$):**

$$p'_1 = 0.8^4 = 0.4096, \quad p'_2 = 0.2^4 = 0.0016$$

- **Step 2 (OR con $b = 4$):**

$$p_1'' = 1 - (1 - 0.4096)^4 \approx 0.878, \quad p_2'' = 1 - (1 - 0.0016)^4 \approx 0.0064$$

- **Risultato:** $(0.2, 0.8, 0.878, 0.0064)$ -sensibile.

Caso 2: OR seguito da AND

- **Step 1 (OR con $b = 4$):**

$$p_1' = 1 - (1 - 0.8)^4 = 0.9984, \quad p_2' = 1 - (1 - 0.2)^4 = 0.5904$$

- **Step 2 (AND con $r = 4$):**

$$p_1'' = 0.9984^4 \approx 0.9936, \quad p_2'' = 0.5904^4 \approx 0.1215$$

- **Risultato:** $(0.2, 0.8, 0.9936, 0.1215)$ -sensibile.

Caso 3: Cascata Multipla (OR → AND → OR → AND)

- Dopo ulteriori passaggi:

$$p_1''' \approx 0.999996, \quad p_2''' \approx 0.0008715$$

- **Risultato finale:** $(0.2, 0.8, 0.999996, 0.0008715)$ -sensibile.

Scelta dei Parametri b e r

- **Regola generale:**

- Usare **AND** per ridurre p_2 (falsi positivi).
- Usare **OR** per aumentare p_1 (ridurre falsi negativi).

- **Trade-off:**

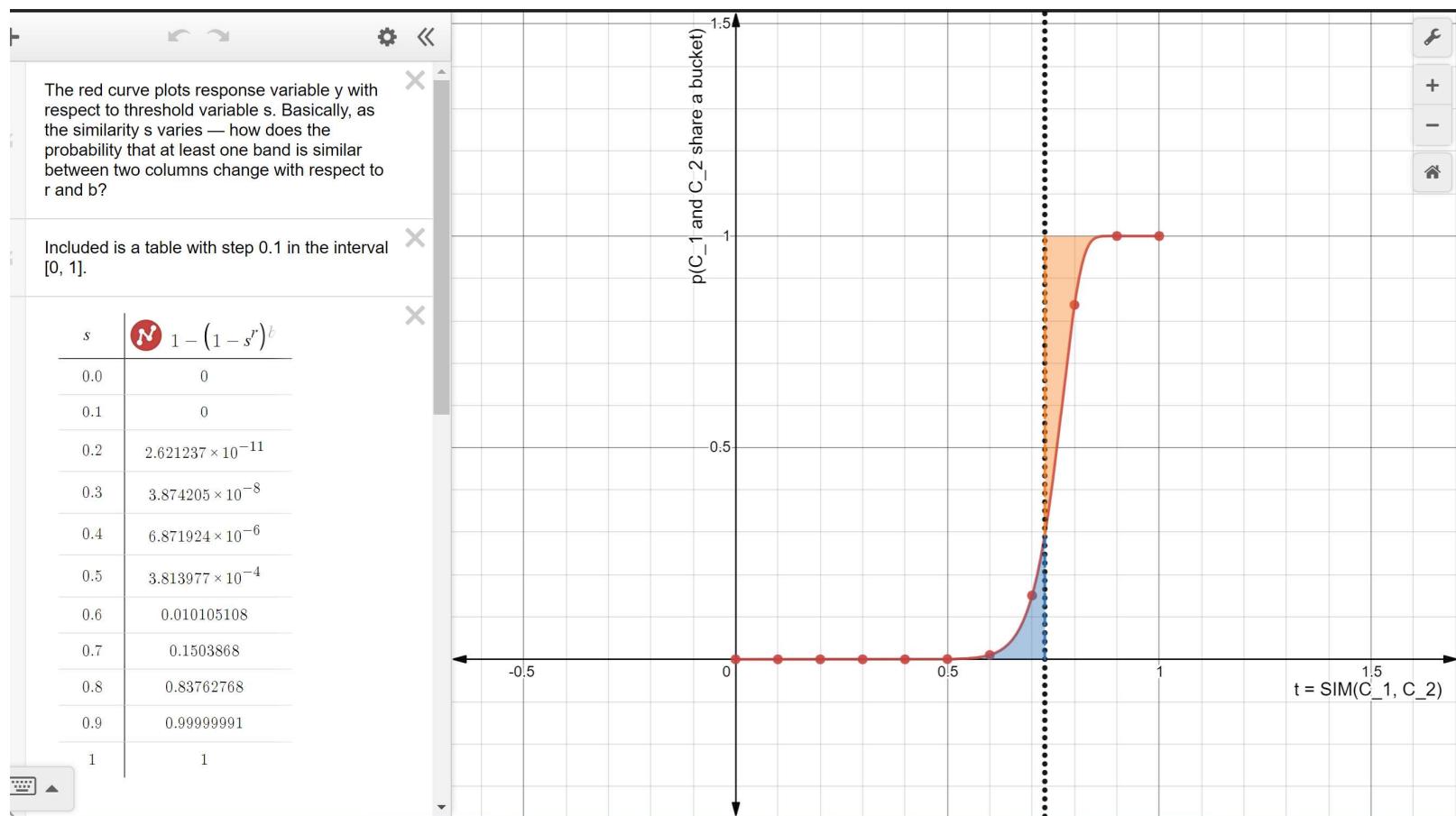
- Aumentare r (AND) rende le collisioni più rare, ma può ridurre eccessivamente p_1 .
- Aumentare b (OR) migliora p_1 , ma aumenta p_2 .

Formula di Sintesi

Con b e r ottimali: $p_1 \rightarrow 1, p_2 \rightarrow 0$
--

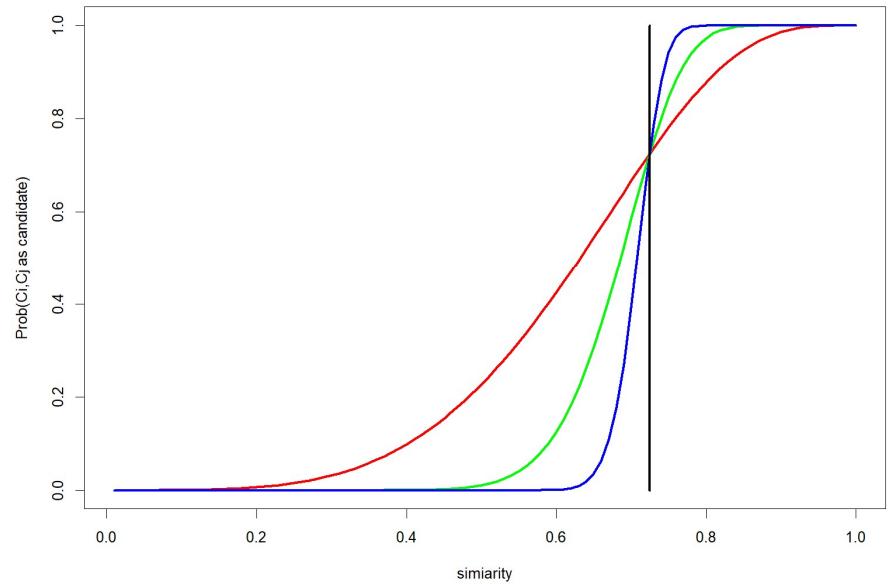
Demo

- <https://www.desmos.com/calculator/lzzvfjiujn>



- For each AND-OR S-curve $1-(1-s)^r$, there is a *threshold* t , for which $1-(1-t^r)^b = t$
- Above t , high probabilities are increased; below t , low probabilities are decreased
- You improve the sensitivity as long as the low probability is less than t , and the high probability is greater than t
- Iterate as you like (computation is not an issue) *Coleado simple*

- RED curve AND-OR
- GREEN curve AND-OR + AND-OR
- BLUE curve AND-OR + AND-OR + AND-OR
- Threshold 0.73



- Similar observation for the OR-AND type of S-curve: $(1-(1-s)^b)^r$

La s-curve si riferisce alla forma della funzione che descrive la probabilità di collisione dopo aver applicato una combinazione di AND-construction e OR-construction

$$1-(1-s^r)^b$$

s probabilità iniziale

r numero di funzioni usate nell'AND-construction

b numero di funzioni usate nell'OR-construction

Questa funzione rappresenta la probabilità di collisione dopo aver applicato:

Un'AND-construction con r funzioni, che porta la probabilità da s a s^r .

Un'OR-construction con b funzioni sull'output dell'AND-construction, che porta la probabilità a $1-(1-s^r)^b$

Esiste un valore di soglia threshold tale che :

$$1-(1-t^r)^b=t$$

t è un punto fisso della funzione $1-(1-t^r)^b$

if $s==t$

 return t

else

 dividi il comportamento della S-curve in due regioni

Per trovare t dobbiamo risolvere l'equazione portando tutto su un lato troverem un'equazione non lineare che può essere risolta numericamente per valori specifici di r e b

EFFETTO del THRESHOLD

IF $s>t$ (probabilità "alte"):

La funzione $1-(1-s^r)^b$ aumenta la probabilità, cioè $1-(1-s^r)^b > s$

Questo significa che le probabilità già alte vengono ulteriori aumentate.

ELSE IF $s<t$:

La funzione $1-(1-s^r)^b$ diminuisce la probabilità, cioè $1-(1-s^r)^b < s$

Probabilità basse vengono ancora diminuite

Perche' succede questo?

Funzione $f - (1-s^r)^b$ ha la forma sigmoidale.

Per s piccolo, s^r è quasi più piccolo, quindi $1-s^r$ è vicino a 1, e $(1-s^r)^b$ relativamente grande, rendendo $1-(1-s^r)^b$ piccolo.

Per s grande, s^r più grande, $1-s^r$ piccolo, $(1-s^r)^b$ molto piccolo, $1-(1-s^r)^b$ vicino a 1.

Il punto in cui la funzione perde del suo comportamento diumente e del comportamento cumulante. Questo è il motivo per cui le S-curva è utile: amplifica le diff. tra prob. alte e basse.

S-curva amplifica le differenze tra P_1 e P_2 , rendendo la funzione più selettiva

Distance Measures

Until now: Jaccard Distance

Distanza che misurano quanto sono vicini due insiemi

- We talked about **Jaccard Distance** and **Hamming**
 - measures how close are two sets
- There are other relevant functions to measure the closeness (distance) *Altre funzioni!*
- Lets recall some properties of distance metrics:
 - $d(x, y) \geq 0$ (no negative distances are allowed)
 - $d(x, y) = 0 \Leftrightarrow x = y$ (distances are positive, except a point from itself)
 - $d(x, y) = d(y, x)$ (distances are symmetric)
 - $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality)

Hamming Distance

- Used with spaces of any vector
- It is computed as the number of components in which two vectors differ.
- **Example:**
 - $x = 10101$
 - $y = 11110$
 - $d(x,y) = 3$

La distanza di hamming può essere applicata a vettori o stringhe di qualsiasi tipo purchè abbiano la stessa lunghezza.
Devono avere stesso numero di componenti

Distanza di Hamming = calcola quante posizioni sono diverse tra i due vettori

Hamming Distance

- Building an LSH family of function for **Hamming Distance** is quite simple
- Let x and y be two **d-dimensional vectors**
 - ...we denote with $h(x, y)$ their hamming distance
- We define a function $f_i(x)$ to be the *i-th* position of vector x
 - $f_i(x) = f_i(y)$ if and only if x and y agree in the *i-th* position.
 - the **probability of $f(x) = f(y)$** can be computed as $1 - h(x, y)/d$
- The **family F** consisting of the functions previously defined is
 $(d_1, d_2, 1 - d_1/d, 1 - d_2/d) - sensitive$
- with $d_1 < d_2$

Famiglia di funzioni LSH = famiglia di funzioni che permettono di trovare punti vicini in modo efficiente, anche in spazi di alta dimensionalità. X

$$\begin{aligned} x &= \underline{\text{10010}} \\ y &= \underline{\text{01010}} \end{aligned}$$
$$f(x) = f(y) \Rightarrow 1 - \frac{h(x, y)}{d}$$
$$P[f(x) = f(y)] = 1 - \frac{d}{q} \approx 1 \text{ if } f(x) = f(y)$$
$$P[f(x) \neq f(y)] = 1 - \frac{d}{q} = 0 \text{ if } f(x) \neq f(y)$$

$$P[f(x) = f(y)] = 1 - \frac{h(x, y)}{d}$$

$$P_1 = x e y \text{ vicini} \Rightarrow h(x, y) \leq d_1$$

↳ funzione decrescente di $h(x, y)$

$$\text{Vicini vicini} = h(x, y) \leq d_2 \\ \text{dato valore}$$

↳ Valore minimo? $\Rightarrow h(x, y) = d_2 \Rightarrow$

$$P[f(x) = f(y)] = 1 - \frac{d_2}{d}$$

Esempio pratico di LSH con distanza di Hamming

Passo 1: Definizione dei vettori e parametri

Consideriamo vettori binari di lunghezza $d = 6$:

- $\mathbf{x} = 101101$
- $\mathbf{y} = 101110$ (vicino a \mathbf{x})
- $\mathbf{z} = 010010$ (lontano da \mathbf{x})

Calcoliamo le distanze di Hamming:

$$h(\mathbf{x}, \mathbf{y}) = \text{numero di posizioni discordanti} = 2 \quad (\text{posizioni 5 e 6})$$

$$h(\mathbf{x}, \mathbf{z}) = \text{numero di posizioni discordanti} = 5 \quad (\text{posizioni 1,2,3,4,6})$$

Soglie di similarità:

$$d_1 = 2 \quad (\text{vicini}), \quad d_2 = 4 \quad (\text{lontani}).$$

Passo 2: Calcolo delle probabilità p_1 e p_2

Le probabilità di collisione sono definite come:

$$\begin{aligned} p_1 &= 1 - \frac{d_1}{d} = 1 - \frac{2}{6} = \frac{2}{3} \approx 0.67 \\ p_2 &= 1 - \frac{d_2}{d} = 1 - \frac{4}{6} = \frac{1}{3} \approx 0.33 \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{bound}$$

Passo 3: Verifica con i vettori

Caso 1: \mathbf{x} e \mathbf{y} (vicini)

$$\text{Probabilità di collisione} = 1 - \frac{h(\mathbf{x}, \mathbf{y})}{d} = 1 - \frac{2}{6} = \frac{2}{3} \approx 0.67 \quad (\geq p_1)$$

Verifica manuale: Concordano in 4 posizioni su 6 $\rightarrow \frac{4}{6} = \frac{2}{3}$.

$$h(\mathbf{x}, \mathbf{y}) \leq 2 \Rightarrow P[f(\mathbf{x}) = f(\mathbf{y})] \geq 0.67$$

Caso 2: \mathbf{x} e \mathbf{z} (lontani)

$$\text{Probabilità di collisione} = 1 - \frac{h(\mathbf{x}, \mathbf{z})}{d} = 1 - \frac{5}{6} = \frac{1}{6} \approx 0.167 \quad (\leq p_2)$$

Verifica manuale: Concordano in 1 posizione su 6 $\rightarrow \frac{1}{6}$.

Passo 4: Interpretazione

La famiglia LSH è (d_1, d_2, p_1, p_2) -sensibile con:

$$d_1 = 2, \quad d_2 = 4, \quad p_1 = \frac{2}{3}, \quad p_2 = \frac{1}{3}.$$

- Vettori vicini ($h \leq d_1$) hanno probabilità $\geq p_1$ di collisione.
- Vettori lontani ($h \geq d_2$) hanno probabilità $\leq p_2$ di collisione.

Cosine Distance

n finiti di dimensioni

- It is a measure defined in **Euclidean spaces** and **discrete Euclidean spaces**.
- It is the cosine of the **angle** that the vectors associated to those points make.
- $d(A, B) = \theta = \arccos(A \cdot B / \|A\| \cdot \|B\|)$
- Has range $[0, \pi]$
- The angle is always in the **range 0 to 180 degrees**.

$$d([x_1, \dots, x_n], [y_1, \dots, y_n]) = 1 - \frac{A \cdot B}{\|A\| \|B\|}$$
$$A \cdot B = \|A\| \|B\| \cos \theta$$

Distanza del coseno si basa sul coseno dell'angolo tra due vettori

Cosine Distance

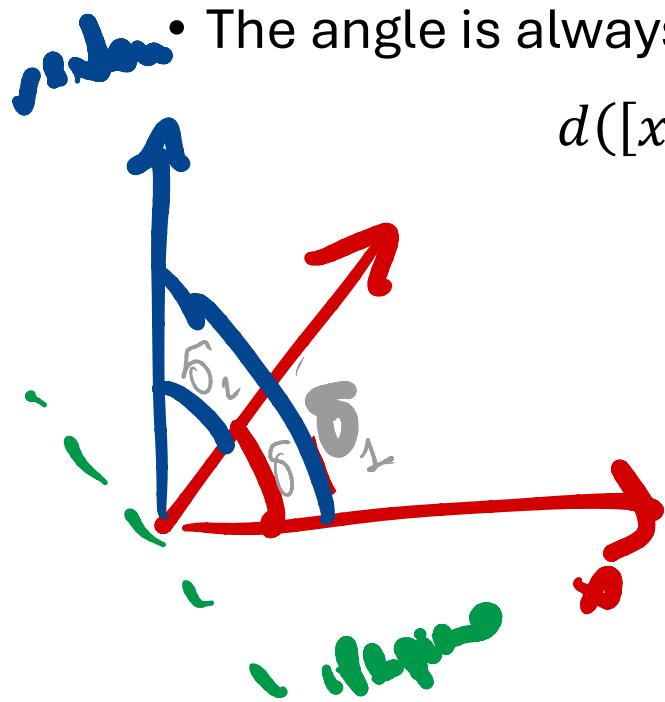
- It is a measure defined in **Euclidean spaces** and **discrete Euclidean spaces**.
- It is the cosine of the **angle** that the vectors associated to those points make.
- $d(A, B) = \theta = \arccos(A \cdot B / \|A\| \cdot \|B\|)$
- Has range $[0, \pi]$
- The angle is always in the **range 0 to 180 degrees**.

$$d(A, B) = \theta = \arccos(A \cdot B / \|A\| \cdot \|B\|)$$

Spost ad alte dimensioni

- **Esempio:** Supponiamo di avere due vettori $A = [1, 0]$ e $B = [0, 1]$ in 2D:
 - $A \cdot B = 1 \cdot 0 + 0 \cdot 1 = 0$
 - $\|A\| = \sqrt{1^2 + 0^2} = 1, \|B\| = \sqrt{0^2 + 1^2} = 1$
 - $\cos(\theta) = A \cdot B / \|A\| \|B\| = 0 / (1 \cdot 1) = 0$
 - $\theta = \arccos(0) = \pi/2$ (90 gradi), il che ha senso perché i vettori sono perpendicolari.

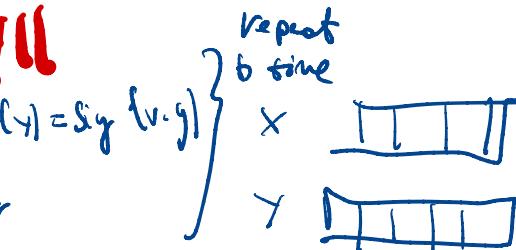
$$d([x_1, \dots, x_n], [y_1, \dots, y_n]) = 1 - \frac{A \cdot B}{\|A\| \|B\|}$$
$$A \cdot B = \|A\| \|B\| \cos \theta$$



$$\cos(\theta) = \frac{x \cdot y}{\|x\| \|y\|} = \sin(x \cdot y)$$

$$f(x) = \text{sgn}(v \cdot x)$$

$v = \text{random vector } v_1, \dots, v_r$

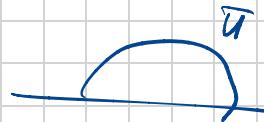


Ans

$$(d_1, d_2, \frac{180 - d_2}{180}, \frac{180 - d_2}{180})$$

if $d(x, y) \leq d_1 \rightarrow P(f(x) = f(y)) \geq$

$$d(x, y) \geq d_2 \quad P(f(x) = f(y)) \leq \frac{180 - d_2}{180}$$



Cosine Distance

- What is Cosine Distance between x and y ?
 - the cosine of the angle between them
- For cosine distance, there is a technique
 - called Random Hyperplanes
 - Technique similar to Min-Hashing
- Lets pick a random hyperplane and its normal vector
 - ...then compute the dot products $v \cdot x$ and $v \cdot y$
 - two cases: $\text{sign}(v \cdot x) = \text{sign}(v \cdot y)$ or $\text{sign}(v \cdot x) \neq \text{sign}(v \cdot y)$

La distanza del coseno misura la dissimilarità tra due vettori in uno spazio euclideo, basandosi sull'angolo tra di loro.

Per la distanza del coseno c'è una tecnica chiamata iperpiani casuali.

Tecnica è un metodo di LSH che serve per approssimare la distanza del coseno tra vettori in modo efficiente, specialmente in spazi ad alta dimensionalità.

Un iperpiano è una generalizzazione di un piano in spazi di dimensione superiore. Un iperpiano divide lo spazio in due semispazi.

Gli iperpiani casuali sono usati per proiettare i vettori in modo da semplificare il calcolo della loro somiglianza!
calcolo di $c_{\text{os}} \theta$ può essere computazionalmente costoso in spazi ad alta dimensionalità.

Tecnica degli iperpiani casuali simile al Min-hashing!

Entrambi i metodi cercano di ridurre la complessità computazione del confronto tra oggetti mappandoli in uno spazio più semplice.

- Min-hashing

Funzioni hash per mappare insiemi in valori più piccoli

Probabilità che due insiemi abbiano lo stesso valore hash è proporzionale alla loro somiglianza.

- Iperpiani casuali

Invece di calcolare direttamente la distanza del coseno si usano iperpiani per classificare i vettori in base al loro orientamento rispetto a un iperpiano casuale.

La probabilità che due vettori finiscano nello stesso semispazio è legata all'angolo θ tra di loro.

Cosine Distance

- We define a function $f(x)$ by choosing a random vector v_f
 - $f(x) = f(y)$ if and only $\text{sign}(v_f \cdot x) = \text{sign}(v_f \cdot y)$.
- The **family F** consisting of the functions previously defined is
 - ($d_1, d_2, (180 - d_1)/180, (180 - d_2)/180$)
– *sensitive*
- with $d_1 < d_2$

- Each vector v determines a hash function h_v with two buckets
 - $h_v(x) = +1 \text{ if } v \cdot x \geq 0$
 - $h_v(x) = -1 \text{ if } v \cdot x < 0$
- LS-family H = set of all functions derived from any vector
- **Claim:**
 - For points x and y , $\Pr[h(x) = h(y)] = 1 - d(x,y) / \pi$

Distanza coseno

Scegliamo un iperpiano casuale e il suo vettore normale. Un iperpiano in uno spazio n -dimensionale è definito da un vettore normale \mathbf{v} e da un'equazione del tipo:

$$\mathbf{v} \cdot \mathbf{x} = 0, \quad (1)$$

dove \mathbf{v} è il vettore normale all'iperpiano e \mathbf{x} è un punto qualsiasi sull'iperpiano. Il vettore normale \mathbf{v} è scelto casualmente, spesso campionando le sue componenti da una distribuzione normale (ad esempio, una distribuzione gaussiana $\mathcal{N}(0, 1)$).

Significato

L'iperpiano divide lo spazio in due semispazi:

- **Semispazio positivo:** dove

$$\mathbf{v} \cdot \mathbf{x} > 0.$$

- **Semispazio negativo:** dove

$$\mathbf{v} \cdot \mathbf{x} < 0.$$

Una volta scelto l'iperpiano con il suo vettore normale \mathbf{v} , calcoliamo il prodotto scalare tra \mathbf{v} e i due vettori \mathbf{x} e \mathbf{y} :

- $\mathbf{v} \cdot \mathbf{x}$: il prodotto scalare tra \mathbf{v} e \mathbf{x} .
- $\mathbf{v} \cdot \mathbf{y}$: il prodotto scalare tra \mathbf{v} e \mathbf{y} .

Il segno del prodotto scalare determina da quale lato dell'iperpiano si trova ciascun vettore:

- Se $\mathbf{v} \cdot \mathbf{x} > 0$, il vettore \mathbf{x} si trova nel semispazio positivo.
- Se $\mathbf{v} \cdot \mathbf{x} < 0$, il vettore \mathbf{x} si trova nel semispazio negativo.

Lo stesso vale per \mathbf{y} .

Casi possibili

Dopo aver calcolato i prodotti scalari, confrontiamo i loro segni:

- **Caso 1:** $\text{sign}(\mathbf{v} \cdot \mathbf{x}) = \text{sign}(\mathbf{v} \cdot \mathbf{y})$:

I vettori \mathbf{x} e \mathbf{y} si trovano dallo stesso lato dell'iperpiano (entrambi nel semispazio positivo o entrambi in quello negativo). Questo suggerisce che i vettori sono "simili", perché l'angolo tra di loro è probabilmente piccolo.

- **Caso 2:** $\text{sign}(\mathbf{v} \cdot \mathbf{x}) \neq \text{sign}(\mathbf{v} \cdot \mathbf{y})$:

I vettori \mathbf{x} e \mathbf{y} si trovano su lati opposti dell'iperpiano. Questo suggerisce che i vettori sono "dissimili", perché l'angolo tra di loro è probabilmente grande.

Collegamento con la distanza del coseno

La probabilità che $\text{sign}(\mathbf{v} \cdot \mathbf{x}) = \text{sign}(\mathbf{v} \cdot \mathbf{y})$ è direttamente legata all'angolo θ tra \mathbf{x} e \mathbf{y} . Più precisamente, questa probabilità è data da:

$$P[\text{sign}(\mathbf{v} \cdot \mathbf{x}) = \text{sign}(\mathbf{v} \cdot \mathbf{y})] = 1 - \frac{\theta}{\pi},$$

dove θ è espresso in radianti.

Definiamo una funzione $f(x)$ che dipende da un vettore casuale \mathbf{v} . Questo vettore \mathbf{v} è il vettore normale dell'iperpiano casuale, scelto campionando le sue componenti da una distribuzione (ad esempio, una distribuzione normale).

La funzione $f(x)$ è definita come segue:

$$f(x) = \begin{cases} 1 & \text{se } \mathbf{v} \cdot \mathbf{x} \geq 0 \quad (\text{semispazio positivo}), \\ 0 & \text{se } \mathbf{v} \cdot \mathbf{x} < 0 \quad (\text{semispazio negativo}). \end{cases}$$

Questa funzione $f(x)$ è una funzione di hash che mappa i vettori in due "bucket" (0 o 1), in base al lato dell'iperpiano in cui si trovano. Se $f(x) = f(y)$, i vettori sono simili; se $f(x) \neq f(y)$, sono dissimili.

La "famiglia \mathcal{F} " si riferisce all'insieme di tutte le possibili funzioni f che possiamo generare scegliendo diversi vettori casuali \mathbf{v} . Ogni \mathbf{v} definisce un iperpiano diverso, e quindi una diversa funzione di hash f .

In Locality-Sensitive Hashing (LSH), usiamo una famiglia di funzioni di hash (qui, la famiglia \mathcal{F}) per mappare i vettori in modo che i vettori simili abbiano una maggiore probabilità di essere mappati nello stesso bucket.

Proprietà di sensitività della famiglia \mathcal{F}

Una famiglia di funzioni di hash è detta (d_1, d_2, p_1, p_2) -sensibile se:

- Se la distanza tra due vettori \mathbf{x} e \mathbf{y} è minore o uguale a d_1 , allora la probabilità che $f(x) = f(y)$ è almeno p_1 .
- Se la distanza tra due vettori \mathbf{x} e \mathbf{y} è maggiore o uguale a d_2 , allora la probabilità che $f(x) = f(y)$ è al massimo p_2 .

In questo caso:

- d_1 e d_2 sono due soglie di distanza (in gradi, perché stiamo parlando dell'angolo θ).
- $p_1 = \frac{180-d_1}{180}$,
- $p_2 = \frac{180-d_2}{180}$.

Questo ci dice che la probabilità che due vettori abbiano lo stesso valore di hash (cioè, $f(x) = f(y)$) dipende dall'angolo θ tra di loro. Più precisamente:

- Se $\theta = d_1$ (un angolo piccolo), la probabilità che $f(x) = f(y)$ è alta: $\frac{180-d_1}{180}$.
- Se $\theta = d_2$ (un angolo grande), la probabilità che $f(x) = f(y)$ è bassa: $\frac{180-d_2}{180}$.

La condizione $d_1 < d_2$ significa che d_1 è una soglia di distanza "piccola" (vettori simili), mentre d_2 è una soglia di distanza "grande" (vettori dissimili). Questo è necessario per garantire che la famiglia \mathcal{F} sia utile per distinguere tra vettori simili e dissimili.

Esempio numerico

Supponiamo $d_1 = 30^\circ$ e $d_2 = 90^\circ$.

- Se $\theta = 30^\circ$, la probabilità che $f(x) = f(y)$ è:

$$P[f(x) = f(y)] = \frac{180 - 30}{180} = \frac{150}{180} = \frac{5}{6} \approx 0.833.$$

- Se $\theta = 90^\circ$, la probabilità che $f(x) = f(y)$ è:

$$P[f(x) = f(y)] = \frac{180 - 90}{180} = \frac{90}{180} = \frac{1}{2} = 0.5.$$

Definizione della funzione di hash e probabilità

Ogni vettore \mathbf{v} (che rappresenta il vettore normale di un iperpiano casuale) definisce una funzione di hash $h_{\mathbf{v}}$ con due bucket (contenitori). La funzione $h_{\mathbf{v}}$ mappa un vettore \mathbf{x} in uno di due bucket in base al suo orientamento rispetto all'iperpiano definito da \mathbf{v} , ed è definita come segue:

- $h_{\mathbf{v}}(\mathbf{x}) = +1$ se $\mathbf{v} \cdot \mathbf{x} \geq 0$,
- $h_{\mathbf{v}}(\mathbf{x}) = -1$ se $\mathbf{v} \cdot \mathbf{x} < 0$.

In altre parole, la funzione $h_{\mathbf{v}}$ assegna il valore $+1$ se il vettore \mathbf{x} si trova nel semispazio positivo (o sull'iperpiano), e il valore -1 se si trova nel semispazio negativo.

La famiglia Locality-Sensitive (LS-family) H è l'insieme di tutte le possibili funzioni $h_{\mathbf{v}}$ che possiamo generare scegliendo diversi vettori casuali \mathbf{v} . Ogni \mathbf{v} definisce un iperpiano diverso, e quindi una diversa funzione di hash $h_{\mathbf{v}}$.

Un risultato fondamentale è la relazione probabilistica tra la probabilità che due vettori \mathbf{x} e \mathbf{y} finiscano nello stesso bucket e l'angolo θ tra di loro. Sia $d(\mathbf{x}, \mathbf{y}) = \theta$ l'angolo tra \mathbf{x} e \mathbf{y} in radianti, dove $\theta = \arccos\left(\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}\right)$. La probabilità che $h(\mathbf{x}) = h(\mathbf{y})$ per una funzione h scelta casualmente dalla famiglia H è data da:

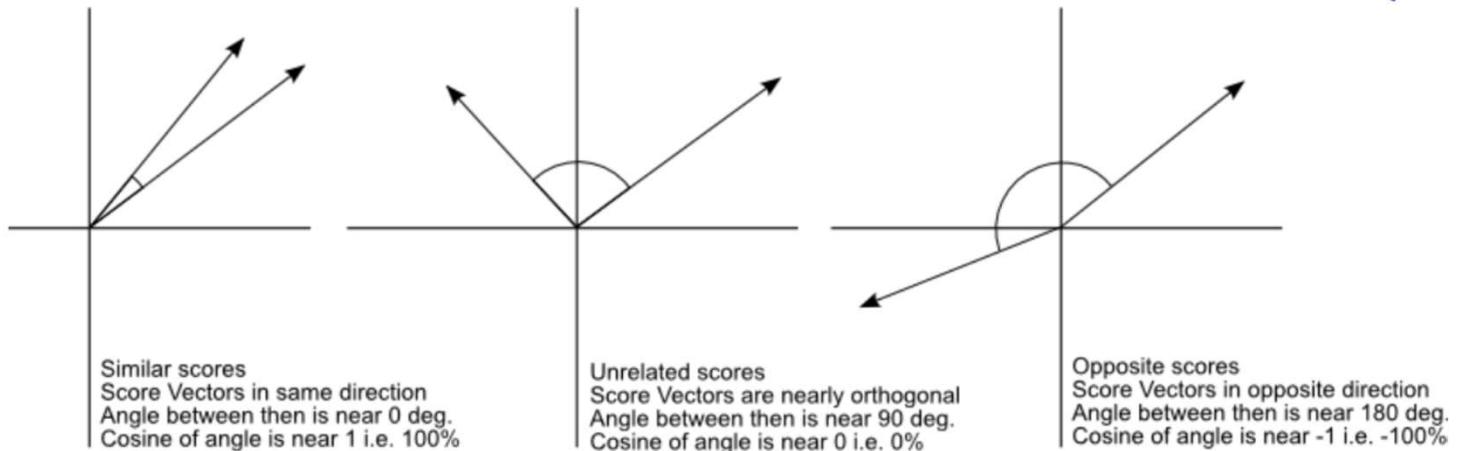
$$\Pr[h(\mathbf{x}) = h(\mathbf{y})] = 1 - \frac{d(\mathbf{x}, \mathbf{y})}{\pi}.$$

Questa relazione ci dice che:

- Se $\theta = 0$ (vettori allineati), la probabilità che $h(\mathbf{x}) = h(\mathbf{y})$ è $1 - \frac{0}{\pi} = 1$, cioè i vettori finiscono sempre nello stesso bucket.
- Se $\theta = \frac{\pi}{2}$ (vettori perpendicolari), la probabilità è $1 - \frac{\pi/2}{\pi} = 1 - \frac{1}{2} = 0.5$.
- Se $\theta = \pi$ (vettori opposti), la probabilità è $1 - \frac{\pi}{\pi} = 0$, cioè i vettori non finiscono mai nello stesso bucket.

Questa proprietà rende la famiglia H particolarmente utile per approssimare la distanza del coseno in modo efficiente, poiché la probabilità che due vettori siano mappati nello stesso bucket è direttamente legata alla loro somiglianza (inversamente proporzionale all'angolo θ).

Random hyperplanes



Consider points x and y

Let's analyze hyperplanes v and v'

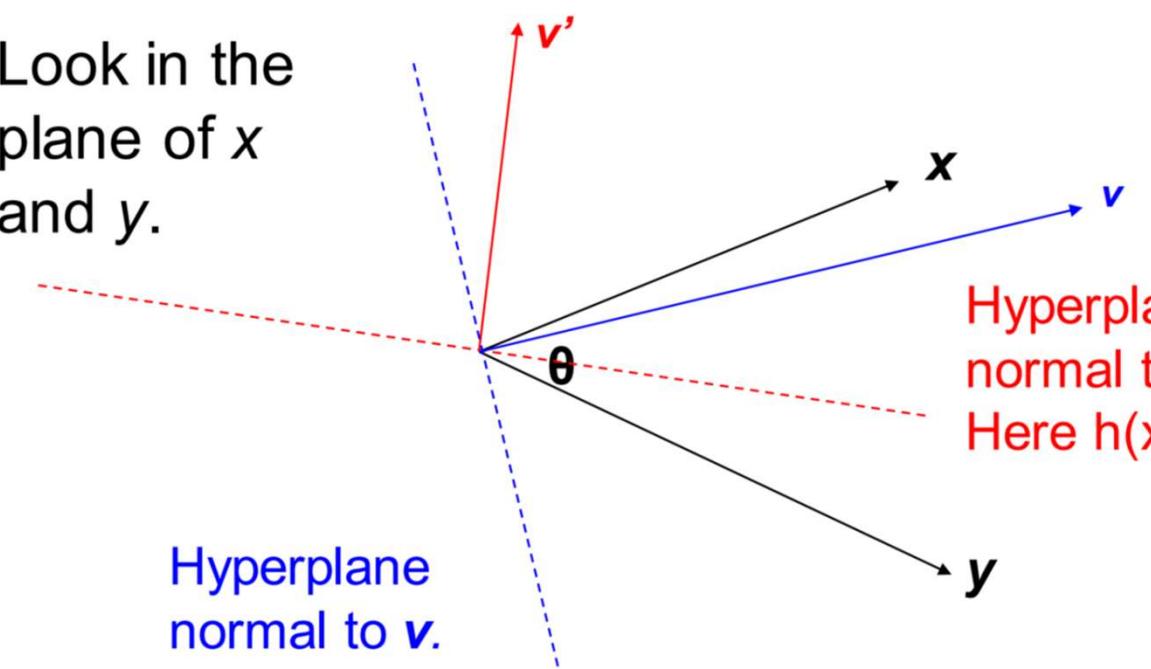
Look in the plane of x and y .

- $h_v(x) = +1$ if $v \cdot x \geq 0$
- $h_v(x) = -1$ if $v \cdot x \leq 0$

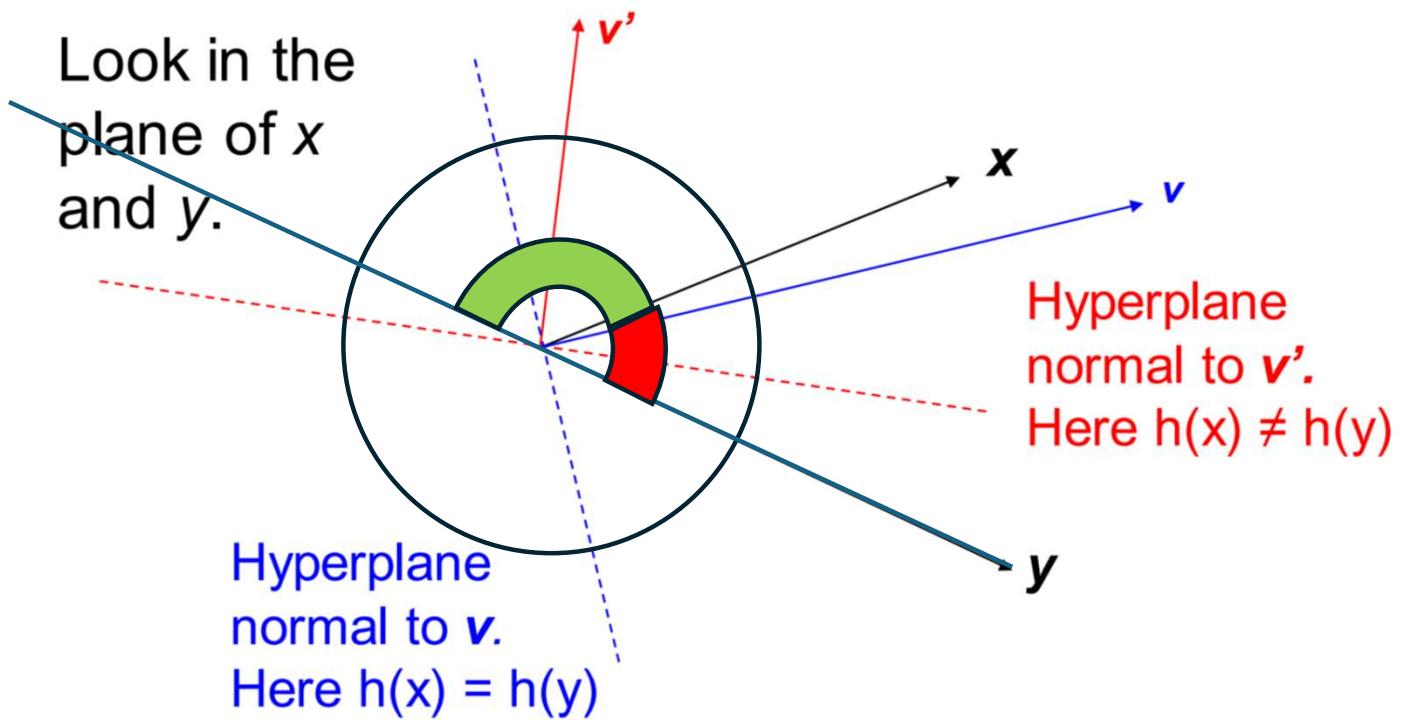
Hyperplane normal to v .
Here $h(x) = h(y)$

θ

Hyperplane normal to v' .
Here $h(x) \neq h(y)$



- Prob. Red case $= \theta / \pi$
- Prob green case $1 - \theta / \pi$



How to build the signature

$$V \rightarrow \{-1, 1\}$$

$$V \times X = -1 \cdot X_1 + 1 + 1 \cdot X_2$$

- Pick some number of random vectors, and hash your data for each vector
- The result is a **signature (*sketch*)** of **+1**'s and **-1**'s for each data point
- Can be used for LSH like we used the Min-Hash signatures for Jaccard distance
- Amplify using **AND/OR** constructions
- We can demonstrate that by restricting our choice to vectors whose component are **+1** and **-1**, we don't lose randomness.

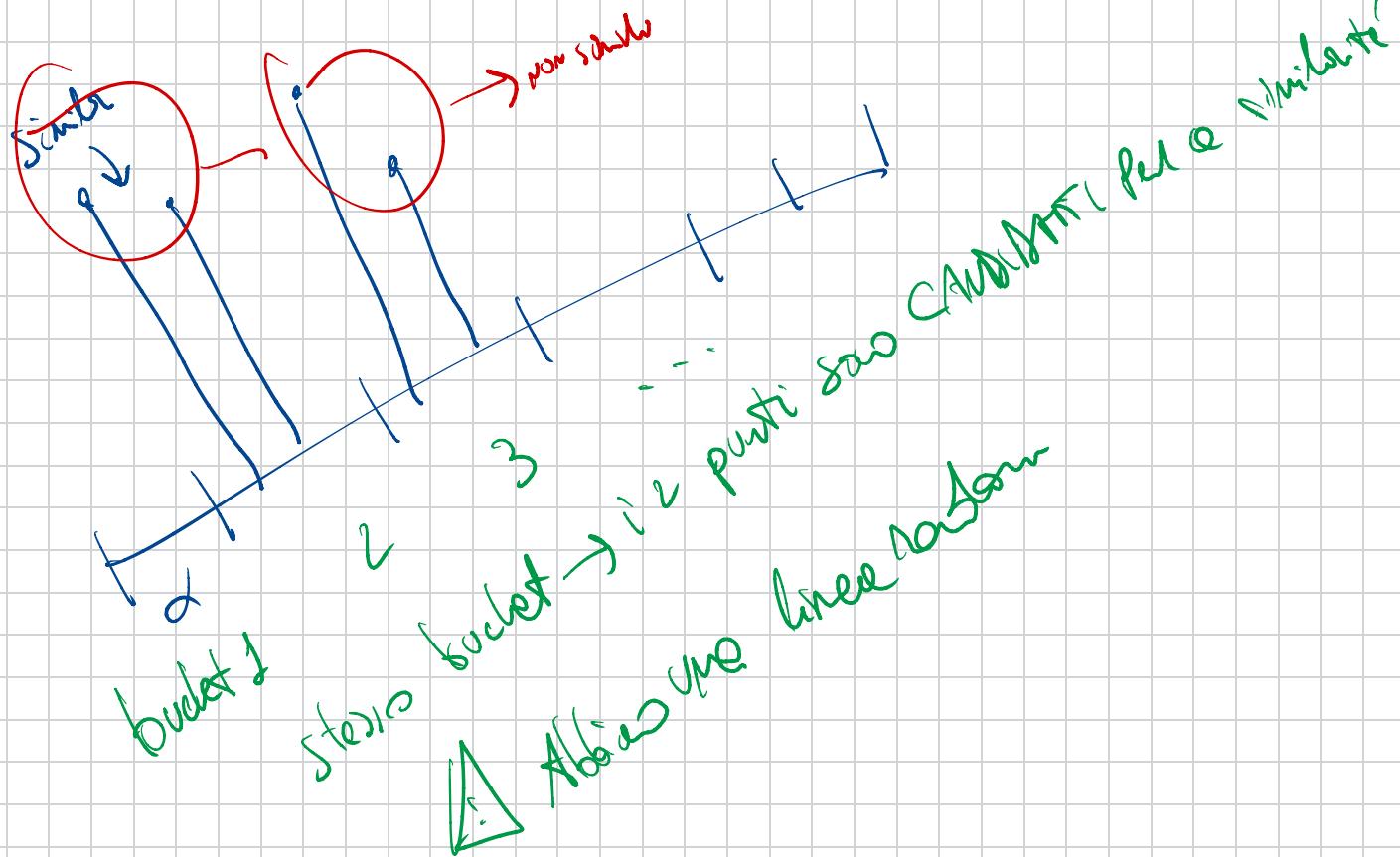
Distance in Euclidean spaces

- n-dimensional Euclidean space:
 - points are vectors of n real numbers.
- For any constant r, an **L_r-norm** is a measure defined as:
- $d([x_1, \dots, x_n], [y_1, \dots, y_n]) = \sqrt[r]{(\sum_{i=1}^n |x_i - y_i|^r)^{1/r}}$
- Some famous norms:
 - L₁-norm – **Manhattan Distance**
 - $d([x_1, \dots, x_n], [y_1, \dots, y_n]) = \sum_{i=1}^n |x_i - y_i|$
 - L₂-norm – **Euclidean Distance**
 - $d([x_1, \dots, x_n], [y_1, \dots, y_n]) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$

Euclidean Distance

- Building an LSH function for Euclidean distance is much more difficult
- 2-dimensional example: lets pick a random horizontal line
 - project each point into such a line then divide it into segments of length a
 - each segment is a bucket where our function $f(x)$ hashes each point.
- We define a family \mathbf{F} by choosing random lines through the space and a bucket size “ a ” that partitions each line.
 - We will hash points by projecting them onto the line $(d_1, d_2, p_1, p_2) - \text{sensitive}$
- We are not able to determine an expression for p_1 and p_2 but we can be certain that $p_1 > p_2$ for each $d_1 < d_2$

Given
✓
 $p_1 > p_2$
not equal



Random Projection for Euclidean Distance

- Let x be a random vector with coordinates selected randomly from a Gaussian Distribution $N(0,1)$
- Let v a query point, $h(v) = x \cdot v$ is the scalar product of the two vectors.
- The scalar projection is then quantized into a set of hash bins with the intention that close items in the original space will fall in the same bin. The hash function is:

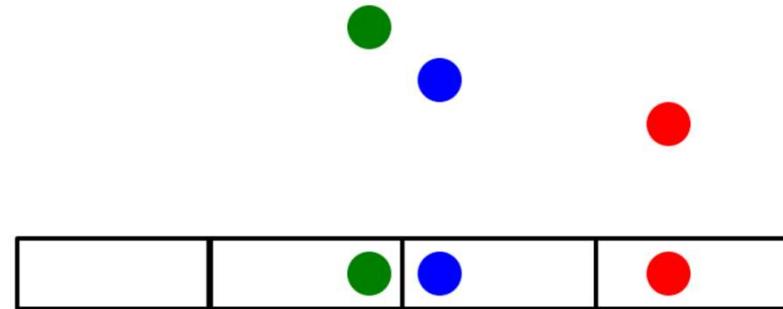
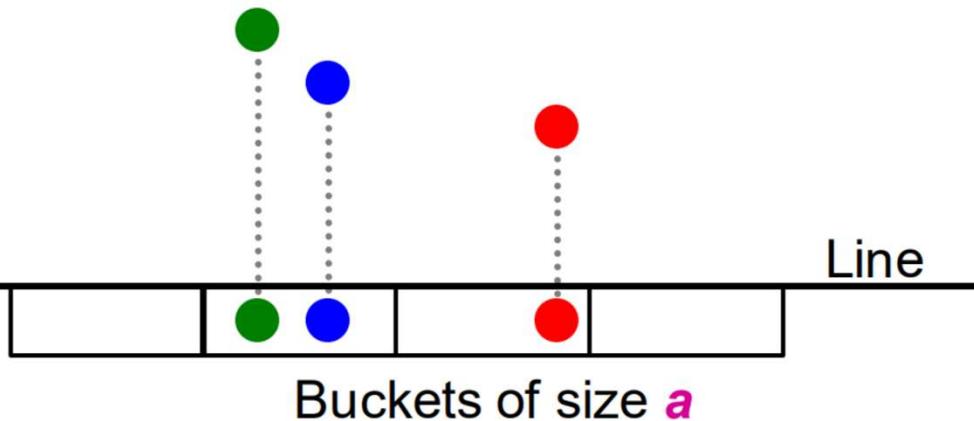
$$h^{x,b}(v) = \left\lfloor \frac{x \cdot v + b}{w} \right\rfloor$$

- w is the width of each quantization bin, and b is a random variable uniformly distributed between 0 and w that makes the quantization error easier to analyze, with no loss in performance.
- for any points p and q in \mathbb{R}^d that are close to each other, there is a high probability p_1 , that they fall into the same bucket

$$P_h[h(p) = h(q)] \geq p_1 \text{ for } \|p - q\| \leq d_1$$

- for any points p and q in \mathbb{R}^d that are far apart, there is a low probability $p_2 < p_1$, that they fall into the same bucket

$$P_h[h(p) = h(q)] \leq p_2 \text{ for } \|p - q\| \geq c * d_1 = d_2$$

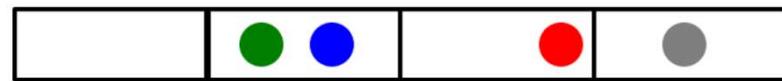
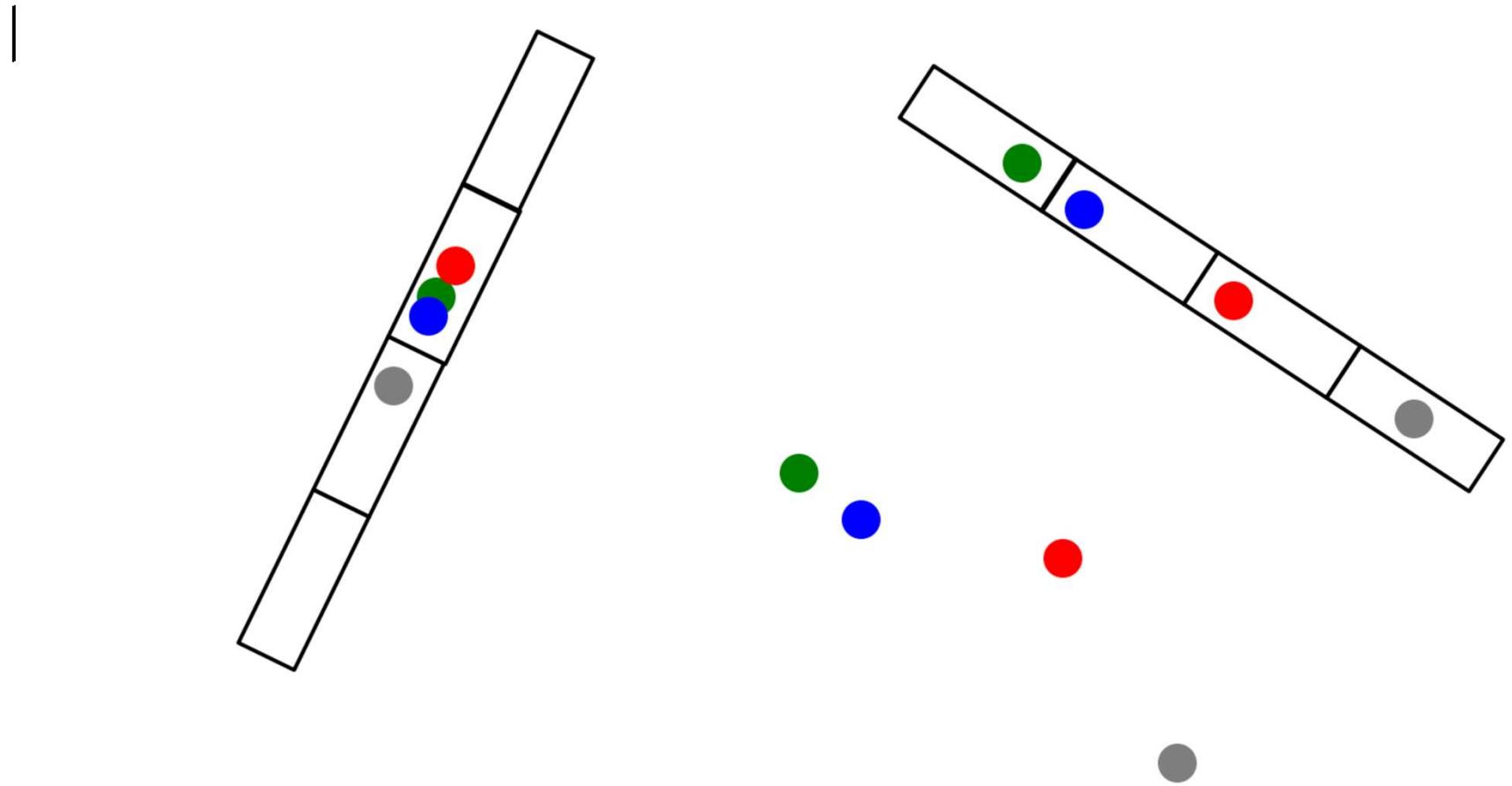


■ “Lucky” case:

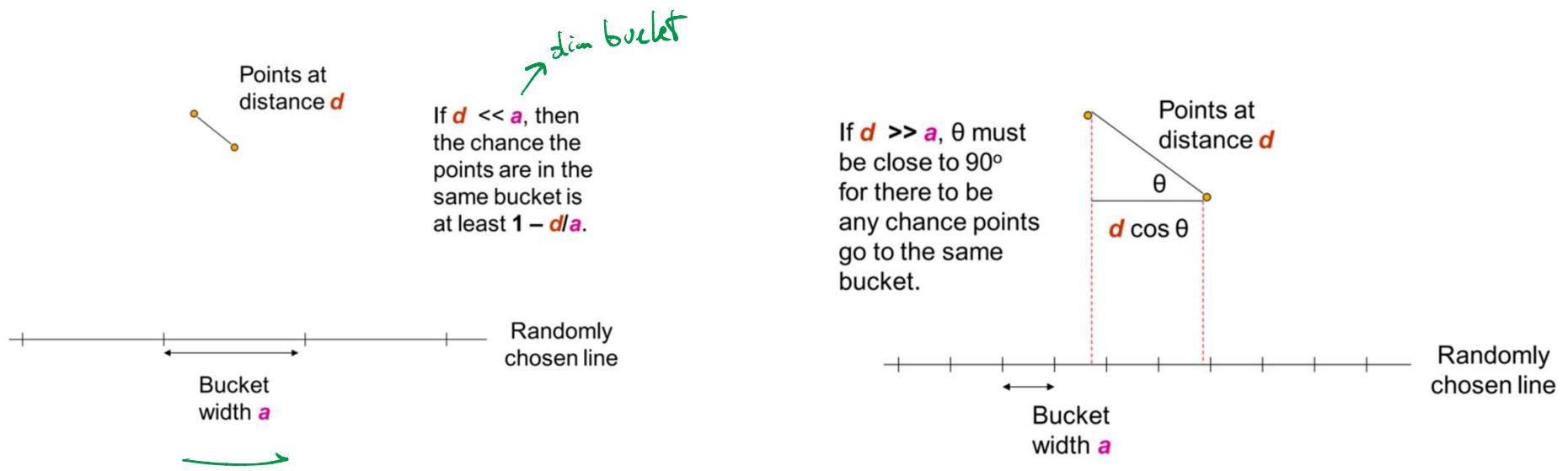
- Points that are close hash in the same bucket
- Distant points end up in different buckets

■ Two “unlucky” cases:

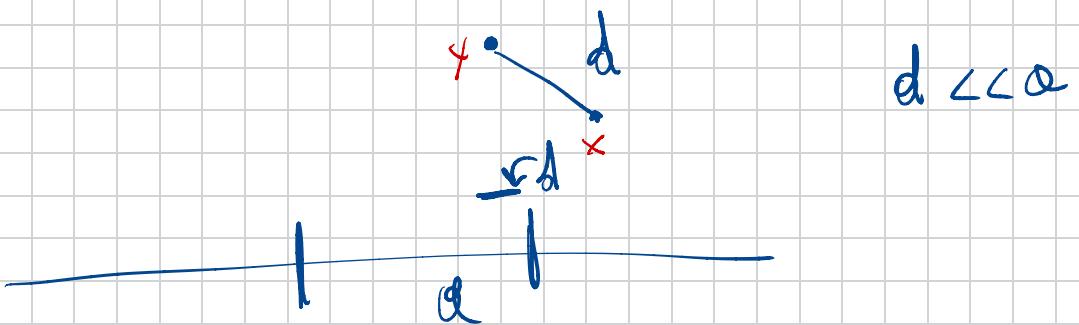
- **Top:** unlucky quantization
- **Bottom:** unlucky projection



Two cases



- If points are distance $d < a/2$, prob. they are in same bucket $\geq 1 - d/a = 1/2$
- If points are distance $d > 2a$ apart, then they can be in the same bucket only if $d \cos \theta \leq a$
- $\cos \theta \leq 1/2$
- $60^\circ < \theta < 90^\circ$, i.e., at most $1/3$ probability
- Yields a $(a/2, 2a, 1/2, 1/3)$ -sensitive family of hash functions for any a

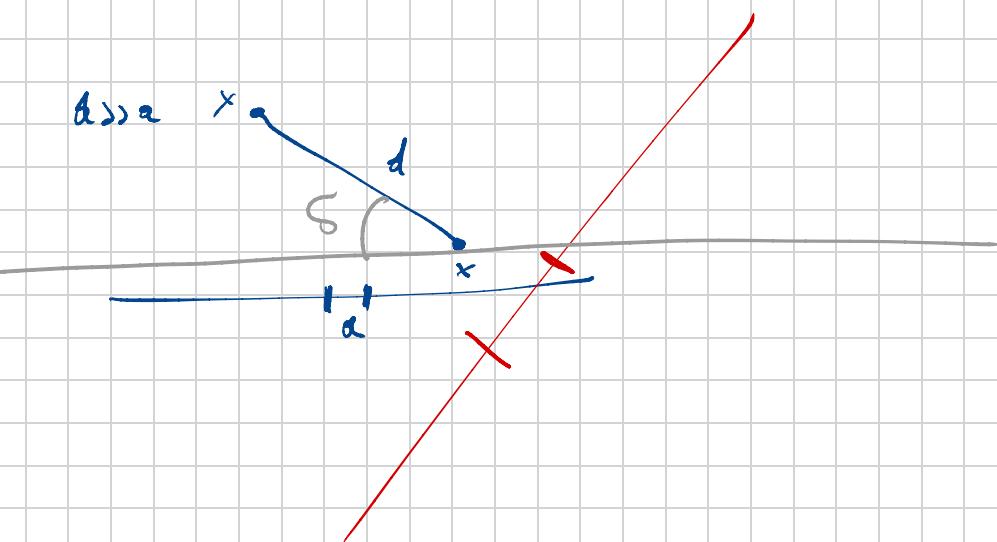


P
bucket

Frontline

Worst case $\rightarrow 1$ in each bucket also all also

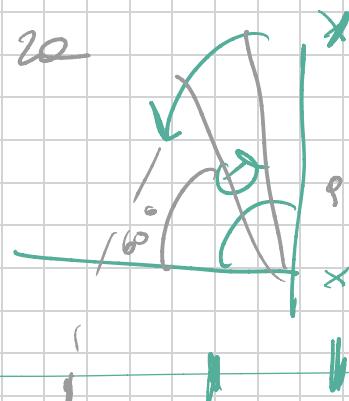
$$P(f(x) = f(y)) = 1 - \frac{d}{\alpha}$$



$$P(f(x) = f(y)) \geq 1 - \frac{\alpha/2}{\alpha} = \frac{1}{2}$$

$$P(f(x) = f(y)) = 1 - \frac{d}{\alpha} \geq 1 - \frac{\alpha/2}{\alpha} = \frac{1}{2}$$

$d > 2\alpha$



90° unlucky case

$\frac{1}{3}$ steps budget

$\frac{2}{3}$ success budget

Edit distance

- Used with **strings**
- The distance between **two strings x and y** is the **smallest** number of **insertions and deletions of single characters** that will convert x to y.
- **Example:**
 - $x = abcde$
 - $y = acfdeg$
 - $d(x,y) = 3$
 - delete «b»
 - insert «f» after «c»
 - insert «g» after «e»

Other applications of LSH

- **Fingerprint matching:** comparing fingerprints
- **Newspaper articles matching:** distinguishing newspaper article from all the extraneous material
- **Entity resolution:** mapping data records that refer to the same entity (i.e. person)

Summary: 3 Steps

- **Shingling:** Convert documents to sets
 - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
 - We used **similarity preserving hashing** to generate signatures with property $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
 - We used hashing to get around generating random permutations
- **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
 - We used hashing to find **candidate pairs** of similarity $\geq s$

Notebook

↳ in pol. Repitition

- Spark Implementation MinHashLSH