Near-Neighbour Search & Locality-Sensitive Hashing.

TSS 20/09/23

-Deepanshu Kush

Outline

- Problem Definition & Background
- Approximate Version
- LSH Families (Simple)
- Data Structure Construction
- LSH Families (Complex)
- Recent Developments

Nearest Neighbour Search

Nearest Neighbour Search

Given:

1. Database $X = \{x_1, \ldots, x_n\} \subset U \subseteq \mathbb{R}^d$ (eg. $U = \{0,1\}^d$)
2. A distance metric $d(a,b)$ for points $a, b \in U$

Nearest Neighbour Search

Given:

1. Database $X = \{x_1, \ldots, x_n\} \subset U \subseteq \mathbb{R}^d$ (eg. $U = \{0,1\}^d$)
2. A distance metric $d(a,b)$ for points $a, b \in U$

Goal:

1. Construct a data structure for X (pre-processing, extra space allowed)

Nearest Neighbour Search

Given:

1. Database $X = \{x_1, \ldots, x_n\} \subset U \subseteq \mathbb{R}^d$ (eg. $U = \{0,1\}^d$)
2. A distance metric $d(a,b)$ for points $a, b \in U$

Goal:

1. Construct a data structure for X (pre-processing, extra space allowed)
2. Given any query pt. $q \in U$, find a nearest pt. in X

ie. $\operatorname{argmin}_{x \in X} d(q,x)$

What parameters do we care about?

What parameters do we care about?

> space complexity & query time

What parameters do we care about?

- Space complexity & query time
- Some tradeoff to be expected

What parameters do we care about?

- > space complexity & query time
- > some tradeoff to be expected

Naively: Just store the n pts using $O(dn)$ space & do linear search ie. $O(n)$ query time.

What parameters do we care about?

> space complexity & query time
> some tradeoff to be expected

Naively: Just store the n pts using $O(dn)$ space & do linear search ie. $O(n)$ query time.

So, goal: try for $o(n)$ time, more space

What parameters do we care about?

- > space complexity & query time
- > some tradeoff to be expected

Naively: Just store the n pts using $O(dn)$ space & do linear search ie. $O(n)$ query time.

So, **goal**: try for $o(n)$ time, more space

⚠ minimize pre-processing time for practical apps

$d = 1$

d = 1

> Pre-processing: Sort the n pts

d = 1

> Pre-processing: Sort the n pts

> Given a query q, binary search to find a nearest pt

d = 1

> Pre-processing: Sort the n pts

> Given a query q, binary search to find a nearest pt

> query-time: $O(\log n)$, space $O(n)$, pre-processing $O(n \log n)$

Higher d?

Higher d?

k-d tree:

Higher d?

k-d trees: Partition $\mathbb{R}^d$ using coord-aligned planes, chosen suitably.

Higher d?

k-d trees: Partition $\mathbb{R}^d$ using coord-aligned planes, chosen suitably.

> Unfortunately, fail to beat naive as soon as $d = \Omega(\log n)$

Higher d?

k-d trees: Partition $\mathbb{R}^d$ using coord-aligned planes, chosen suitably.

> Unfortunately, fail to beat naive as soon as $d = \Omega(\log n)$

> Moreover, in all known approaches, size of DS / q. time grows like $\exp(d)$

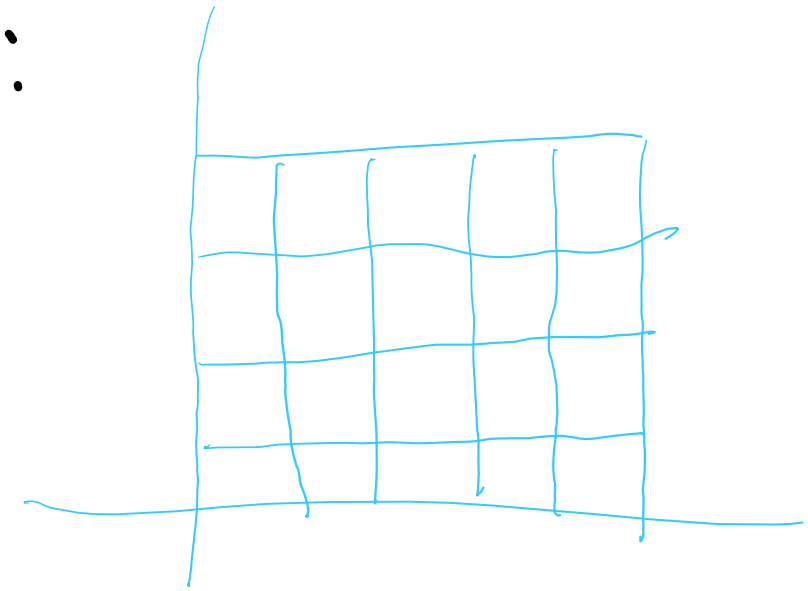
Curse of Dimensionality

Curse of Dimensionality

> Well-known phenomenon observed in many geometric problems, including NNS.

Curse of Dimensionality

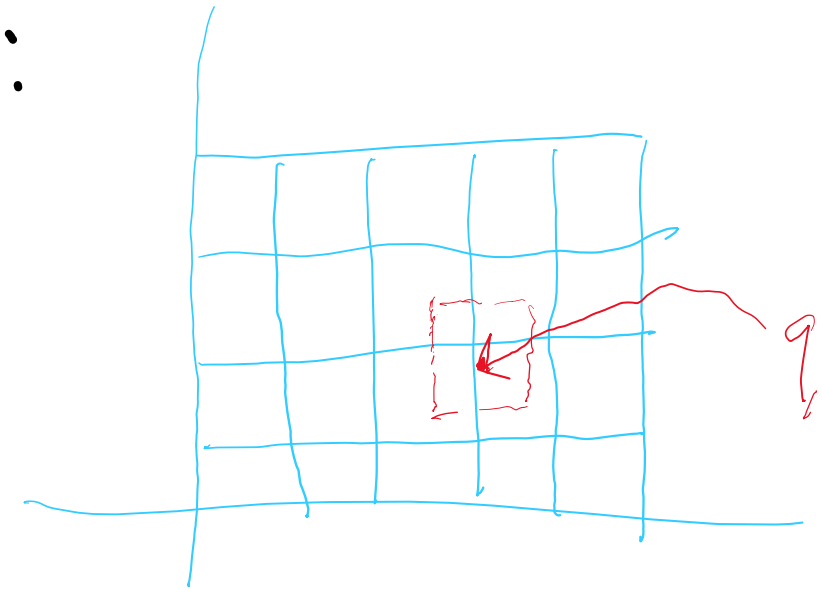
- Well-known phenomenon observed in many geometric problems, including NNS.
- Intuition:



grid of side a.

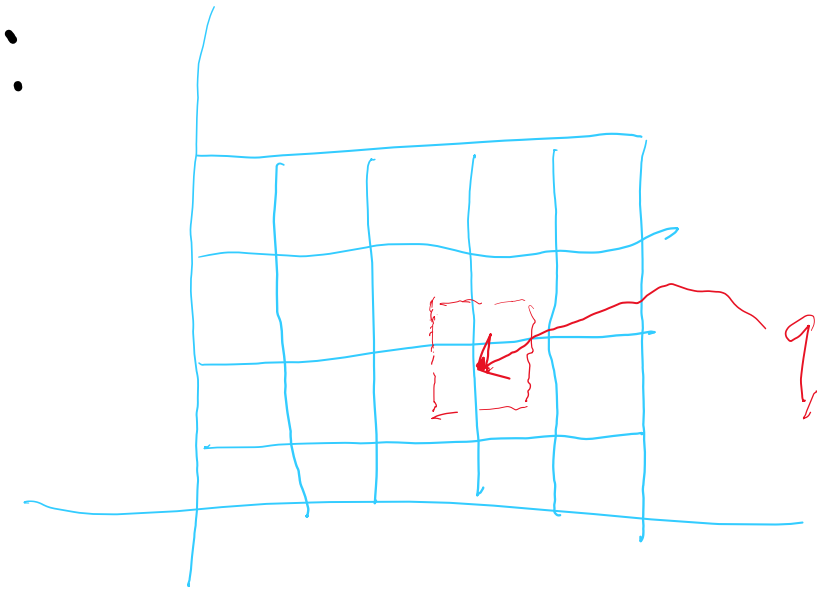
Curse of Dimensionality

- Well-known phenomenon observed in many geometric problems, including NNS.
- Intuition:



Curse of Dimensionality

- Well-known phenomenon observed in many geometric problems, including NNS.
- Intuition:



- NNS based on k-d trees will need $\Omega(2^d)$ lookup time

Example Applications

Example Applications

> Shazam!

Example Applications

> Shazam!

> Task: Identify a song from an audio recording of a short clip from the song

Example Applications

> Shazam!

> Task: Identify a song from an audio recording of a short clip from the song

> Idea (from Avery Li-Chun Wang '03, ICMIR):

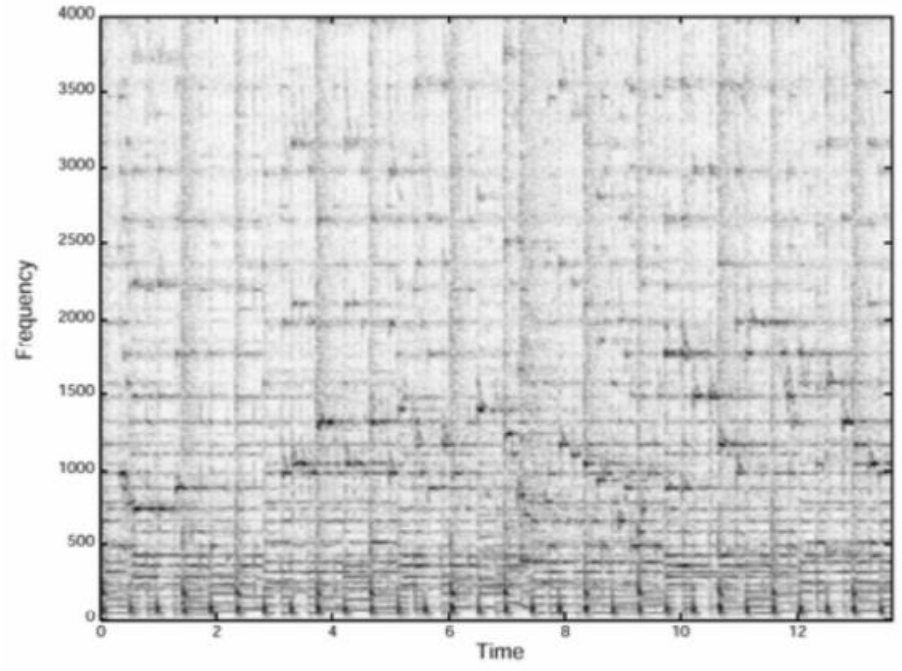
Example Applications

> Shazam!

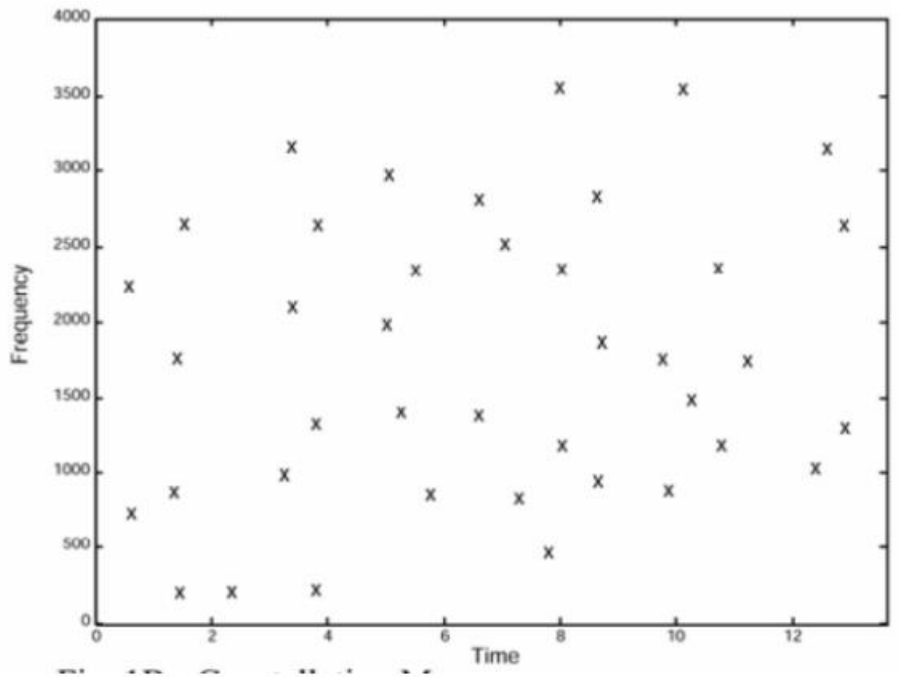
> Task: Identify a song from an audio recording of a short clip from the song

> Idea (from Avery Li-Chun Wang '03, ICMIR): First construct a "spectrogram" (some windowed FFT). Then identify the clip. From "peaks" in the spectrogram to construct $q \in \mathbb{R}^d$. Then apply NNS for a huge database of already processed clips.

Picture:



(a) Spectrogram extracted from clip of an audio file.



(b) Post-processed spectrogram which can be used to construct an audio "fingerprint" $q \in \mathbb{N}^d$ for identifying a song clip.

Figure 1: Images from the paper outlining Shazam's approach to audio retrieval [2]. Once features are extracted from an audio clip, a hashing based algorithm is used to perform approximate nearest neighbor search in a database of known songs.

More Examples:

More Examples:

- Used in detecting early warning signs of earthquakes and other seismic events

More Examples:

> Used in detecting early warning signs of earthquakes and other seismic events

> Again, spectrogram of a seismogram is computed in real-time & compared (quickly) against a database of surrounding major seismic events

More Examples:

> Used in detecting early warning signs of earthquakes and other seismic events
> Again, spectrogram of a seismogram is computed in real-time & compared (quickly) against a database of surrounding major seismic events
> Also a very high-dim problem

Approximate NNS

> Fortunately, break the curse if we're willing to tolerate some level of approximation.

Approximate NNS

> Fortunately, break the curse if we're willing to tolerate some level of approximation.

> (c, r)-NNS: Given $X \subset U \subseteq \mathbb{R}^d$ of n pts, construct a DS s.t. for any query pt q, if $\exists p \in X$ s.t. $d(p, q) \leq r$, output a pt p' s.t. $d(p', q) \leq cr$.

Approximate NNS

> Fortunately, break the curse if we're willing to tolerate some level of approximation.

> (c, r)-NNS: Given $X \subset U \subseteq \mathbb{R}^d$ of n pts, construct a DS s.t. for any query pt q, if $\exists p \in X$ s.t. $d(p, q) \leq r$, output a pt p' s.t.

$$d(p', q) \leq cr.$$

> if there is no such p, no requirements.

Picture:

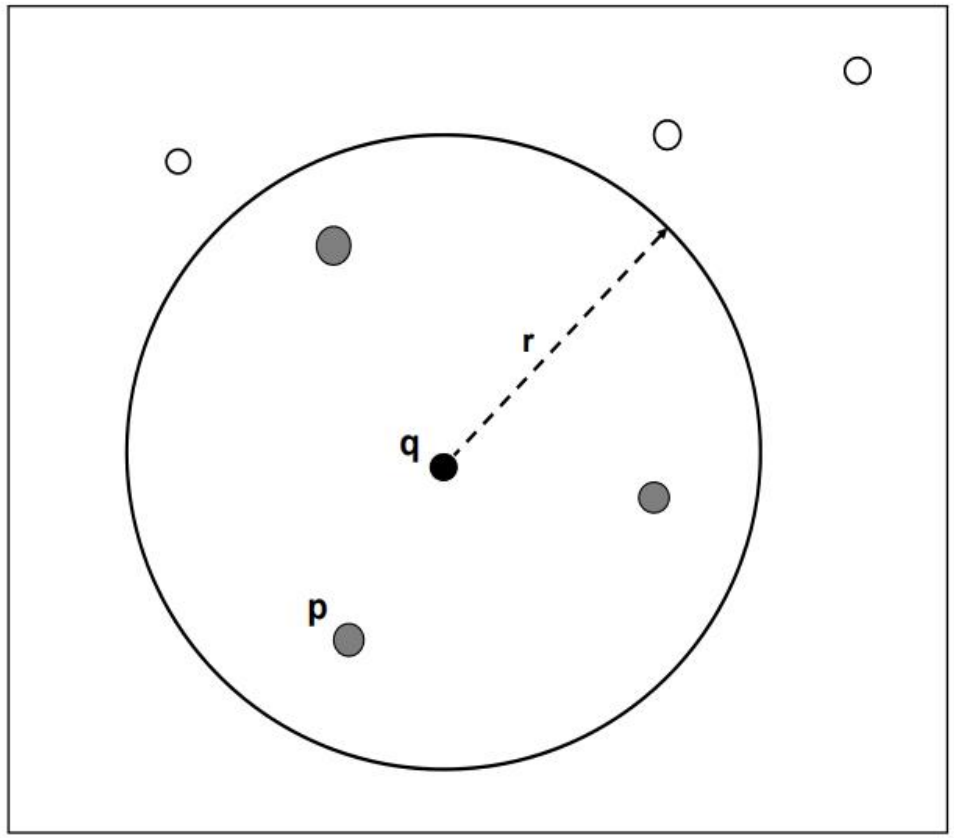


Figure 1: Near Neighbor problem, given q find p s.t. $D(q, p) \leq r$

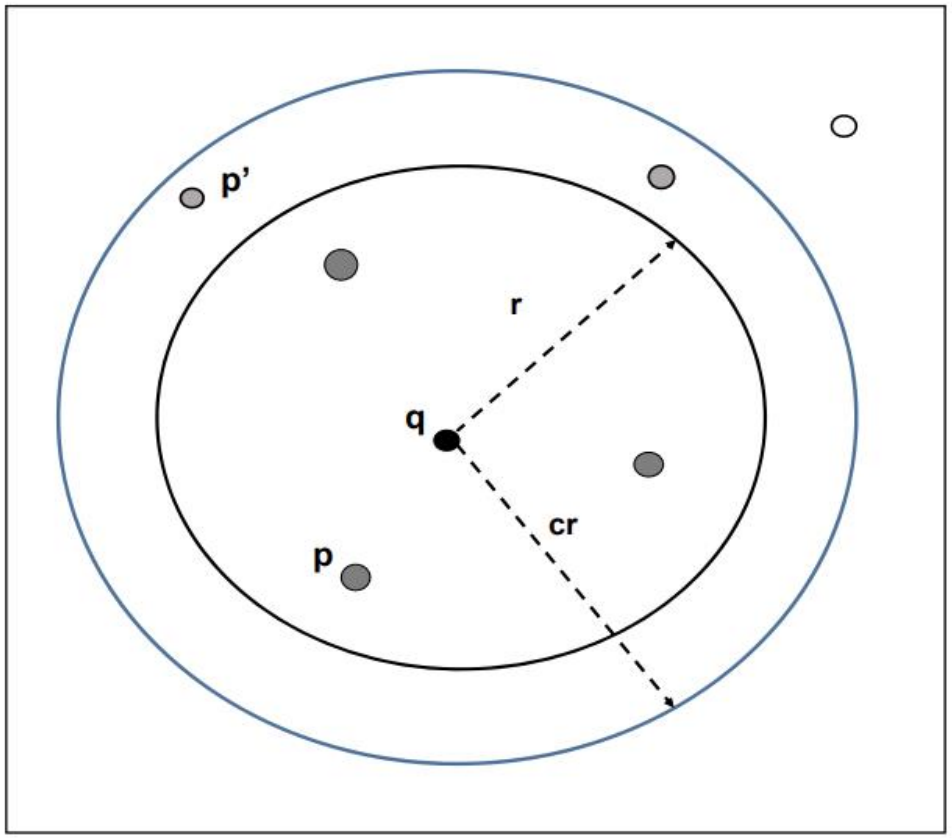


Figure 2: Approx Near Neighbor problem, given q find a p' s.t. $D(q, p') \leq cr$

(Approximately) Solve NNS using (c, r)-NNS

(Approximately) Solve NNS using (c, r)-NNS

> Guess $\min_{p \in X} d(p, q)$ up to a mult. factor $1 \pm \varepsilon$.

(Approximately) Solve NNS using (c,r)-NNS

- Guess $\min_{p \in X} d(p,q)$ up to a mult. factor $1 \pm \varepsilon$.
- Assume $\text{diam}(P) = \max_{p,p' \in X} d(p,p') \leq 1$

(Approximately) Solve NNS using (c,r)-NNS

- Guess $\min_{p \in X} d(p,q)$ up to a mult. factor $1 \pm \varepsilon$.
- Assume $\text{diam}(P) = \max_{p,p' \in X} d(p,p') \leq 1$
- Let $\delta =$ min dist. in X

(Approximately) Solve NNS using (c,r)-NNS

- Guess $\min_{p \in X} d(p,q)$ up to a mult. factor $1 \pm \varepsilon$.
- Assume $\text{diam}(P) = \max_{p,p' \in X} d(p,p') \leq 1$
- Let $\delta =$ min dist. in X, so $1/\delta =$ "bit-precision" of dataset X

(Approximately) Solve NNS using (c,r)-NNS

- Guess $\min_{p \in X} d(p,q)$ up to a mult. factor $1 \pm \varepsilon$.
- Assume $\text{diam}(P) = \max_{p,p' \in X} d(p,p') \leq 1$
- Let $\delta =$ min dist. in X, so $1/\delta =$ "bit-precision" of dataset X
- Next, solve $(c(1-\varepsilon), r)$-NNS for $r = \delta, (1+\varepsilon)\delta, (1+\varepsilon)^2\delta, \ldots, 1$

(Approximately) Solve NNS using (c,r)-NNS

- Guess $\min_{p \in X} d(p,q)$ up to a mult. factor $1 \pm \varepsilon$.
- Assume $\text{diam}(P) = \max_{p,p' \in X} d(p,p') \leq 1$
- Let $\delta =$ min dist. in X, so $1/\delta =$ "bit-precision" of dataset X
- Next, solve $(c(1-\varepsilon), r)$-NNS for $r = \delta, (1+\varepsilon)\delta, (1+\varepsilon)^2\delta, \ldots, 1$
- Report the minimal value of r for which we find a pt at dist $c(1-\varepsilon)$ of q.

Red^n to (c, r)-NNS (contd)

> This red^n imposes an additional $O(\log 1/\delta)$ overhead to the query time & memory

Red^n to (c, r)-NNS (contd)

> This red^n imposes an additional $O(\log 1/\delta)$ overhead to the query time & memory

(As we maintain a separate DS for each value of r above)

Red^n to (c, r)-NNS (contd)

- This red^n imposes an additional $O(\log 1/\delta)$ overhead to the query time & memory
  (As we maintain a separate DS for each value of r above)
- So, for the rest of the talk, we focus on (c, r)-NNS !

LSH Families

LSH Families

> Kind of hash family that is sensitive to distance

LSH Families

> Kind of hash family that is sensitive to distance
> Ideally, want a hash fn. that maps "close points" to the same value whp and maps "far points" to different values

LSH Families

> Kind of hash family that is sensitive to distance
> Ideally, want a hash fn. that maps "close points" to the same value whp and maps "far points" to different values. Formally,

$\mathcal{H} = \{h : X \to \mathbb{Z}\}$ is $(c, c \cdot r, p_1, p_2)$-LSH if

$\forall p, p' \in X$:

LSH Families

> Kind of hash family that is sensitive to distance
> Ideally, want a hash fn. that maps "close points" to the same value whp and maps "far points" to different values. Formally,

$\mathcal{H} = \{h : X \to \mathbb{Z}\}$ is (c, cr, p_1, p_2)-LSH if

$\forall p, p' \in X$:

① $d(p, p') \leq r \Rightarrow Pr[h(p) = h(p')] \geq p_1$

② $d(p, p') \geq cr \Rightarrow Pr[h(p) = h(p')] \leq p_2$

LSH Families

> Kind of hash family that is sensitive to distance
> Ideally, want a hash fn. that maps "close points" to the same value whp and maps "far points" to different values. Formally,

$\mathcal{H} = \{h : X \to \mathbb{Z}\}$ is $(c, c \cdot r, p_1, p_2)$-LSH if

$\forall p, p' \in X$:

① $d(p, p') \leq r \Rightarrow \Pr[h(p) = h(p')] \geq p_1$

② $d(p, p') \geq cr \Rightarrow \Pr[h(p) = h(p')] \leq p_2$

where the probabilities are over $h \sim \mathcal{H}$.

LSH Families (Contd)

$\mathcal{H} = \{h : X \to \mathbb{Z}\}$ is $(c, c \cdot r, p_1, p_2)$-LSH if

$\forall p, p' \in X$:

① $d(p, p') \leq r \Rightarrow Pr[h(p) = h(p')] \geq p_1$

② $d(p, p') \geq cr \Rightarrow Pr[h(p) = h(p')] \leq p_2$

where the probabilities are over $h \sim \mathcal{H}$.

> Ideally, we want $p_1 >> p_2$, but as we'll see, this highly depends on c.

LSH Families (Contd)

$\mathcal{H} = \{h : X \to \mathbb{Z}\}$ is (c, cr, p_1, p_2)-LSH if
$\forall p, p' \in X$:

① $d(p, p') \leq r \Rightarrow Pr[h(p) = h(p')] \geq p_1$

② $d(p, p') \geq cr \Rightarrow Pr[h(p) = h(p')] \leq p_2$

where the probabilities are over $h \sim \mathcal{H}$.

> Ideally, we want $p_1 \gg p_2$, but as we'll see, this highly depends on c.

> Main idea of IM98; even if $p_1 > p_2$ slightly, can boost $p_1 \approx 1$ & $p_2 \approx 1/n$.

LSH Family for binary vectors

LSH Family for binary vectors

> Say $U = \{0,1\}^d$, $dist(p,p') := \|p - p'\|_1$ ie. # of coordinates in which p, p' differ

LSH Family for binary vectors

> Say $U = \{0,1\}^d$, $dist(p,p') := \|p - p'\|_1$ ie. # of coordinates in which p, p' differ

> Consider $\mathcal{H} = \{h_i\}_{i=1}^{d}$ where

$$h_i(p) = p_i$$

LSH Family for binary vectors

> Say $U = \{0,1\}^d$, $dist(p,p') := \|p - p'\|_1$ ie. # of coordinates in which p, p' differ

> Consider $\mathcal{H} = \{h_i\}_{i=1}^{d}$ where

$$h_i(p) = p_i$$

> Observe: for $p, p' \in U$,

$$Pr[h(p) = h(p')] = \frac{\#\text{ agreeing coords}}{\text{total \# coords}} = \frac{d - \|p-p'\|_1}{d} = 1 - \frac{\|p-p'\|_1}{d}$$

LSH Family for binary vectors

> Say $U = \{0,1\}^d$, $dist(p,p') := \|p-p'\|_1$ i.e. # of coordinates in which p, p' differ

> Consider $\mathcal{H} = \{h_i\}_{i=1}^{d}$ where

$$h_i(p) = p_i$$

> Observe: for $p, p' \in U$,

$$Pr[h(p)=h(p')] = \frac{\#\text{ agreeing coords}}{\text{total \# coords}} = \frac{d - \|p-p'\|_1}{d} = 1 - \frac{\|p-p'\|_1}{d}$$

> So, $Pr[h(p)=h(p')] = \begin{cases} \geq 1 - \frac{r}{d} \approx e^{-r/d} & \text{if } d(p,p') \leq r \\ \leq 1 - \frac{cr}{d} \approx e^{-cr/d} & \text{if } d(p,p') \geq cr \end{cases}$

LSH Family for binary vectors

> Say $U = \{0,1\}^d$, $dist(p,p') := \|p-p'\|_1$ ie. # of coordinates in which p, p' differ

> Consider $\mathcal{H} = \{h_i\}_{i=1}^d$ where

$$h_i(p) = p_i$$

> Observe: for $p, p' \in U$,

$$Pr[h(p) = h(p')] = \frac{\#\text{ agreeing coords}}{\text{total \# coords}} = \frac{d - \|p-p'\|_1}{d} = 1 - \frac{\|p-p'\|_1}{d}$$

> So, $Pr[h(p)=h(p')] = \begin{cases} \geq 1 - \frac{r}{d} \approx e^{-r/d} & \text{if } d(p,p') \leq r \\ \leq 1 - \frac{cr}{d} \approx e^{-cr/d} & \text{if } d(p,p') \geq cr \end{cases}$

Conclusion: $\mathcal{H}$ is $(c, cr, e^{-r/d}, e^{-cr/d})$-LSH

LSH Family $\Rightarrow$ NNS - data structures!

LSH Family $\Rightarrow$ NNS - data structures!

> Suppose we had a (r, cr, p_1, p_2)-LSH fam. s.t. $p_1 \approx 1$, $p_2 \approx 0$. How to construct DS?

LSH Family $\Rightarrow$ NNS - data structures!

> Suppose we had a (r, cr, p_1, p_2)-LSH fam. s.t. $p_1 \approx 1$, $p_2 \approx 0$. How to construct DS?

> Choose $h \sim \mathcal{H}$ once & store $h(p)$ $\forall p \in X$.

LSH Family $\Rightarrow$ NNS - data structures!

> Suppose we had a (r, cr, p_1, p_2)-LSH fam. s.t. $p_1 \approx 1$, $p_2 \approx 0$. How to construct DS?

> Choose $h \sim \mathcal{H}$ once & store $h(p)$ $\forall p \in X$.

> Given $q \in U$, compute $h(q)$ & see if $\exists p \in X$ with $h(p) = h(q)$.

LSH Family $\Rightarrow$ NNS - data structures!

> Suppose we had a (r, cr, p_1, p_2)-LSH fam. s.t. $p_1 \approx 1$, $p_2 \approx 0$. How to construct DS?

> Choose $h \sim \mathcal{H}$ once & store $h(p)$ $\forall p \in X$.

> Given $q \in U$, compute $h(q)$ & see if $\exists p \in X$ with $h(p) = h(q)$. Can do this in $O(1)$ time using a hash table!

LSH Family $\Rightarrow$ NNS - data structures!

> Suppose we had a (r, cr, p_1, p_2)-LSH fam. s.t. $p_1 \approx 1$, $p_2 \approx 0$. How to construct DS?

> Choose $h \sim \mathcal{H}$ once & store $h(p)$ $\forall p \in X$.

> Given $q \in U$, compute $h(q)$ & see if $\exists p \in X$ with $h(p) = h(q)$. Can do this in $O(1)$ time using a hash table!

> If there is no such pt p, the whp. there is no pt at dist. $\leq cr$ of q.

LSH Family $\Rightarrow$ NNS - data structures!

> Suppose we had a (r, cr, p_1, p_2)-LSH fam. s.t. $p_1 \approx 1$, $p_2 \approx 0$. How to construct DS?

> Choose $h \sim \mathcal{H}$ uar & store $h(p)$ $\forall p \in X$.

> Given $q \in U$, compute $h(q)$ & see if $\exists p \in X$ w. $h(p) = h(q)$. Can do this in $O(1)$ time using a hash tbl!

> If there is no such pt p, the whp. there is no pt at dist. $\leq cr$ of q.

> So, only need to show: given (r, cr, p_1, p_2)-LSH fam with $p_1 > p_2$, we can "boost" it to get $p_1 \approx 1$, $p_2 \approx 0$.

LSH Parameter Boosting

> Two steps: first just make p_2 small

LSH Parameter Boosting

> Two steps: first just make p_2 small
> For this, take k independent hashes from $\mathcal{H}$ &
hash each $p \in X$ to

$$h(p) = (h_1(p), \ldots, h_k(p)) \in \mathbb{Z}^k$$

LSH Parameter Boosting

> Two steps: first just make p_2 small
> For this, take k independent hashes from $\mathcal{H}$ &
hash each $p \in X$ to

$$h(p) = (h_1(p), \ldots, h_k(p)) \in \mathbb{Z}^k$$

> Then $\forall p, p'$, $d(p, p') \geq cr \Rightarrow \Pr[h(p) = h(p')] \leq p_2^k$.

LSH Parameter Boosting

> Two steps: first just make p_2 small
> For this, take k independent hashes from $\mathcal{H}$ &
hash each $p \in X$ to

$$h(p) = (h_1(p), \ldots, h_k(p)) \in \mathbb{Z}^k$$

> Then $\forall p, p'$, $d(p,p') \geq cr \Rightarrow \Pr[h(p) = h(p')] \leq p_2^k$.

> Great, but this doesn't $\uparrow p_1$! In fact, 2 "close" pts are mapped to same vector only with prob. p_1^k.

Boosting $p_1 \uparrow$

Boosting $p_1 \uparrow$

> Choose ℓ ind. copies of above k-dim hash $f_1, \dots, f_\ell$!

Boosting $p_1 \uparrow$

- Choose l ind. copies of above k-dim hash $f_1, \ldots, f_l$!
- We show that if pts are close, then whp, at least one of the hashes agree.

Boosting $p_1 \uparrow$

- Choose l ind. copies of above k-dim hash $f_1, \ldots, f_l$!
- We show that if pts are close, then whp, at least one of the hashes agree.
- Assume $d(p,q) \leq r$. Then:

Boosting $p_1 \uparrow$

- Choose l ind. copies of above k-dim hash $f_1, \ldots, f_l$.
- We show that if pts are close, then whp, at least one of the hashes agree.
- Assume $d(p,q) \leq r$. Then:

$$\Pr[\exists i \mid f_i(p) = f_i(q)] = 1 - \Pr[\forall i \mid f_i(p) \neq f_i(q)]$$
$$= 1 - \Pr[f_i(p) \neq f_i(q)]^l$$
$$\geq 1 - (1 - p_1^k)^l$$

Boosting $p_1 \uparrow$

> Choose ℓ ind. copies of above k-dim hash $f_1, \ldots, f_\ell$.
> We show that if pts are close, then whp, at least one of the hashes agree.
> Assume $d(p,q) \le r$. Then:

$$\Pr[\exists i \mid f_i(p) = f_i(q)] = 1 - \Pr[\forall i \mid f_i(p) \neq f_i(q)]$$
$$= 1 - \Pr[f_i(p) \neq f_i(q)]^\ell$$
$$\geq 1 - (1 - p_1^k)^\ell$$

gotta make sure this is small!

The Algorithm

Algorithm 1 LSH Algorithm

Preprocessing:

Choose $k \cdot \ell$, $h_{1,1}, \ldots, h_{\ell,k}$ functions uniformly at random from $\mathcal{H}$.

Construct ℓ hash tables; for all $1 \leq i \leq \ell$ store $f_i(p) = (h_{i,1}(p), \ldots, h_{i,k}(p))$ for all $p \in P$ in the i-th has table.

Query(q)**:**

for $i = 1 \to \ell$ **do**

Compute $f_i(q)$.

Go over all points p where $f_i(p) = f_i(q)$. For all such points if $\text{dist}(p, q) \leq c \cdot r$, output p.

end for

Parameter Tuning

Parameter Tuning

→ Choose k s.t. $p_2^k = 1/n$.

Parameter Tuning

- Choose k s.t. $p_2^k = 1/n$.
- Assume $p_1 = p_2^{\rho}$ (recall $p_1 > p_2$, $\Rightarrow \rho < 1$).

Parameter Tuning

- Choose k s.t. $p_2^k = 1/n$.
- Assume $p_1 = p_2^{\rho}$ (recall $p_1 > p_2$, $\Rightarrow \rho < 1$).
- Choose $\ell = n^{\rho} \cdot \ln n$

Parameter Tuning

- Choose k s.t. $p_2^k = 1/n$.
- Assume $p_1 = p_2^{\rho}$ (recall $p_1 > p_2 \Rightarrow \rho < 1$).
- Choose $\ell = n^{\rho} \cdot \ln n$
- Fix a query $q \in U$. By lin. of exp., $\forall i$,

$$\Pr[\exists p : d(p,q) > cr, f_i(p) = f_i(q)] = n \cdot p_2^k \leq 1$$

Parameter Tuning

- Choose k s.t. $p_2^k = 1/n$.
- Assume $p_1 = p_2^{\rho}$ (recall $p_1 > p_2$, $\Rightarrow \rho < 1$).
- Choose $\ell = n^{\rho} \cdot \ln n$
- Fix a query $q \in U$. By lin. of exp., $\forall i$,

$$\Pr[\exists p : d(p,q) > cr, f_i(p) = f_i(q)] = n \cdot p_2^k \leq 1$$

- Summing over i, in exp, $O(\ell)$ far pts hash to same value as q for some i. $\Rightarrow O(\ell)$ overhead in q time.

Parameter Tuning (cont'd)

- If $d(p,q) \leq r$ for some $p \in X$, then,

$$\Pr[\exists i : f_i(p) = f_i(q)] \geq 1 - (1 - p_1^k)^l$$

$$= 1 - (1 - p_2^{ck})^l$$

$$= 1 - (1 - n^{-\rho})^l$$

$$\approx 1 - e^{ln^{-\rho}} = 1 - 1/n.$$

Parameter Tuning (cont'd)

> If $d(p,q) \le r$ for some $p \in X$, then,

$$\Pr[\exists i : f_i(p) = f_i(q)] \ge 1 - (1 - p_1^k)^\ell$$
$$= 1 - (1 - p_2^{\delta k})^\ell$$
$$= 1 - (1 - n^{-\delta})^\ell$$
$$\approx 1 - e^{\ell n^{-\delta}} = 1 - 1/n.$$

> Summary: for a pt p with $d(p,q) \le r$, our algo outputs p w.p. $\ge 1 - \frac{1}{n}$. In exp., $O(\ell \cdot d)$ overhead to examine $O(\ell)$ far pts with some hash value.

Space Analysis

Space Analysis

> Maintain $O(1)$ hash tables.

Space Analysis

- Maintain $O(l)$ hash tables.
- In each table, store $n = |X|$ hash values, each a k-dim vector.

Space Analysis

- Maintain $O(l)$ hash tables.
- In each table, store $n = |X|$ hash values, each a k-dim vector.
- Overall, $O(l \cdot n \cdot k) = O(n^{1+\rho} \log \frac{n}{p_2})$

Query Time Analysis

Query Time Analysis

- $O(lq)$ time to store $f_i(q)$ $\forall$ $i \in [l]$

Query Time Analysis

- $O(l_q)$ time to some $f_i(q)$ $\forall$ $i \in [l]$
- any candidate close pt p, spend $O(d)$ time to calculate $d(p, q)$.

Query Time Analysis

- $O(lq)$ time to compute $f_i(q)$ $\forall$ $i \in [l]$
- Any candidate close pt p, spend $O(d)$ time to calculate $d(p, q)$.
- Let $|O|$ = size of output = # of pts at dist $\leq cr$ from q

Query Time Analysis

- $O(lq)$ time to compute $f_i(q)$ $\forall$ $i \in [l]$
- Any candidate close pt p, spend $O(d)$ time to calculate $d(p, q)$.
- Let $|O|$ = size of output = # of pts at dist $\leq cr$ from q
- In exp., we examine $O(l)$ far pts that we don't output. So, overall, time is

$$O(d(l + |O|) + l \cdot k) = O(n^{\rho}(d + \log \tfrac{n}{p_2}) + |O|d)$$

Conclusion

> Memory $O(n^{1+\varepsilon})$, q-time $O(n^{\varepsilon})$

Conclusion

> Memory $O(n^{1+\rho})$, q-time $O(n^{\rho})$

> For the binary vector example, recall $p_1^{\rho} = p_2$ ⇒

$$\rho = \frac{\ln 1/p_1}{\ln 1/p_2} = \frac{r/d}{cr/d} = \frac{1}{c}$$

> For eg if $c=2$, need $O(n^{1.5})$ space for hash tables & $O(\sqrt{n})$ q-time.

> q-time & memory get significantly better as we relax c.

For other metrics

For other metrics

- For l', can 'reduce' to the binary vector case using a 'coordinate red^n' idea to get $\beta = 1/c$ as well.

For l_2:

- Use Johnson-Lindenstrauss to reduce dimension d to $\log n$.
- Assume all pts in X & query pts lie in S^d. Then

$h(x) = \text{sgn}(\langle g, x \rangle)$ where $g = (g_i)_{i=1}^{d}$ is sf.

$g_i \sim N(0,1)$

Picture

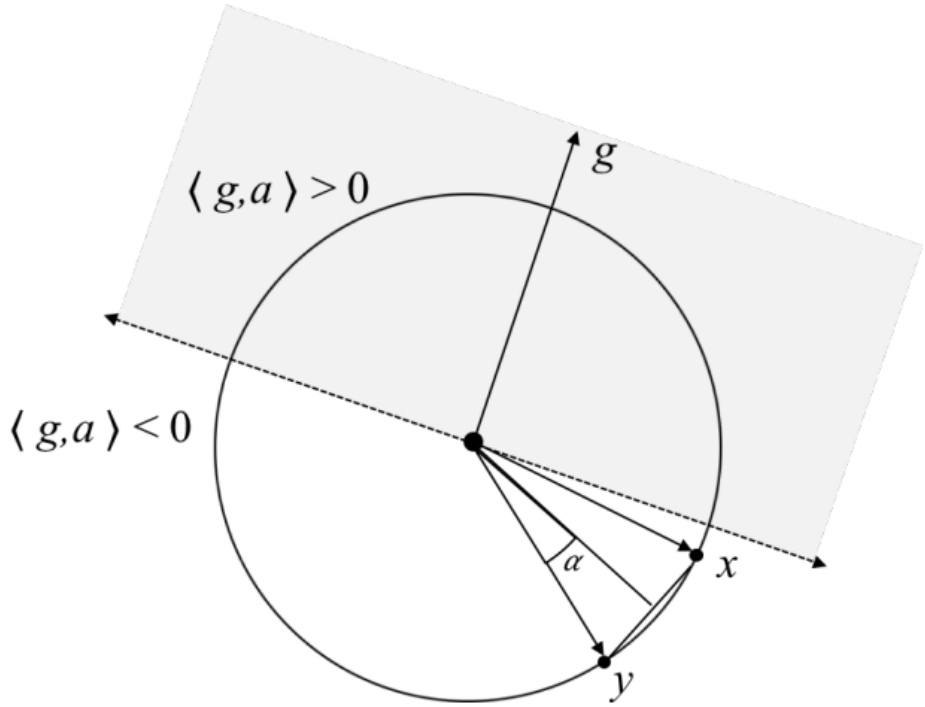


Figure 4: Our Euclidean locality sensitive hash function draws a random Gaussian vector g. Consider the hyperplane orthogonal to g and passing through $\{0\}^d$. All points above the hyperplane are hashed to 1 and all points below are hashed to -1.

> Overall, each hash fn. g in the LSH scheme is a bit vector constructed by taking the sgn of each entry in Πx, Π = random gaussian matrix.

> Overall, each hash fn. g in the LSH scheme is a bit vector constructed by taking the sgn of each entry in Πx, Π = random gaussian matrix.

> this also gives a $\rho = 1/c$ construction.

> Overall, each hash fn. g in the LSH scheme is a bit vector constructed by taking the sgn of each entry in Πx, Π = random gaussian matrix.

> this also gives a $\rho = 1/c$ construction.

> Possible to improve to (the optimal) $1/c^2$ using a more complex LSH scheme.

Recent Developments

(On board)

Thank You!