

Mining the Web

Similarity search

Soumen Chakrabarti

September 6, 2019

Similarity search

- Given a query, find similar documents
- Docs d_1, d_2 very similar to each other both irrelevant or both relevant to query q
- Extreme case: duplicates, mirror sites
- Given d_1 as query, find similar d_2 s
- Indexed access much slower because docs longer than queries
- Ideally, detect before incurring crawling costs
- At least dedup after crawling for better search response pages
- Exact duplicates: hashing
- Near-duplicates: ?
- Also useful for images, other multimedia

Exact hashing

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 αb_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 αb_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

Locality sensitive hashing

Locality sensitive hash functions

- Sometimes, you want the hash function to be **locality preserving** — **similar** domain objects should map to nearby numbers in the range
- Wish to support different notions of similarity based on application
- Jaccard similarity between sets: $\text{sim}(A, B) = \frac{|A \cap B|}{|A \cup B|}$
 - ▶ HW Show that $1 - J(A, B)$ satisfies triangle inequality
- Cosine similarity between vectors: $\text{sim}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|_2 \|\vec{b}\|_2}$
 - ▶ HW Show that $1 - \cos(\vec{a}, \vec{b})$ satisfies triangle inequality
- L_1 distance between vectors: $\|\vec{a} - \vec{b}\|_1 = \sum_j |a_j - b_j|$
- L_2 distance $\|\vec{a} - \vec{b}\|_2 = \sqrt{\sum_j (a_j - b_j)^2}$

“Find similar”

Several variations

- Given page, find (approximate) mirror pages
- Given fragments of a site, find (approximate) mirror sites
- Given page, find page close in vector space but likely not a mirror

Challenge: Given $n \approx 10^{10}$ pages, for each page, list 10 pages most likely to be a mirror (or most similar in vector space)

Do this in $o(n^2)$ time, before the sun becomes a red giant

We will begin with a simple notion of document similarity:

- A document is a set of words
- Used Jaccard similarity between sets

Min-hash

- Cannot evaluate $J(S_i, S_j)$ over all pairs i, j
- Suppose all $S \subset \{1, 2, \dots, n\}$
- Let π be a random permutation from $\{1, 2, \dots, n\}$ to $\{1, 2, \dots, n\}$ (i.e., one of the $n!$ permutations chosen uar)
- $\pi(S) = \{\pi(s) : s \in S\}$
- What is $\Pr(\min\{\pi(S_i)\} = \min\{\pi(S_j)\})$?
- Equivalently, what fraction of $n!$ permutations satisfy $\min\{\pi(S_i)\} = \min\{\pi(S_j)\}$?

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:

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

where x is the min, in turn

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

$$\frac{|\pi S \cap \pi T|}{|\pi S \cup \pi T|} = J(S, T)$$

Sample many permutations

- Over random permutations π , we saw that $\Pr(\min\{\pi(S)\} = \min\{\pi(T)\}) = J(S, T)$
- Sample M (trial) permutations π_1, \dots, π_M
- Count (successes) $Y = |\{m \in [M] : \min \pi_m(S) = \min \pi_m(T)\}|$
- Estimate $J(S, T) \approx Y/M$
- Variance decreases with M
- But how does this remove the all-pairs bottleneck?

First-cut pseudocode

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

Avoiding random permutations

- Needed to pick π from the “hash family” of all $n!$ random permutations
- Family size is $n!$, so need $\log n! \sim n \log n$ random bits to sample one
- High-quality random bits are expensive: `/dev/random` vs. `/dev/urandom`
- 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 π 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 π
- 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 π is chosen uar from \mathcal{F} ,

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

- There exist families of size $O(n^2/\epsilon^2)$ that are ϵ -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 ϵ -approximate min-wise independence reduces family size to $O(n^2/\epsilon^2)$, which needs $O(\log n)$ random bits to sample

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
 - 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/; \`, virtual hosts, URL rewrites, etc.
- During crawling
 - 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⁾

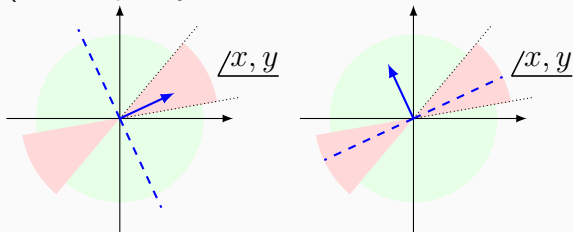
- Identify candidate host pairs that might be mirrors to perform a more thorough check
- Convert host and path to all lowercase characters
- Let any punctuation or digit sequence be a token separator
- Tokenize the URL into a sequence of tokens, for example, *www6.infoseek.com* gives *www*, *infoseek*, *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

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)



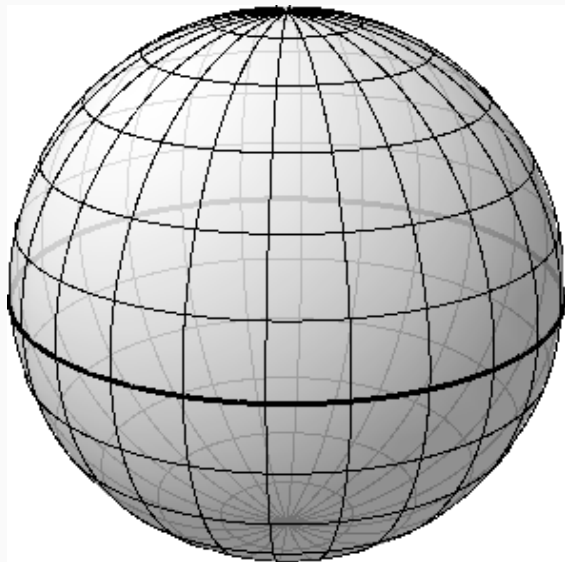
Cosine similarity (2)

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

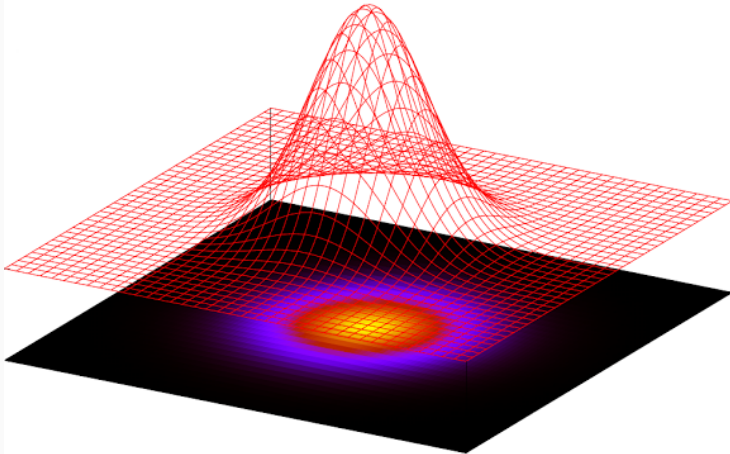
$$\Pr_{u \in \mathcal{F}}(h_u(x) = h_u(y)) = 1 - \frac{\angle x, y}{\pi} \stackrel{\text{rank}}{=} \cos \angle x, y$$

- 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



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 Θ 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

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

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 :

$$R = \sqrt{-2 \ln(1 - U)}$$

- Even though X and Y are apparently coupled via R, Θ , 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 μ and covariance Σ to transform to arbitrary multivariate normal

LSH for cosine: implementation details

- A single random hyperplane gives a 1-bit hash
- Divides all vectors in the corpus into two roughly equal halves
- Will still result in too many pairwise comparisons
- Consider K random hyperplanes
- Each vector in the corpus gets K hash bits
- Hyperplanes define 2^K cones
- Two vectors in the same cone agree on all K hash bits
- Tune K to balance pairwise comparison cost vs. recall of near pairs
- Loss of near-pair recall across cone boundaries?

Hamming distance and bit sampling

- 2^N strings $\{0, 1\}^N$
- 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.,

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

- **Define** $\text{sim}(x, y) = 1 - \frac{\|x, y\|_H}{N} \in [0, 1]$
- 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
 - $\|x, y\|_H \leq r \implies \Pr(h(x) = h(y)) \geq P_1$
 - $\|x, y\|_H \geq cr \implies \Pr(h(x) = h(y)) \leq P_2$
- $c > 1, P_1 > P_2$ 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 $p : d(q, p) \leq r$, then return some p' with $d(q, p') \leq cr$

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 string 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
- Each data item x is replicated (“pointer” only) to all L hash tables
- In hash table ℓ , 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
- At any time if more than $2L$ items x have been checked (including duplicates), terminate the search

The above setup is correct under these conditions:

- If $\exists x : \|q, x\|_H \leq r$, then $g_\ell(q) = g_\ell(x)$ for some ℓ
- There are at most $2L$ items x such that $\|q, x\|_H \geq cr$ and yet $g_\ell(q) = g_\ell(x)$ for some ℓ

Analysis: Few far points collide ⁽¹⁾

- Notational convenience: $H(x, y) = \|x, y\|_H$
- For any ℓ and any x , if $H(q, x) \geq cr$ (point is *far* from query), then $\Pr(g_\ell(q) = g_\ell(x)) \leq P_2^k$
- Set $k = \frac{\log n}{\log(1/P_2)}$, so that $P_2^k = 1/n$
- There are at most n 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 ℓ) is at most $n(1/n) = 1$
- 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 ℓ 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 ℓ , 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 $1 - (1 - n^{-\rho})^L$
- Set $L = n^\rho$, then the probability of a near point colliding is at least $1 - 1/e$

► HW

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 $v(x')$

LSH for L_2 distance

- Choose randomly oriented line ℓ
- Partition ℓ into segments/buckets of width a
- Project each point x to ℓ
- 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 θ be the angle between \overline{xy} and ℓ
- Let $d(x, y)$ be L_2 distance between x and y

Analysis

- If $d(x, y) \gg a$, then θ must be close to 90° 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

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