

# **Mining the Web**

## **Similarity search**

---

Soumen Chakrabarti

2024SEP-27-

# Similarity search

## Point query:

- Given query  $q$ , find similar documents from corpus of  $N$  docs
- In case of sparse indices,  $q$  had very few words
- Words and documents increasingly indexed as dense vector embeddings
- Now query is  $q \in \mathbb{R}^D$  and so are docs  $d \in \mathbb{R}^D$
- Find the ‘closest’  $K$  docs in  $o(N)$  time

## All-to-all:

- For each ‘query’ doc, find similar docs
- Given  $d_1$  as query, find  $K$  most similar  $d_2$ s
- Do this for all  $N$  query docs in  $o(N^2)$  time, say  $\tilde{O}(N)$  time
- Motivation: de-duplication
- Also useful for images, other multimedia



## Exact hashing

- Input domain to be uniformly spread over  $2^{128}$  buckets
- 
- Output - sensitive to input

$2^{-128}$

## Hash functions (1)

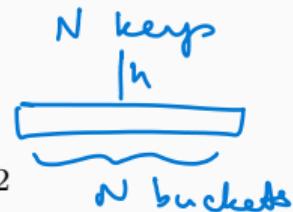
- Basic data structure in information retrieval (to map strings to integers)
- First given the key set, then choose hash function(s) from families of hash functions, then hash keys into buckets
- (May be impossible to ensure low collisions if hash function is chosen first and then adversarial keys presented)
- Generally, you want hash functions to disperse the input  $\mathcal{U}$  completely uniformly over  $[M]$ , minimizing collisions
  - Weakly universal hash family

$$\Pr(h(x_1) = h(x_2)) \leq 1/M \quad \forall x_1 \neq x_2$$

## Hash functions (2)

- Strongly or 2-universal or pairwise independent hash family

$$\Pr(h(x_1) = y_1 \wedge h(x_2) = y_2) = 1/M^2 \quad \forall x_1 \neq x_2, y_1 \neq y_2$$



- $h_{a,b}(x) = ax + b \bmod p$  is weakly universal
- Weakly universal hash functions are adequate for  $O(1)$  expected probe time ...
- ... but not worst case probe time: When  $n$  balls are thrown into  $n$  bins uar, some bin has  $\Omega(\log n / \log \log n)$  wp at least  $1 - 1/n$
- Perfect hash: get  $O(n)$  storage,  $O(n)$  preprocessing time, and  $O(1)$  worst case lookups

## Perfect hashing (1)

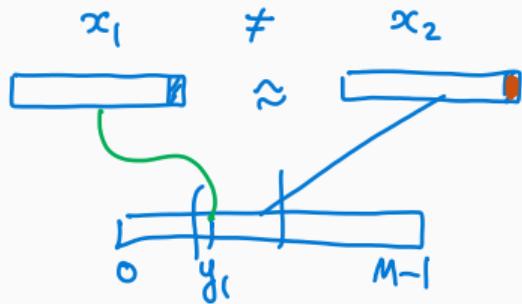
- One level is not enough; need two levels of hashing
- First level: hash function  $f$  from weakly universal family partitions  $S \subset U$ ,  $|S| = n$ , into buckets  $B_0, \dots, B_{n-1}$
- Suppose  $b_i = |B_i|$
- Keep choosing  $f$  until we get one such that  $\sum_i b_i^2 \leq \beta n$
- Can show: if  $\beta \geq 4$ , then the expected number of times we need to sample  $f$  is at most 2
- In the second level, allocate a memory array of size  $\alpha b_i^2$  for bucket  $B_i$

## Perfect hashing (2)

- Keep choosing hash function  $g_i$  from weakly universal family, until there is no collision among keys in  $B_i$
- Note  $b_i$  keys go into  $\alpha b_i^2$  buckets
- Can show: if  $\alpha \geq 2$ , expected work to find good  $g_i$  is  $O(b_i^2)$
- *Expected* time to build the hash table is  $O(n)$ , but *worst-case* time may be larger
- However, storage is  $O(n)$  worst case, and lookups are  $O(1)$  worst case, once the hash table is built

## Tools for analysis

- Linearity of expectation  $E[X] + E[Y] = E[X + Y]$  for random variables  $X, Y \in \mathbb{R}$
- Union bound:  $\Pr(A \cup B) \leq \Pr(A) + \Pr(B)$  for events  $A, B$
- **Markov's inequality:**  $\Pr(X \geq a) \leq E[X]/a$
- **Geometric distribution:** if coin head probability is  $p$ , then probability of  $n$  tosses before first head is  $(1 - p)^{n-1} p$
- Expected number of tosses to get first head is  $1/p$



## Locality sensitive hashing

---

# Locality sensitive hash functions

- Sometimes, you want the hash function to be **locality preserving** — **similar** domain objects should map to similar hash codes
- Wish to support different notions of distance/similarity based on application needs
- **Hamming distance** between bit vectors (**number of disagreeing bits**)
- $L_1$  distance between <sup>real</sup>  $\vec{a}$  vectors:  $\|\vec{a} - \vec{b}\|_1 = \sum_j |a_j - b_j|$
- Cosine similarity between vectors:  $\text{sim}(\vec{a}, \vec{b}) = (\vec{a} \cdot \vec{b}) / (\|\vec{a}\|_2 \|\vec{b}\|_2)$
- Dot-product similarity  $\vec{a} \cdot \vec{b}$
- $L_2$  distance  $\|\vec{a} - \vec{b}\|_2 = \sqrt{\sum_j (a_j - b_j)^2}$
- Jaccard similarity between sets:  $\text{sim}(A, B) = (|A \cap B|) / (|A \cup B|)$ 
  - HW Show that  $1 - J(A, B)$  satisfies triangle inequality

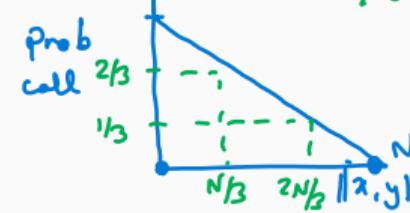
# Hamming distance and bit sampling

- $2^N$  strings  $\{0, 1\}^N \subset [0, N]$   $\frac{\|x, y\|_H}{N} \in [0, 1]$
- Hamming distance  $\|x, y\|_H$  between two strings is the number of disagreeing bits
- Define hash family  $\mathcal{F}$  as  $h_i(x) = x_i$ , the  $i$ th bit of  $x$
- Choosing a bit position  $1 \leq i \leq N$  u.a.r.,  $x \neq y$

$$1 - \Pr_{i \in \mathcal{H}}(h_i(x) = h_i(y)) = \Pr_{i \in \mathcal{H}}(h_i(x) \neq h_i(y)) = \frac{\|x, y\|_H}{N}$$

prob of collision

- Define  $\text{sim}(x, y) = 1 - \frac{\|x, y\|_H}{N} \in [0, 1]$   $\rightarrow$  Prob. of coll
- E.g., if  $\|x, y\|_H \leq N/3$ , then  $\Pr(h(x) = h(y)) \geq 2/3$
- And if  $\|x, y\|_H \geq 2N/3$ , then  $\Pr(h(x) = h(y)) \leq 1/3$



# LSH and range queries for Hamming distance

#

- Family  $\mathcal{F}$  is  $(c, r, P_1, P_2)$ -sensitive if for any two strings  $x, y$

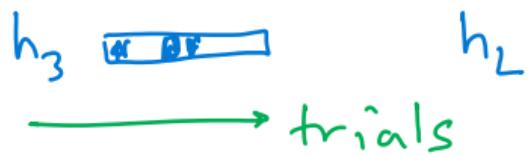
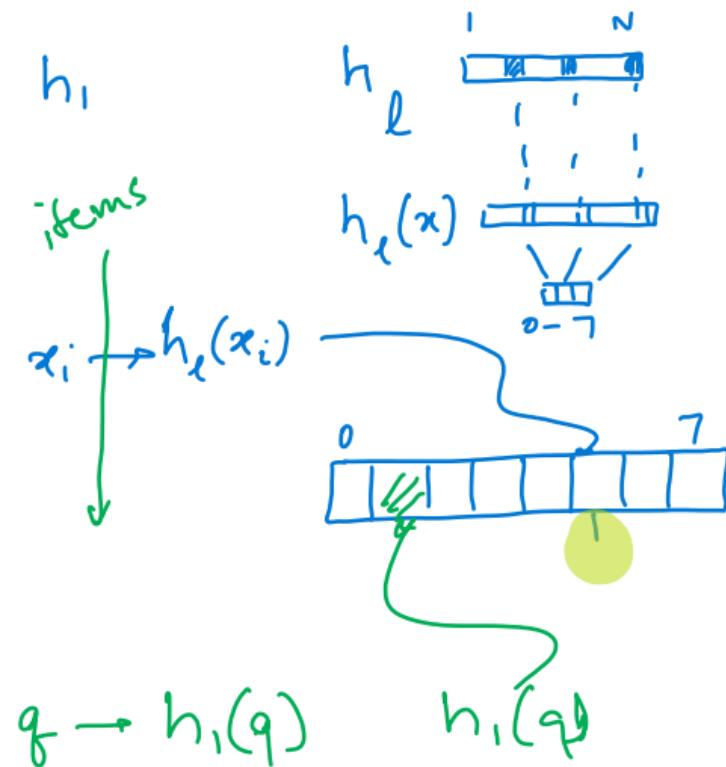
Near

Far

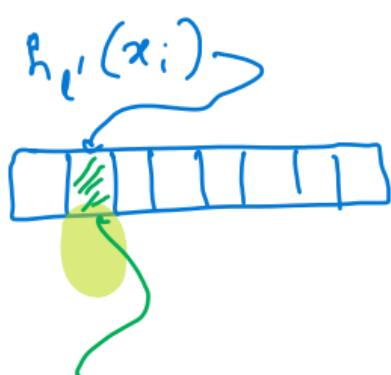
- $\|x, y\|_H \leq r \implies \Pr(h(x) = h(y)) \geq P_1$
- $\|x, y\|_H \geq cr \implies \Pr(h(x) = h(y)) \leq P_2$

- $c > 1, P_1 > P_2$  is the interesting case
- Single bit sampling hash family satisfies  $c = 2, r = N/3, P_1 = 2/3, P_2 = 1/3$
- Or, in general,  $c, r, P_1 = 1 - r/N, P_2 = 1 - cr/N$
- Will *amplify* this separation
- **$c$ -approximate  $r$ -range query:** given query point  $q$ , if there exists at least one near  $p : d(q, p) \leq r$ , then return some  $p'$  with  $d(q, p') \leq cr$

not too far



sampled  
from  $\binom{N}{K}$



$$h_1(x_i)$$



$$h_L(q)$$

$$h_e(g)$$

Fully score  $\bigcup_{l=1}^L \{x \in \text{bucket}[h_e(q)]\}$

# Data structure and preprocessing

- Instead of one bit position, sample  $k$  bit positions
- Given an  $N$ -bit string, look up these  $k$  positions
- Pack into a  $k$ -bit sketch of the original string
- Now do this  $L$  times (independently)
- Thus each of  $n$  strings leads to  $L$  hash values, each  $k$  bits wide
- Call these  $g_1(x), \dots, g_L(x)$ , each  $g_\ell(x) \in \{0, 1\}^k$
- Create a hash table for each  $\ell = 1, \dots, L$
- Each hash table has  $2^k$  slots ~~slots~~ buckets
- Each data item  $x$  is replicated ("pointer" only) to all  $L$  hash tables
- In hash table  $\ell$ , item  $x$  is pushed into slot  $g_\ell(x)$

## Probe step

- Given query  $q$
  - For  $\ell = 1, \dots, L$ 
    - Compute slot number  $g_\ell(q)$
    - Exhaustively check  $\|q, x\|_H$  for all  $x$  in this slot
    - If any  $\|q, x\|_H \leq r$  report  $x$  as a near point
  - At any time if more than  $2L$  items  $x$  have been checked (including duplicates), terminate the search
- $\frac{|S|}{2^k} \quad L \quad \longrightarrow 2L$

The above setup is correct under these (high probability) conditions:

- Query will collide with a near point: If  $\exists x : \|q, x\|_H \leq r$ , then  $g_\ell(q) = g_\ell(x)$  for some  $\ell$
- Not too many far points will collide with query: There are at most  $2L$  items  $x$  such that  $\|q, x\|_H \geq cr$  and yet  $g_\ell(q) = g_\ell(x)$  for some  $\ell$

## Analysis: Few far points collide (1)

$$d(x, y)$$

- Notational convenience:  $H(x, y) = \|x, y\|_H$
- For any  $\ell$  and any  $x$ , if  $H(q, x) \geq cr$  (point is *far* from query), then  
 $\Pr(g_\ell(q) = g_\ell(x)) \leq P_2^k$  — one in corpus size      Prob. (q collides with one far pt.)  $\leq \frac{1}{n}$
- Set  $k = \frac{\log n}{\log(1/P_2)}$ , so that  $P_2^k = 1/n$
- There are at most  $n$  <sup>far</sup> points  $x$  with  $H(q, x) \geq cr$
- Therefore the expected number of far points  $x$  with  $g_\ell(q) = g_\ell(x)$  (for a fixed  $\ell$ ) is at most  $n(1/n) = 1$       collide
- Over all  $L$  tables, the expected number of far points that collide with  $q$  is at most  $L$

## Analysis: Few far points collide (2)

- Markov inequality: for any random variable  $X \geq 0$ ,  $\Pr(X \geq a) \leq E(X)/a$
- By Markov inequality, the number of far, colliding points is at most  $2L$  with probability at least  $1/2$

## Analysis: Near points tend to collide

- For any  $\ell$  and any  $x$ , if  $H(q, x) \leq r$  (*near point*), then  $\Pr(g_\ell(q) = g_\ell(x)) \geq P_1^k$
- For a fixed  $\ell$ , the probability of collision with the query,  $\Pr(g_\ell(q) = g_\ell(x)) \geq P_1^k = P_1^{\frac{\log n}{\log(1/P_2)}} = \exp\left(\frac{\log n}{\log(1/P_2)} \log P_1\right) = \exp\left(-\frac{\log(1/P_1)}{\log(1/P_2)} \log n\right) = n^{-\rho}$   
where  $\rho = \frac{\log(1/P_1)}{\log(1/P_2)}$
- The probability that near point  $x$  collides with  $q$  in at least one of  $L$  tables is  
$$1 - \underbrace{(1 - n^{-\rho})^L}_{\substack{\text{pr. coll in 1 trial} \\ \text{prob. of no coll in 1 trial}}} = \underbrace{\text{pr. coll in}}_{\substack{\text{prob. of no coll in } L \text{ trials}}}$$
- Set  $L = n^\rho$ , then the probability of a near point colliding is at least  $1 - 1/e$

▶ HW

# The ‘master’ LSH

- Hamming LSH can be regarded as the ‘master’ LSH
- For other distance measures, choose a suitable hash family
- Apply sampled hash functions to get the hash code for each item
- The hash code is a bit vector
- If items are similar, hash codes have small Hamming distance
- Now apply the Hamming LSH

$$x \xrightarrow{g \in \mathcal{G}} \{0, 1\}^D$$

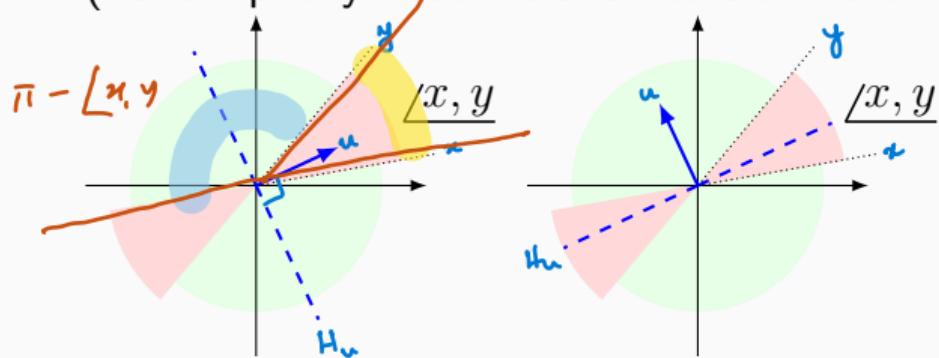
↓ Hamming  
LSH

## From Hamming to $L_1$ distance

- Set  $P$  of  $n$  points in  $\mathbb{R}^d$
- $L_1$  distance used
- Coordinates are positive integers in  $[0, C]$
- Let  $x = (x_1, \dots, x_d)$
- Write down each coordinate in  $C$ -bit unary code
- Gives  $Cd$ -bit representation  $v(x)$  of each point  $x$
- Note that  $L_1$  distance between  $x$  and  $x'$  is the Hamming distance between  $v(x)$  and  $v(x')$

# Cosine similarity (1)

- Documents represented as vectors  $x, y$
- $\cos \angle x, y = \frac{x \cdot y}{\|x\|_2 \|y\|_2}$  is the cosine of the angle between the vectors
- Choose a **unit vector**  $u$  with its arrow lying with uniform density on the unit sphere with center at origin (this characterizes the family  $\mathcal{F}$ )
- (For simplicity visualize two vectors in the 2d plane)



$$\text{sign}(u \cdot x) = \text{sign}(u \cdot y)$$

collision

$$\text{sign}(u \cdot x) \neq \text{sign}(u \cdot y)$$

non-collision

$$h_u(x) = \text{sign}(u \cdot x)$$
$$\in \pm 1$$

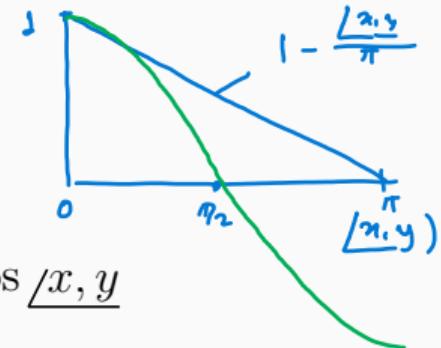


## Cosine similarity (2)

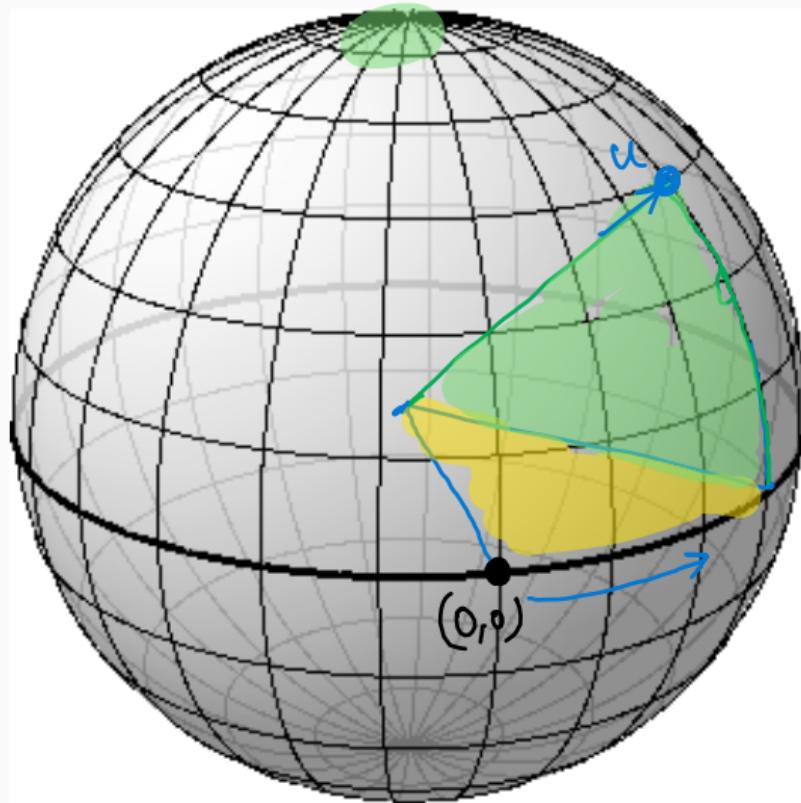
- Hyperplane perpendicular to  $u$  and passing through origin separates  $x, y$  for  $2\underline{x}, \underline{y}$  fraction out of  $2\pi$  rotation of  $u$
  - Define  $h_u(x) = \text{sign}(u \cdot x) \in \pm 1$  (one-bit hash)
  - Easy to see that

$$\Pr_{u \in \mathcal{F}}(h_u(x) = h_u(y)) = 1 - \frac{\langle x, y \rangle_{\text{rank}}}{\pi} = \cos \langle x, y \rangle$$

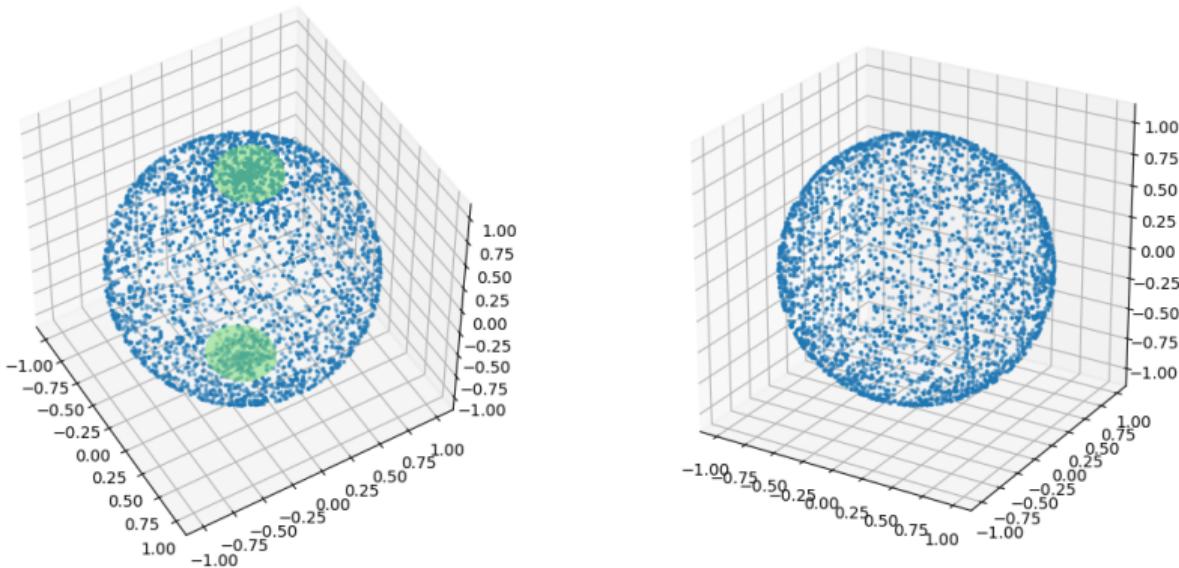
- How to generate random normal vector  $u$ ?
    - Projection to any plane through origin must be symmetric
    - In 3d, uniform latitude then uniform longitude will not do (demo)



## Random vector on unit sphere

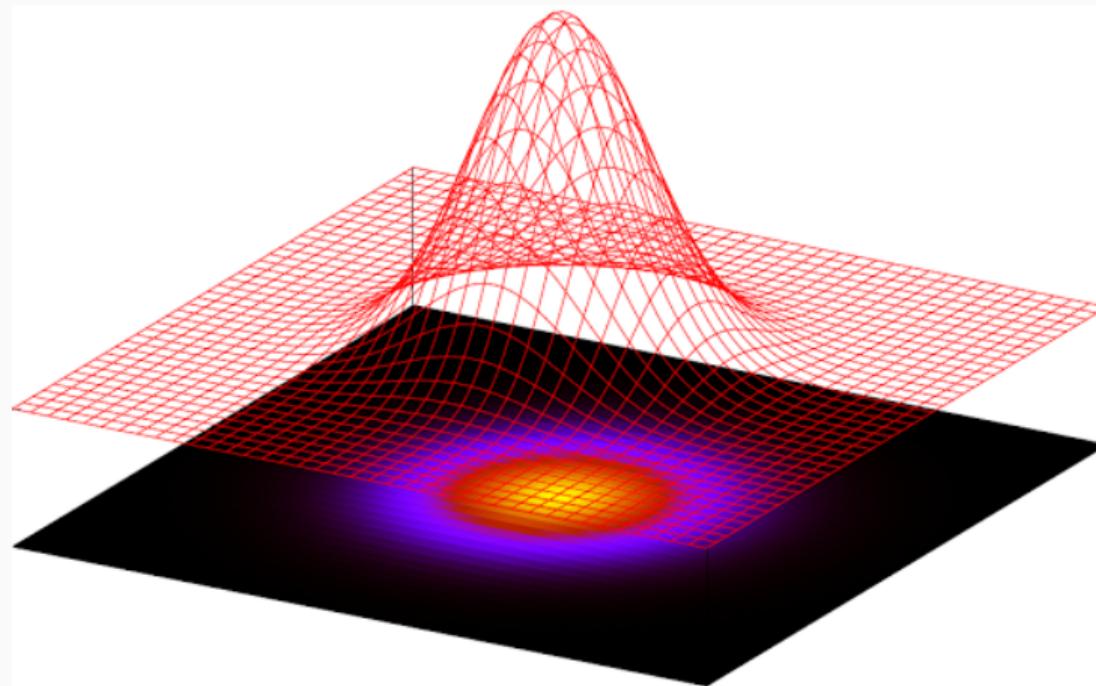


# Random vector on unit sphere



(code)

# Multivariate Gaussian



- Symmetry generalizes to any number of dimensions!
- How to generate multivariate Gaussians?

## Generating random Gaussians (1)

- Suppose we are given a uniform random number generator  $\mathcal{U}[0, 1]$
- How to generate  $X \sim \mathcal{N}(0, 1)$ ?
- Recall the familiar trick to compute  $I = \int_{-\infty}^{\infty} e^{-x^2} dx$ :  
$$I^2 = \left( \int_{x=-\infty}^{\infty} e^{-x^2} dx \right) \left( \int_{y=-\infty}^{\infty} e^{-y^2} dy \right) = \int_{x=-\infty}^{\infty} \int_{y=-\infty}^{\infty} e^{-(x^2+y^2)} dx dy = \\ \int_{\phi=0}^{2\pi} \int_{r=0}^{\infty} e^{-r^2} r dr d\phi = \pi, \text{ so } I = \sqrt{\pi}$$
- Suppose  $X, Y \sim \mathcal{N}(0, 1)$  are two independent random variables distributed standard normal. Then their joint density is

$$f(x, y) = \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right)$$

## Generating random Gaussians (2)

- To sample  $X$  and  $Y$ , we will instead sample two random variables  $R$  and  $\Theta$  from suitable distributions, then apply the transformation  $X = R \cos \Theta$ ,  $Y = R \sin \Theta$
- We can pick  $\Theta \sim \mathcal{U}[0, 2\pi]$ , by the symmetry of  $f$ .
- The cumulative distribution of  $R$  is

$$\begin{aligned} G(r) &= \Pr(R \leq r) = \int_{x^2+y^2=0}^{r^2} f(x, y) dx dy \\ &= \int_{x^2+y^2=0}^{r^2} \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right) dx dy \\ &= \int_{\phi=0}^{2\pi} \int_{t=0}^r \frac{1}{2\pi} \exp\left(-\frac{t^2}{2}\right) t dt d\phi \end{aligned}$$

## Generating random Gaussians (3)

$$= \int_{t=0}^r e^{-t^2/2} t dt = 1 - e^{-r^2/2}$$

- To generate  $R$  from cumulative density  $G$ , generate  $U \sim \mathcal{U}[0, 1]$  and “invert”  $G$ :

$$R = \sqrt{-2 \ln(1 - U)} \quad \stackrel{\text{dist}}{=} \quad \sqrt{-2 \ln U}$$

- Even though  $X$  and  $Y$  are apparently coupled via  $R, \Theta$ , they are independent
- HW
- Can generate any number of standard normal random numbers this way
- Thus sample multivariate normal distribution with identity covariance matrix
- (Can use given mean  $\mu$  and covariance  $\Sigma$  to transform to arbitrary multivariate normal, but we don't need that here)

# From random hyperplanes to Hamming LSH

- A single random hyperplane gives a 1-bit hash
- Sample  $\sqrt{N}$  random hyperplanes
- Each vector in the corpus gets  $\sqrt{N}$  hash bits
- Hyperplanes define  $2\sqrt{N}$  cones
- Two vectors in the same cone agree on all  $\sqrt{N}$  hash bits
- Use these  $\sqrt{N}$ -bit hash codes in a Hamming LSH

Data has  
low conicity



# From cosine LSH to dot LSH

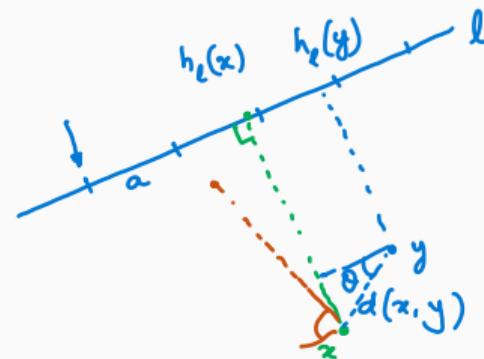
- Suppose we have a random hyperplanes LSH
- How to use it to implement dot-product LSH?
- Given corpus of docs  $\mathbf{x}_n \in \mathbb{R}^D$  and query  $\mathbf{q} \in \mathbb{R}^D$
- Dot-product similarity is  $\text{sim}(\mathbf{q}, \mathbf{x}_n) = \mathbf{q} \cdot \mathbf{x}_n$
- Scale all  $\mathbf{x}_n$  such that  $\max_n \|\mathbf{x}_n\|_2 \leq 1$
- Now transform  $\hat{\mathbf{x}}_n = \left( \underbrace{\mathbf{x}_n}_{\|\mathbf{x}_n\|_2=1}, \sqrt{1 - \|\mathbf{x}_n\|_2^2} \right) \in \mathbb{R}^{D+1}$  and  $\hat{\mathbf{q}} = (\mathbf{q}, 0)/\|\mathbf{q}\|_2 \in \mathbb{R}^{D+1}$   $\|\hat{\mathbf{x}}_n\|_2 = 1$
- Note  $\hat{\mathbf{x}}_n \cdot \hat{\mathbf{q}} = \mathbf{x}_n \cdot \mathbf{q}$  and  $\|\hat{\mathbf{q}}\|_2 = 1$  and all  $\|\hat{\mathbf{x}}_n\|_2 = 1$

# LSH for $L_2$ distance

- Choose randomly oriented line  $\ell$
- Partition  $\ell$  into segments/buckets of width  $a$
- Project each point  $x$  to  $\ell$
- Hash value is the bucket (index) where  $x$  got projected
- (Use multiple randomly oriented lines for more hash values)

For subsequent analysis

- Join points  $x, y$  with line  $\overline{xy}$
- Let  $\theta$  be the angle between  $\overline{xy}$  and  $\ell$
- Let  $d(x, y)$  be  $L_2$  distance between  $x$  and  $y$



# Analysis

- If  $d(x, y) \gg a$ , then  $\theta$  must be close to  $90^\circ$  for there to be a reasonable chance that  $x$  and  $y$  go to the same bucket
- Specifically, if  $d(x, y) \geq 2a$ , then we need  $\theta \in [60^\circ, 90^\circ]$  for  $x$  and  $y$  to go to the same bucket, which happens with probability at most  $1/3$
- If  $d(x, y) \ll a$ , then the chance that  $x, y$  go to the same bucket is large
- Specifically, if  $d(x, y) \leq a/2$ , then the probability that  $x$  and  $y$  will share a bucket is at least  $1/2$
- Thus, we have a ( $d_1 = a/2, d_2 = 2a, p_1 = 1/2, p_2 = 1/3$ )-LSH



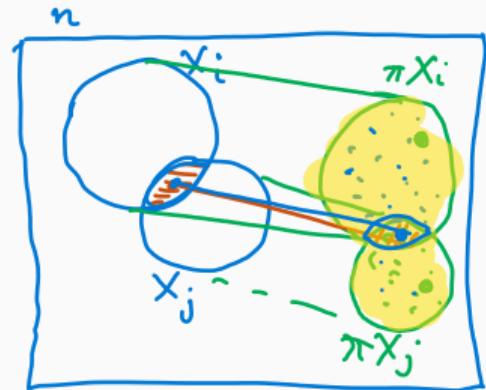
## Jaccard similarity

- Query  $q$ , ‘docs’  $x_i$  are sets — will use  $Q, X_i$  etc.
- Similarity  $\text{sim}(Q, X) = J(Q, X) = (|Q \cap X|)/(|Q \cup X|)$
- Point query: given  $Q$ , find  $K$  docs  $X$  with largest  $J(Q, X)$
- All-pairs: no query; for each doc  $X_i$ , find  $K$  docs  $X_j$  with largest  $J(X_i, X_j)$
- What is a **suitable hash family?**
- How to prepare the sketch of each doc?
- What are  $c, r, P_1, P_2$ ?

# Min-hash

- Suppose all  $X_i \subset \{1, 2, \dots, n\}$
- Let  $\pi$  be a random permutation from  $\{1, 2, \dots, n\}$  to  $\{1, 2, \dots, n\}$  (i.e., one of the  $n!$  permutations chosen uar)
- $\pi X = \pi(X) = \{\pi(x) : x \in X\}$
- What is  $\Pr(\min\{\pi(X_i)\} = \min\{\pi(X_j)\})$ ?
- Equivalently, what fraction of  $n!$  permutations satisfy  $\min\{\pi(X_i)\} = \min\{\pi(X_j)\}$ ?
- Simplify notation: use  $A$  for  $X_i$ ,  $B$  for  $X_j$

H



## Min-hash analysis (1)

- If  $x = \min(\pi(A)) = \min(\pi(B))$ , then  $\pi^{-1}(x)$  must belong to  $A \cap B$
- Pick the range to which  $A \cup B$  is mapped in  $\binom{n}{|A \cup B|}$  ways
- Remaining  $n - |A \cup B|$  elements not chosen may be permuted every way:  
 $(n - |A \cup B|)!$
- Element to be mapped to the minimum value in the range can be chosen in  $|A \cap B|$  ways
- Remaining elements in  $A \cup B$  can be permuted in  $(|A \cup B| - 1)!$  ways

$$\binom{n}{|A \cup B|} (n - |A \cup B|)! |A \cap B| (|A \cup B| - 1)! = \frac{|A \cap B|}{|A \cup B|} n! =$$

## Min-hash analysis (2)

- Simpler way to think:

$$\begin{aligned} & \Pr(\min(\pi A) = \min(\pi B)) \\ &= \Pr(\text{min in } \pi A \cup \pi B \text{ lies in } \pi A \cap \pi B) \\ &= \sum_{x \in \pi A \cup \pi B} \Pr(x \in \pi A \cap \pi B), \end{aligned}$$

where  $x$  is the min, in turn

- or  $|\pi A \cap \pi B| \times \Pr(\text{some fixed member of } \pi A \cup \pi B \text{ is the minimum})$
- Either way, we get

$$\frac{|\pi A \cap \pi B|}{|\pi A \cup \pi B|} = J(A, B)$$

## Sample many permutations

- Over random permutations  $\pi$ , we saw that  
$$\Pr(\min\{\pi A\} = \min\{\pi B\}) = J(A, B)$$
- Sample  $M$  (trial) permutations  $\pi_1, \dots, \pi_M$
- Count (successes)  $Y = |\{m \in [M] : \min \pi_m A = \min \pi_m B\}|$
- Estimate  $J(A, B) \approx Y/M$
- Variance decreases with  $M$  (like coin bias from tosses)
- Randomness is expensive (`/dev/random` vs. `/dev/urandom`)
- Need  $\log n! \sim n \log n$  random bits to sample one of  $n!$
- Can we reduce the size of the family from which we draw permutations?

## Min-wise independent hash functions

- Denote  $[n] = \{1, \dots, n\}$
- Let  $\mathcal{S}_n$  be all permutations from  $[n]$  to  $[n]$
- A family of permutations  $\mathcal{F} \subseteq \mathcal{S}_n$  is exactly min-wise independent if for any  $X \subseteq [n]$  and any  $x \in X$ , when  $\pi$  is chosen uar from  $\mathcal{F}$ , we have

$$\Pr(\min\{\pi(X)\} = \pi(x)) = \frac{1}{|X|}$$

- $\mathcal{F} = \mathcal{S}_n \Rightarrow n!$  permutations
- Sufficient, but not necessary for Jaccard
- Need  $\log(n!) \approx n \log n$  bits to sample one

## Lower bounding $|\mathcal{F}|$

- Small  $\mathcal{F} \Rightarrow$  fewer bits needed to sample  $\pi$
- Each element of  $X$  must be the minimum under  $\mathcal{F}$  the same number of times
- Therefore  $|X|$  must divide  $|\mathcal{F}|$  exactly
- $|X|$  can be  $1, 2, \dots, n$  (or  $1, 2, \dots, k$  for some smaller  $k$ )
- Lcm of  $1, 2, \dots, n$  (or  $1, 2, \dots, k$ ) must divide  $|\mathcal{F}|$
- Lcm of  $1, 2, \dots, n$  (or  $1, 2, \dots, k$ ) is at least  $e^{n-o(n)}$  (or  $e^{k-o(k)}$ ) — Number Theory

## Upper bounding $|\mathcal{F}|$

- Let  $n = 2^r$
- Construct  $\mathcal{F}$  in stages recursively
- First stage: divide  $[n]$  into two halves, top and bottom
- $\binom{n}{n/2}$  ways to do this
- $n$ -bit long string,  $n/2$  0s,  $n/2$  1s
- Second stage: divide the halves into quarters, etc.
- Important: can reuse one  $n/2$ -bit string
- Number of permutations

$$|\mathcal{F}| = \prod_{i=1}^{\log n} \binom{n/2^{i-1}}{n/2^i} \leq \prod_{i=1}^{\log n} 2^{n/2^{i-1}} \leq 2^{n(1+1/2+\dots)} \leq 4^n$$

- I.e.,  $O(n)$  random bits suffice—still large

## Approximate min-wise independent families

- For any  $X \subseteq [n]$  and any  $x \in X$ , when  $\pi$  is chosen uar from  $\mathcal{F}$ ,

$$\left| \Pr\left(\min\{\pi(X)\} = \pi(x)\right) - \frac{1}{|X|} \right| \leq \frac{\epsilon}{|X|}$$

- There exist families of size  $O(n^2/\epsilon^2)$  that are  $\epsilon$ -approximately min-wise independent ( $0 \leq \epsilon \leq 1$ )
- Linear independent families with prime  $n$  and  $h(x) = ax + b \pmod{n}$
- For each  $|X| = k$  and for each  $x \in X$

$$\Pr(\min\{\pi(X)\} = \pi(x)) \geq \frac{1}{2(k-1)}$$

## Min hash summary

- Min-wise independent family has size at least  $e^{n-o(n)}$
- Min-wise independent family has size at most  $4^n$
- Allowing  $\epsilon$ -approximate min-wise independence reduces family size to  $O(n^2/\epsilon^2)$ , which needs  $O(\log n)$  random bits to sample

# Applications

---

## All-pairs “find similar”

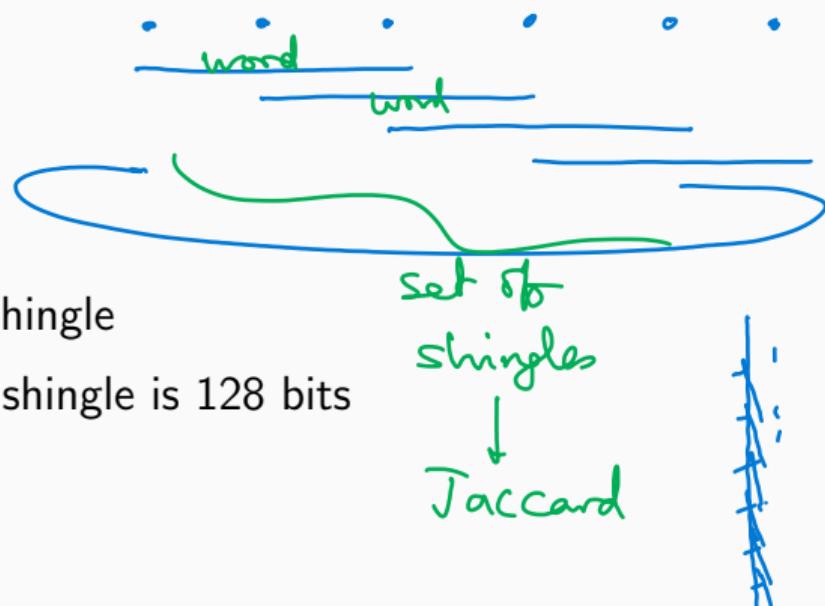
- Given  $n \approx 10^{10}$  Web pages (documents)
- For each doc, find 10 most similar documents
- Output size is  $10n$
- Produce the output in  $o(n^2)$  time
- Can use standard definitions of similarity
- For any definition, we eventually use the Hamming hash
- Each doc goes into one bucket in each of  $L$  hashtables
- High-recall policy: docs  $d_i, d_j$  may be scored for similarity if they share a bucket in at least one hashtable *only*
- Count the number of hashtables where  $d_i, d_j$  share a bucket
- If the count is “large enough” then compute similarity score

## First-cut pseudocode

```
1: for each random permutation  $\pi$  do
2:   create a file  $f_\pi$ 
3:   for each document  $d$  with word set  $T(d)$  do
4:     write out  $\langle s = \min \pi(T(d)), d \rangle$  to  $f_\pi$ 
5:   end for
6:   sort  $f_\pi$  using key  $s$ —this results in contiguous blocks with fixed  $s$  containing
    all associated  $d$ s
7:   create a file  $g_\pi$ 
8:   for each pair  $(d_1, d_2)$  within a run of  $f_\pi$  having a given  $s$  do
9:     write out a document pair record  $(d_1, d_2)$  to  $g_\pi$ 
10:  end for
11:  sort  $g_\pi$  on key  $(d_1, d_2)$ 
12: end for
13: merge  $g_\pi$  for all  $\pi$  in  $(d_1, d_2)$  order
```

# Shingling

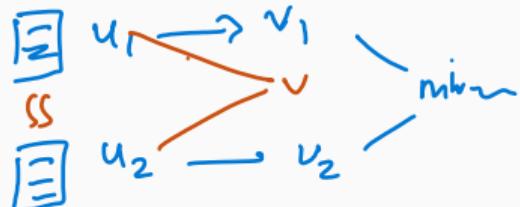
- From word set overlap to detecting mirroring or plagiarism
- Approximate sequence matching
- Pairwise edit distance too slow
- Turn document into token sequence
- Sliding window of size  $w = 4$ , say
- Each possible window value is called a shingle
- If each token represented using 32 bits, shingle is 128 bits
- Thus  $n = 2^{128}$



# Detecting locally similar Web subgraphs

- After crawling *saves storage*
  - Contract graph, save RAM and disk
  - More faithful link-based ranking algorithms
  - Apart from genuine content mirrors, cases like `http://www.yahoo.com/` and `http://dir.yahoo.com/; \<a base=.../\>`, virtual hosts, URL rewrites, etc.
- During crawling *saves both n/w and storage*
  - Especially valuable in avoiding fetching same content many times
  - Only clue available is the URL (and possible text on a few crawled pages)
  - Often enough if corroborative info used properly
  - Can use known suspects from previous crawls

# Graph contraction



- Consider links  $(u_1, v_1)$  and  $(u_2, v_2)$
- Say shingling leads you to believe that  $v_1$  and  $v_2$  are mirror pages
- Replace outlinks on  $u_1$  and  $u_2$  to point to unified node  $v$
- May make  $u_1$  and  $u_2$  look more similar than before, especially if  $u$  are represented only by outlink sequences
- Could lead to cascaded collapses and whole site mirror folding

## Shingling URLs (1)

- Identify candidate host pairs that might be mirrors to perform a more thorough check
- Convert host and path to all lowercase characters
- Let any punctuation or digit sequence be a token separator
- Tokenize the URL into a sequence of tokens, for example, *www6.infoseek.com* gives *www, infoseek, com*
- Eliminate frequent URL components ('stopwords') such as *htm, html, txt, main, index, home, bin, cgi*

## Shingling URLs (2)

- Form positional bigrams from the token sequence, for example, /cellblock16/inmates/dilbert/personal/foo.htm yields bigrams (*cellblock,inmates,0*), (*inmates,dilbert,1*), (*dilbert,personal,2*), and (*personal,foo,3*)
- A host is now represented by a set of positional bigrams, just like documents were represented as sets of shingles
- Once min-hash identifies suspected mirror hosts, can do a slower but more thorough textual similarity check

# Summary

---

## LSH summary

- Artifacts may be bit vectors, real vectors / directions, sets
- Map each artifact to a set of sketches
- Sketch values often used to populate (hash) tables
- Approximate range query
- Exhaustively search through very few buckets