# Hashing for Data with some Distance Structure

(Near Neighbor Search and Locality Sensitive Hashing).

Often data has structure and hashing tends to remove it—
we want that changing a single letter in a word causes it
to hash to a very different place — look independent!

But at other times we may want to preserve some structure
- we're aiming for compression without losing the notion of
similarity in the original space

Example: we're given strings and want to find pairs of "similar" strings

e.g in document de-duplication.

or given a corpus $S$ of strings, and a query $q$,
find strings in $S$ that are "close to" $q$.

e.g in web searching.

Note: exact problem is easier, using just standard hashing.
- hash all strings in the corpus into a table
- when query comes, look at strings that hash to location $h(q)$
and explicitly compare to them.

We're interested in approximate similarity.

Let's look at an example.

Suppose data are strings (maybe files of text)

need a notion of similarity (metric space)

(a) could imagine the underlying points belong to $\{A\text{-}Za\text{-}z0\text{-}9\}^*$

and distance is Edit distance.

+ Natural
- difficult to manipulate, do searches.

(b) strings are bit vectors (say in ASCII representation).

distance is Euclidean or Hamming distance

+ Easier to manipulate / understand
- no more intuition for what is similar or different.
- different length strings make it awkward

(c) Convert document into a "bag of words".

- Say $10^5$ words in English, keep count of how many times each word appears in document / string.

+ each document $\in \mathbb{Z}^{10^5}$
- lost order information / proximity information.
- taking two copies of same doc gives very different point
- stop words can overwhelm all else

- Normalize, say by total # terms in doc (TF/IDF term frequency / inverse doc. freq).

and ~~else~~ multiply by how common/rare word is in the corpus.

Say $\left(\frac{\text{freq of } w}{\text{total \# of words in doc}}\right) \times \log\left(\frac{\text{\#docs in corpus}}{\text{\#docs with word } w}\right)$

TF IDF

Also to keep some local structure :-

shingling — take k consecutive words in doc, call that a shingle

(eg. 3-shingles)

Now do same things for these shingles. ← also do for images

Maybe drop stop words.

+ hundreds of useful cool tricks.

---

But eventually have say ← a bag of words

(a) a set of items, want to see if they are similar, or

(b) a set of vectors ← bag put in some geometric representation.

---

What's the notion of similarity ? or distance ?

Let's look at the vector case, say.

. Each item is a vector $v_i$

. Distance is Hamming distance :- $dist(v_i, v_j)$ = # coordinates where they differ.

or Euclidean distance :- $\|v_i - v_j\| = \sqrt{\sum_{k=1}^{d} |v_{ik} - v_{jk}|^2}$

or $L_1$-distance

(Manhattan / taxicab distance)

$$\|v_i - v_j\|_1 = \sum_{k=1}^{d} |v_{ik} - v_{jk}|$$

or millions of others.

Let's try to solve the (approximate) near neighbors problem

Given a metric space (distances, say $\mathbb{R}^d$)

and a corpus $S \subseteq \mathbb{R}^d$ of points

Create a data structure. such that it answers

queries of the form "return a near-neighbor of point $q$".

i.e. return a point $a \in S$ ← answer

$$\text{st} \quad d(q,a) \leq c \times \min_{a^* \in S} d(q,a^*)$$

↑ nearest neighbor distance

↑ approx factor. say 2 ← today's lecture

say $(1+\varepsilon)$.

---

Challenge: $S$ is large. say $n = |S| = 10^{10}$ points

Strawman solution:

no data structure. Brute force search each time.

So $O(nd)$ time per query

↑ # points in $S$ ↑ $O(d)$ to compute distance, say

Should do better.

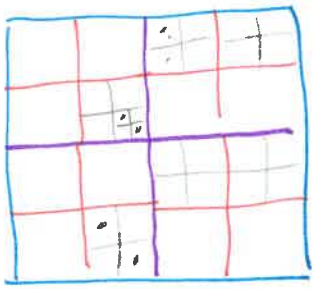
Indeed, will use Locality Sensitive hashing

- similar items fall "close" to each other.

dissimilar items do not.

Want: data structure with small storage space, small preprocessing time and small query time

## Solution #2 (Strawman++)



Use k-d trees (or variant)

Partition bounding box for all points in S into half along each coordinate

and so on., until each smallest box contains a single point.

For each box keep a representative point

Given query, go down the hierarchy, keeping closest representative at each level ~~find~~ look at all neighboring boxes as well.


So takes time $2^d \cdot \log\left(\frac{d_{max}}{d_{min}}\right)$ for queries

$2^d$: # of neighboring boxes. $\log\left(\frac{d_{max}}{d_{min}}\right)$: # levels of recursion

## Curse of dimensionality!

- high dimensional point sets take exp(dimension) query time

want to do better than that.

Note: have gone from $O(nd)$ query time, no preprocessing

to $O(2^d \cdot \log\left(\frac{d_{max}}{d_{min}}\right))$, $O(n \cdot \log\left(\frac{d_{max}}{d_{min}}\right))$ space & preprocessing.

## Curse of dimensionality.

if $n$ items in $d$-dimensional space, searching for near neighbors in this space typically takes $2^{\Omega(d)}$ time

Exponential dependence. ☹ if we do it naively.

(E.g. if we do it via quad-trees).
↑ again naively.

---

So lets be smarter.

**Idea #1:** Doubling Search.

Given $S$ and parameter $r$, solve the problem

Ball search
- given query $q$, if $\exists\ a \in S$ at distance$(q,a) \le r$
  return some $a'$ at distance $\le (1+\varepsilon)r$.
- if no $a \in S$ at distance $\le r$ from query $q$,
  return whatever you want.

Now can try all values of $r$, • get an answer $a_r$ for that value
↑ powers of $(1+\varepsilon)$
i.e. $r_i = (1+\varepsilon)^i$
- check the distance $(q, a_r)$
- return the best of those.

if actual closest point is $a^* \in S$ at distance $\in [(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$

then will get a good answer for $r = (1+\varepsilon)^{i+1}$!

that has distance from $q$ being $\le (1+\varepsilon)^{i+2}$

$\le (1+\varepsilon)^2 \cdot d(q, a^*)$

$\cong (1+2\varepsilon)\ dist(q, a^*)$.

OK: how to solve the doubling search version?

Given $S$, radius $r$, answer (approx) queries:

- if $\exists$ a point at distance $\le r$ from $q$
  return a point at dist $\le 2r$ (say)

Here's idea:

Use hash function $h$: metric space $\longrightarrow$ buckets

such that

locality sensitive hash
LSH.

$$\text{if } \begin{cases} \text{dist}(x,y) \le r & \Rightarrow \Pr[h(x)=h(y)] \ge P_{close} \\ \text{dist}(x,y) \ge 2r & \Rightarrow \Pr[h(x)=h(y)] \le P_{far} \end{cases}$$

Ideally: $P_{close} \gg P_{far}$

Use these to hash into buckets.

So $E[\#\text{junk points in } h(q) \text{ bucket}] \le \sum_{x:\text{dist}(q,x) \ge 2r} P_{far} \cdot n.$

$\le 1$ if $P_{far} = 1/n$.

But then maybe $P_{close}$ is small as well.

So repeat $\cong 1/P_{close}$ times, i.e. use $1/P_{close}$ many independent hash functions to increase probability we see a close point.

## Let's do a concrete example:

Want to distinguish points in $\{0,1\}^d$ ← Boolean cube with Hamming distance.

that are at distance $\leq r$ or at distance $\geq 2r$.

Try #0: $h(x) =$ pick a random coordinate $i$ and output $x_i$

if $dist(x,y) \leq r \Rightarrow Pr[h(x)=h(y)] \geq 1 - r/d = P_{close}$

$dist(x,y) \geq 2r \Rightarrow Pr[h(x)=h(y)] \leq 1 - \frac{2r}{d} = P_{far}$

Very small gap between $P_{close}$ & $P_{far}$.

Try #1: Parallel Repetition

$h(x) =$ pick $\ell$ random coordinates with repetition output those $\ell$ coordinate values

if $dist(x,y) \leq r \Rightarrow Pr[h(x)=h(y)] \geq (1-r/d)^\ell = P_{close}$

$\geq 2r \Rightarrow \leq (1-2r/d)^\ell = P_{far}$

As always: if $r \ll d$ (say) then $1 - r/d \approx e^{-r/d}$ etc.

$\Rightarrow P_{close} \approx e^{-r/d \cdot \ell}$

$P_{far} \approx e^{-2r/d \ell} = 1/n$ (say)

$\Rightarrow \frac{2r\ell}{d} = \log_2 n \Rightarrow \ell = \frac{d \cdot \log_2 n}{2r}$.

$\Rightarrow P_{close} \cong e^{-r/d \cdot \ell} = e^{-\frac{1}{2}\log_2 n} = 1/\sqrt{n}.$

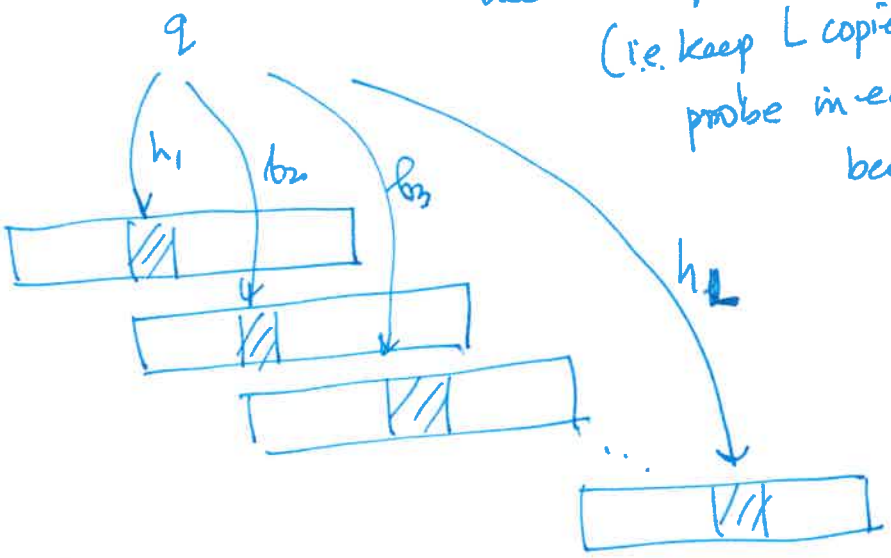
$\Rightarrow$ if $x$ and $q$ are indeed at distance $\leq r$

then $Pr[h(x) = h(q)] \geq 1/\sqrt{n}$ ← small probability, but much better than $P_{far}$.

So final ingredient: ~~Parallel~~ Serial Repetition

Use $L$ independent hash functions

(i.e. keep $L$ copies of data structures, probe in each and return the best




If we set $L = \sqrt{n}\log n$

$\Rightarrow Pr[\exists \text{ table } i \text{ s.t. } h_i(q) = h_i(x)] \geq 1 - \left(1 - \frac{1}{\sqrt{n}}\right)^L$

$\cong 1 - \left(e^{-1/\sqrt{n}}\right)^L$

$= 1 - e^{-\log n} = 1 - \frac{1}{n}.$

So in summary:

used a weak hash function $h$ with $P_{close}, P_{far}$ not very different

then reduced $P_{far}$ by parallel repetition

and handled the "small $P_{close}$ value" problem by serial repetition.

Note: if $s = \Pr[h(x) = h(q)$ for a single hash function$]$.

then $\Pr(h^\ell(x) = h^\ell(q)$ for the function after parallel rep$]$

$= s^\ell$.

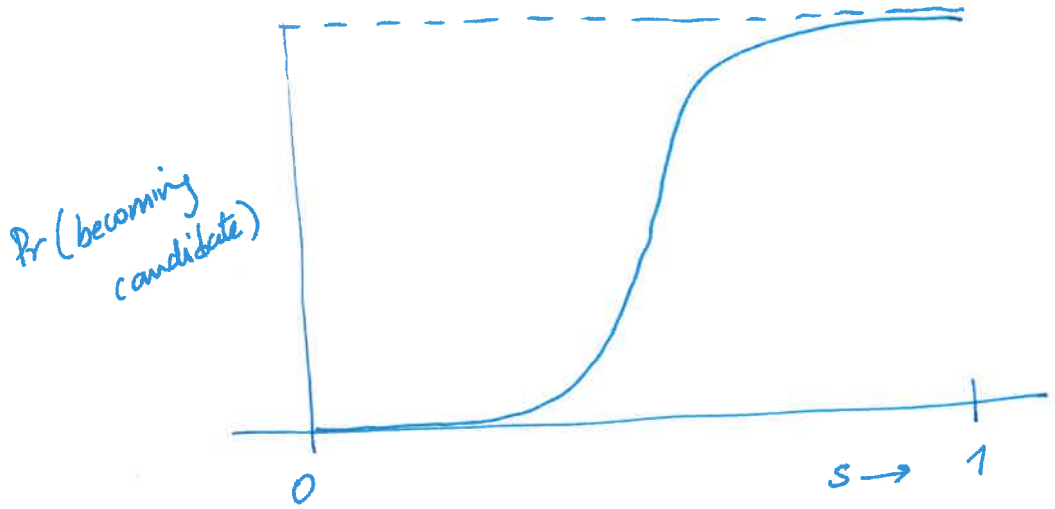
$\Rightarrow \Pr[h^\ell(x) \neq h^\ell(q)] = (1-s^\ell)$.

$\Rightarrow \Pr[x$ is not seen in all $t$ copies of the data structure$]$

$= (1-s^\ell)^t$

$\Rightarrow \Pr[x$ is one of the candidates for query $q]$

$= 1 - (1-s^\ell)^t$




$1 - (1-s^\ell)^t \cong 1/2$ at $(1-s^\ell)^t \cong 1/2$ or $s \cong (1/t)^{1/\ell}$

⇒

by choosing $t, \ell$ carefully

- can get space usage $n^{2-\varepsilon}$ and query time $n^{1-\varepsilon}$ in worst case
- in practice does quite a bit better.

↑ for $\gamma$ vs $(1+\varepsilon)r$

Observe: steps II (parallel repetition) and III (serial repetition) are completely generic.

So: if we can get a small gap between collision probability in the close & far cases ⇒ can amplify the success.

---

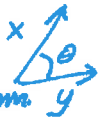
Saw Step I for bit strings and Hamming distance.

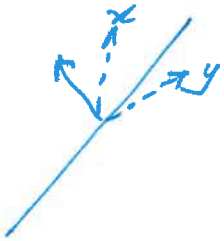
Can do this for other metric spaces:-

- Unit vectors and cosine distances

Elements are unit vectors.

$d(x,y) = \theta$ $\theta$ is angle between them




hash function: pick a random hyperplane.
defines a halfspace. $x \in$ Halfspace ⇒ $h(x) = 1$
else $h(x) = 0$.

$$Pr[h(x) \neq h(y)] = \theta/\pi$$

- Sets and Jacquard distance

. $d(A,B) = 1 - \frac{|A \wedge B|}{|A \cup B|}$.

"minhash"

. hash function: pick a random permutation on elements.
$h(S) =$ first element (in this permutation) in $S$.

$$Pr[h(A) = h(B)] = \frac{|A \wedge B|}{|A \cup B|} \Rightarrow Pr[h(A) \neq h(B)] = d(A,B).$$

## Take-aways

- Can easily extend the basic ideas behind hashing to structured data.

  - Call two items $x, y$ similar if $d(x,y) \leq r$

    dissimilar if $d(x,y) \geq cr$.

    Grey zone in the middle (don't care either way)

LSH idea:

then can ask for

$P_r$ [similar items collide] $\geq$ large

$P_r$ [dissimilar items collide] $\leq$ small

And then amplify correctness by repetition.

- Can do this for many metric spaces
  - Hamming Cube (in lecture)
  - Euclidean distances (embed into Ham Cube) ← omitted, see MMDS book.
  - Cosine distances
  - Jacquard distances
  - ⋮

↑ depends on dimension to some extent (not in same way as k-d tree)

- Popular and commonly used heuristic

  (good in theory, even better in practice).

Next time: Reduce the ambient dimension of your data.