Approximate Nearest Neighbors Search in High Dimensions and Locality-Sensitive Hashing

PAPERS

Piotr Indyk, Rajeev Motwani: *Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality*, STOC 1998.

Eyal Kushilevitz, Rafail Ostrovsky, Yuval Rabani: *Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces.* SIAM J. Comput., 2000.

Mayur Datar, Nicole Immorlica, Piotr Indyk, Vahab S. Mirrokni: *Locality-Sensitive Hashing Scheme Based on p-Stable Distributions.* Symposium on Computational Geometry 2004.

Alexandr Andoni, Mayur Datar, Nicole Immorlica, Vahab S. Mirrokni, P. Indyk: *Locality-sensitive hashing using stable distributions.* In: *Nearest Neighbor Methods in Learning and Vision: Theory and Practice*, 2006.

Aneesh Sharma, Michael Wand

Overview

Overview

- Introduction
- Locality Sensitive Hashing (Aneesh)
- Hash Functions Based on *p*-Stable Distributions (Michael)

Overview

Overview

- Introduction
  - Nearest neighbor search problems
  - Higher dimensions
  - Johnson-Lindenstrauss lemma
- Locality Sensitive Hashing (Aneesh)
- Hash Functions Based on *p*-Stable Distributions (Michael)

Problem

Problem Statement

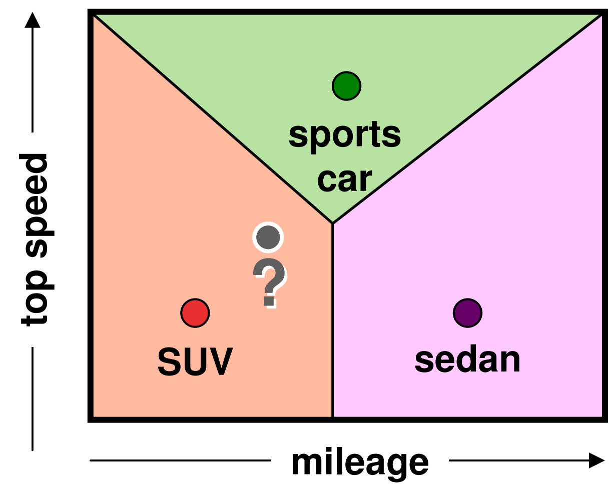
Today's Talks: NN-search in high dimensional spaces

- Given
  - Point set $P = \{p_1, \ldots, p_n\}$
  - a query point q
- Find
  - [ε-approximate] nearest neighbor to q from P
- Goal:
  - Sublinear query time
  - "Reasonable" preprocessing time & space
  - "Reasonable" growth in d (exponential not acceptable)

Applications

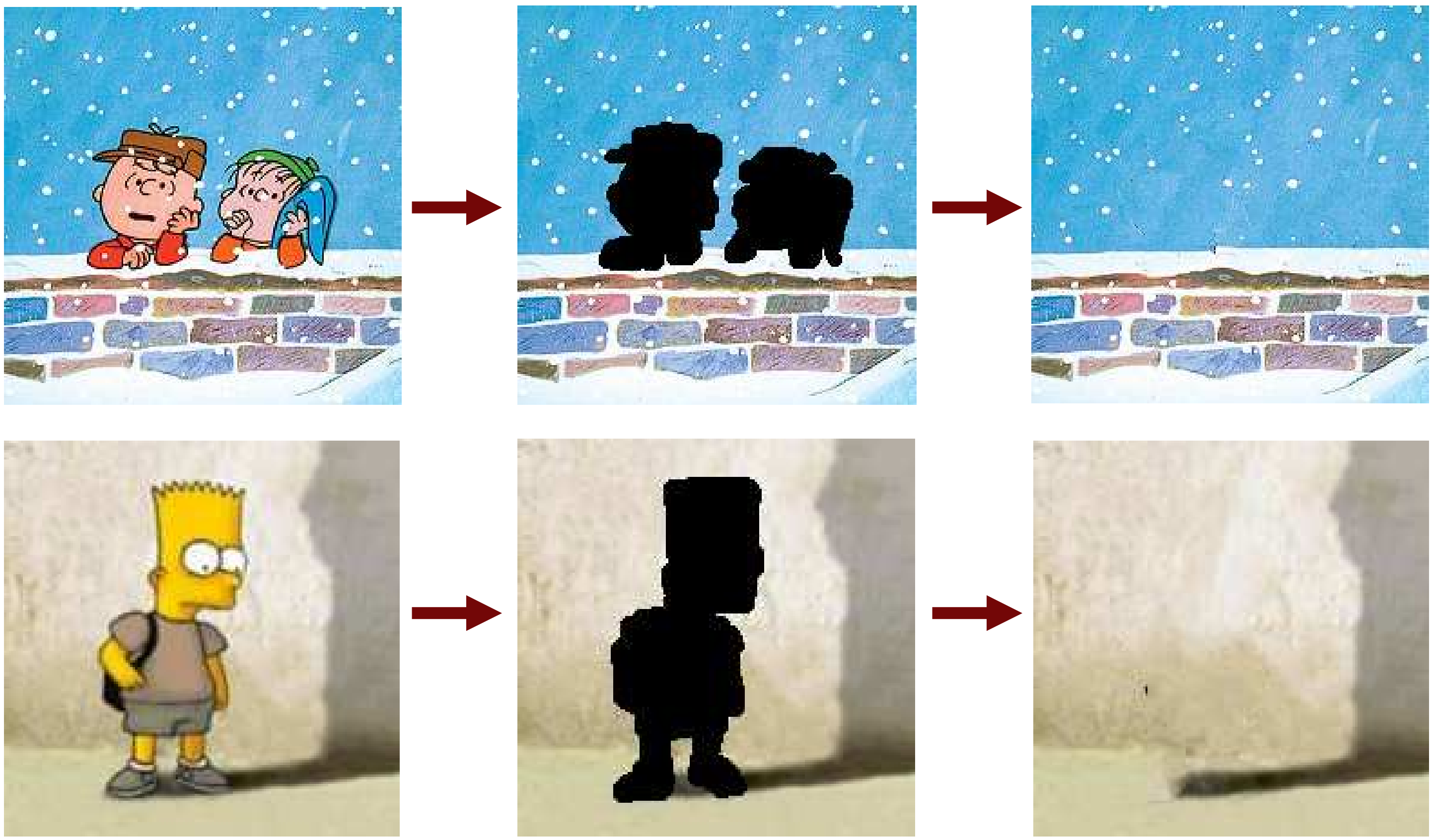
Example Application: Feature spaces

- Vectors $x \in \mathbb{R}^d$ represent characteristic features of objects
- There are often many features
- Use nearest neighbor rule for classification / recognition



Applications

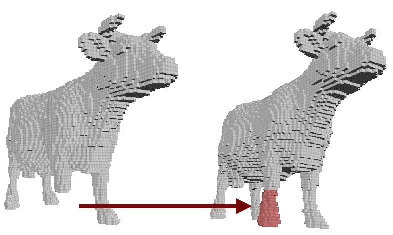
“Real World” Example: Image Completion



Applications

"Real World" Example: Image Completion

- Iteratively fill in pixels with best match (+ multi scale)
- Typically 5×5 … 9×9 neighborhoods, i.e.: dimension 25 … 81
- Performance limited by nearest neighbor search
- 3D version: dimension 81 … 729

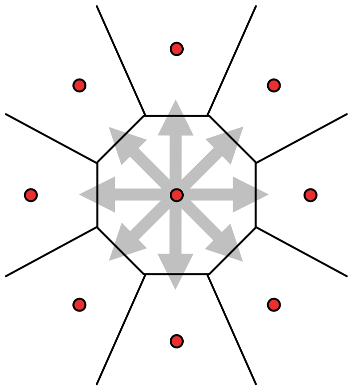


Higher Dimensions

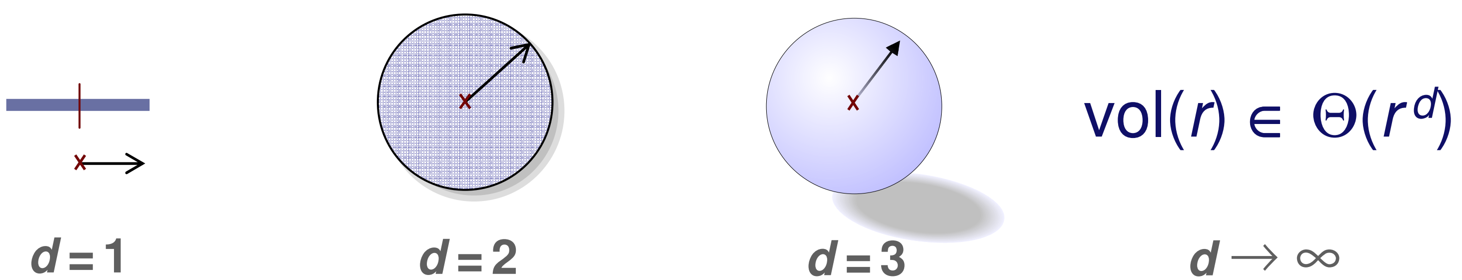
Higher Dimensions are Weird

Issues with High-Dimensional Spaces :

- *d*-dimensional space:
  d independent neighboring
  directions to each point



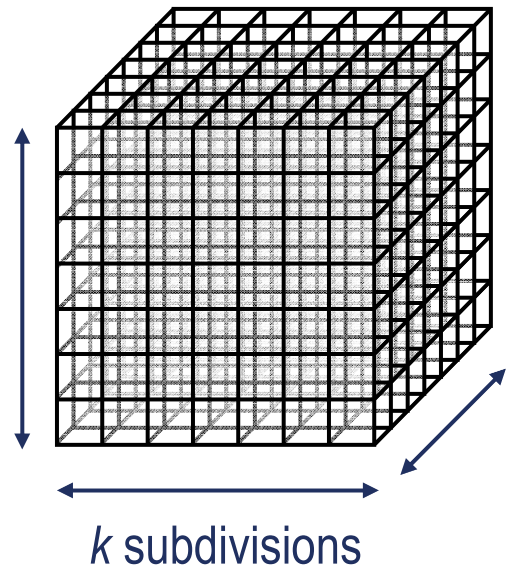
- Volume-distance ratio explodes



No Grid Tricks

Regular Subdivision Techniques Fail

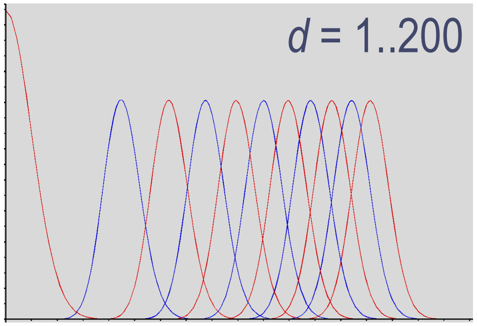
- Regular k-grids contain k^d cells
- The "grid trick" does not work
- Adaptive grids usually also do not help
- Conventional integration becomes infeasible ($\Rightarrow$ MC-approx.)
- Finite element function representation become infeasible

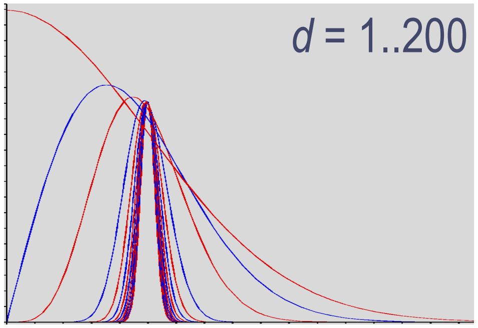


Higher Dimensions are Weird

More Weird Effects:

- Dart-throwing anomaly
  - Normal distributions gather prob.-mass in thin shells
  - [Bishop 95]





- Nearest neighbor ~ farthest neighbor
  - For unstructured points (e.g. iid-random)
  - Not true for certain classes of structured data
  - [Beyer et al. 99]

Johnson-Lindenstrauss Lemma

Johnson-Lindenstrauss Lemma

JL-Lemma: [Dasgupta et al. 99]

- Point set P in $\mathbb{R}^d$, $n := \#P$
- There is $f: \mathbb{R}^d \to \mathbb{R}^k$, $k \in O(\varepsilon^{-2} \ln n)$
  $(k \geq 4(\varepsilon^2/2 - \varepsilon^3/3)^{-1} \ln n)$
- …that preserves all inter-point distances up to a factor of $(1 + \varepsilon)$

Random orthogonal linear projection

works with probability $\geq (1\text{-}1/n)$

This means…

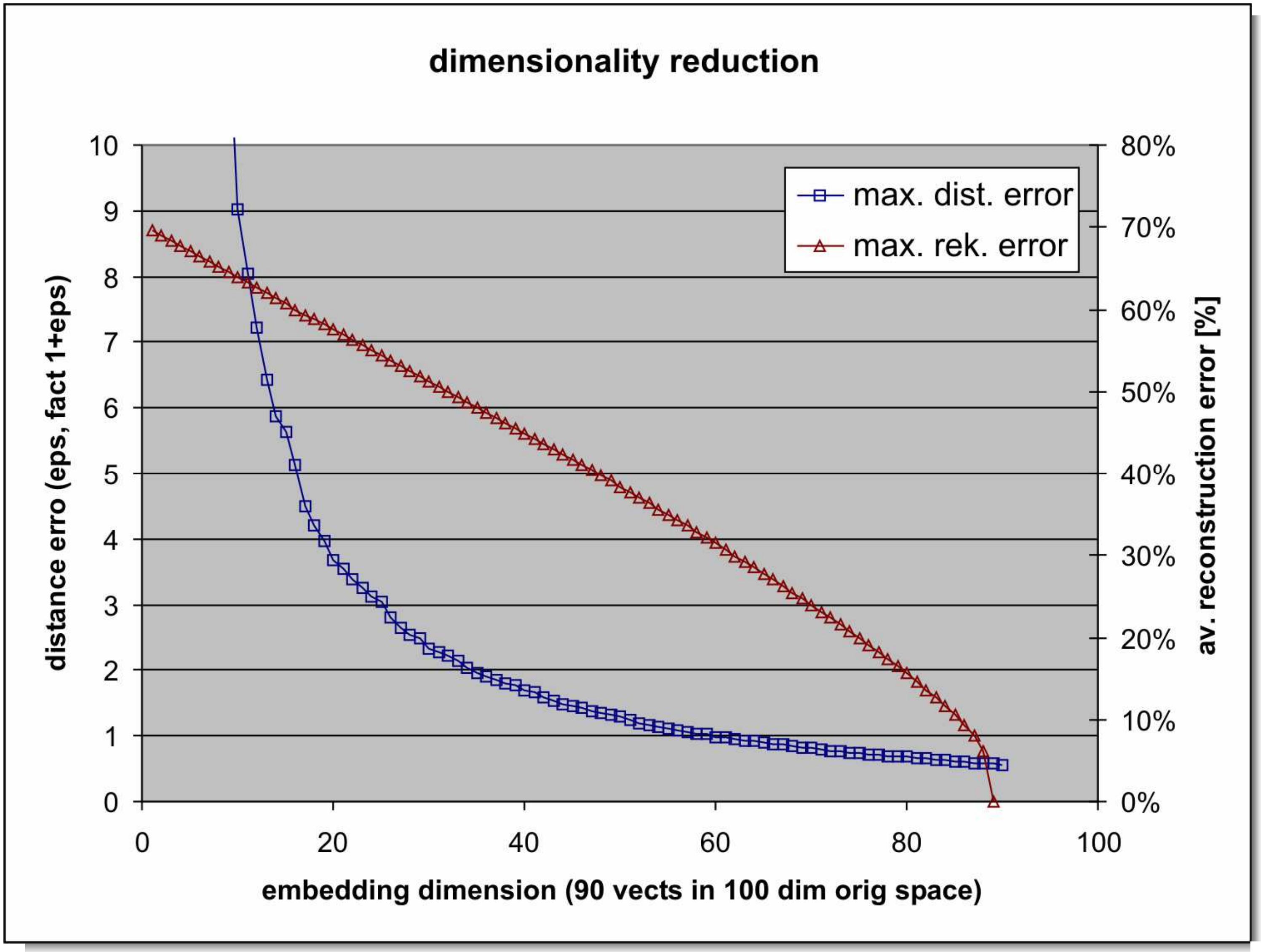
What Does the JL-Lemma Imply?

Pairwise distances in small point set *P*
(sub-exponential in *d*)
can be well-preserved in low-dimensional
embedding

What does it not say?

Does not imply that the points *themselves* are
well-represented (just the pairwise distances)

Experiment

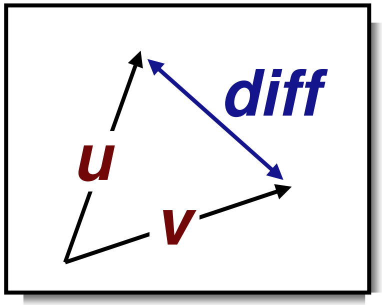


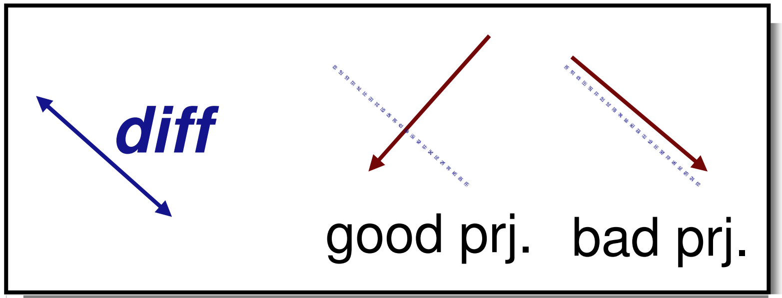
dimensionality reduction
max. dist. error
max. rek. error
distance erro (eps, fact 1+eps)
av. reconstruction error [%]
embedding dimension (90 vects in 100 dim orig space)
0
1
2
3
4
5
6
7
8
9
10
0%
10%
20%
30%
40%
50%
60%
70%
80%
0
20
40
60
80
100

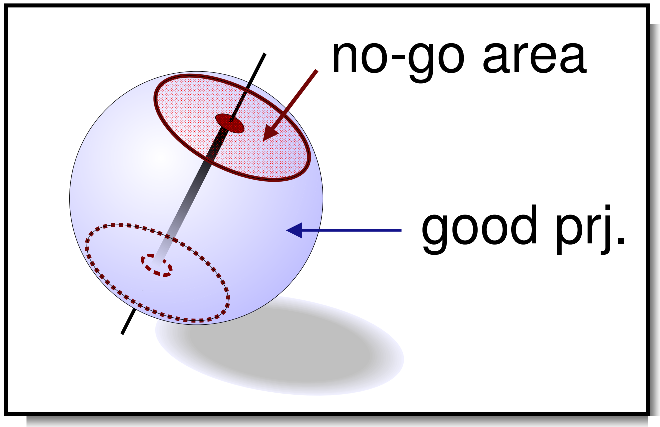
Intuition

Difference Vectors

- Normalize (relative error)
- Pole yields bad approximation
- Non-pole area much larger (high dimension)
- Need large number of poles (exponential in *d*)







Overview

Overview

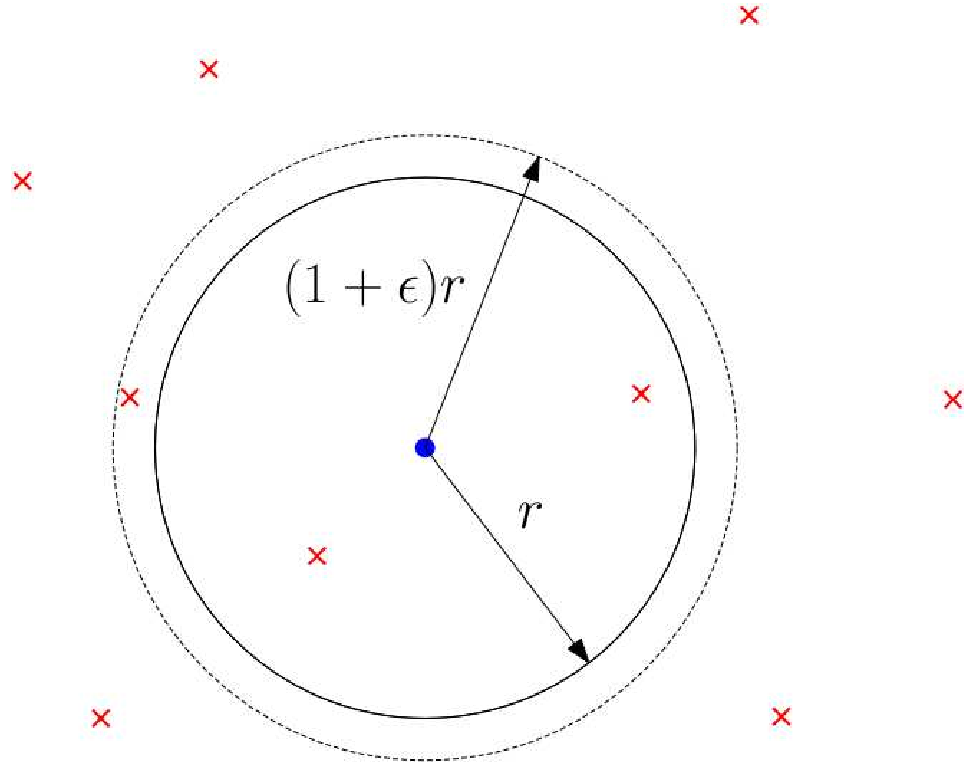
- Introduction
- Locality Sensitive Hashing
  - Approximate Nearest Neighbors
  - Big picture
  - LSH on unit hypercube
    - Setup
    - Main idea
    - Analysis
    - Results
- Hash Functions Based on *p*-Stable Distributions

Approximate Nearest Neighbors

ANN: Decision version

Input: *P*, *q*, *r*
Output:

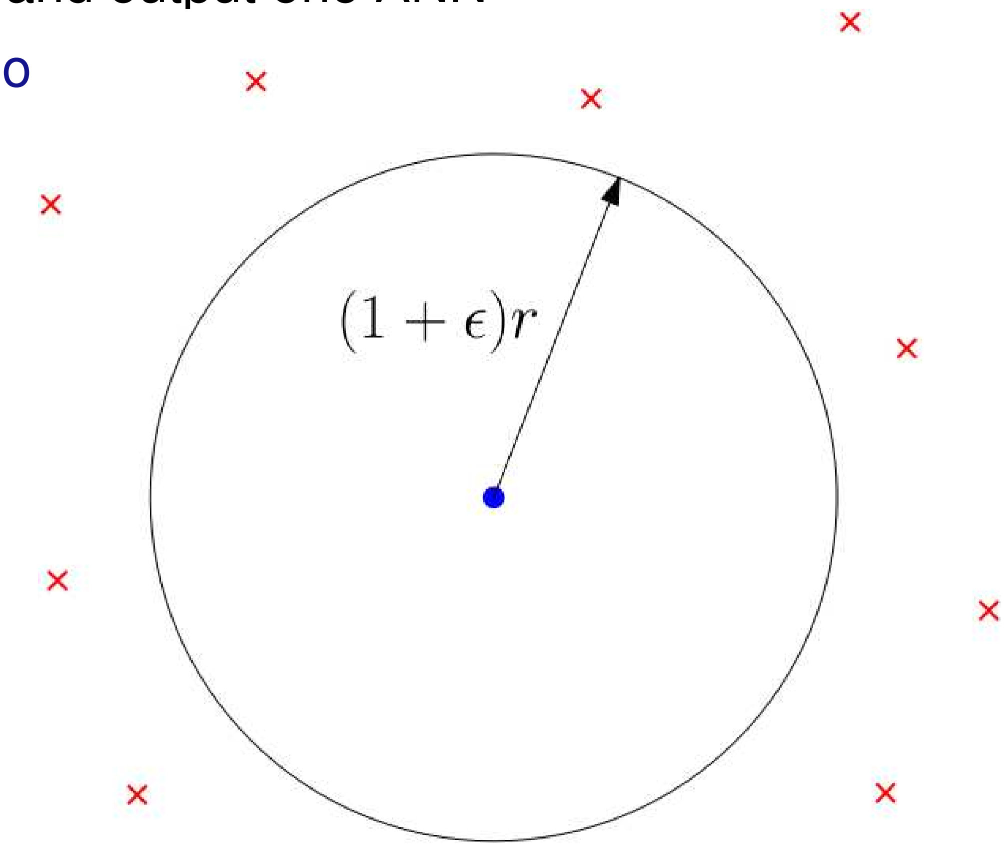
- If there is a NN, return yes and output one ANN
- If there is no ANN, return no
- Otherwise, return either



ANN: Decision version

Input: ***P*****,** ***q*****,** ***r***
Output:

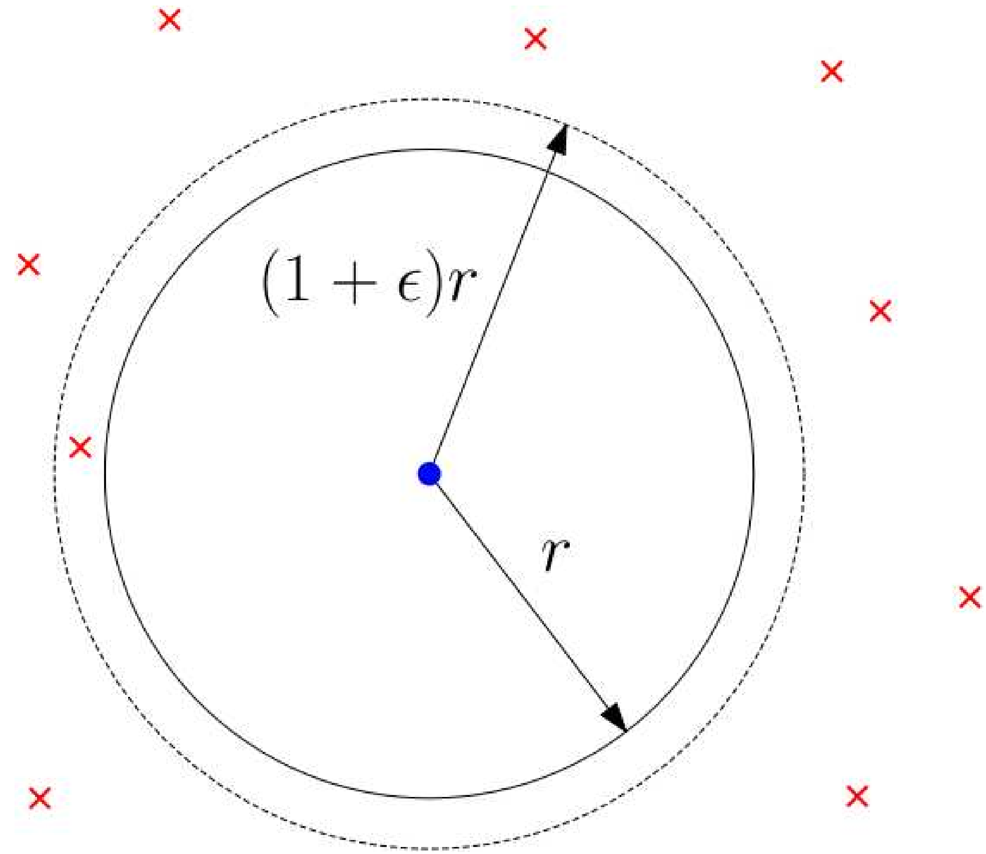
- If there is a NN, return yes and output one ANN
- If there is no ANN, return no
- Otherwise, return either



ANN: Decision version

Input: ***P***, ***q***, ***r***
Output:

- If there is a NN, return yes and output one ANN
- If there is no ANN, return no
- Otherwise, return either



ANN: Decision version

General ANN

$\Updownarrow$ **PLEB**

Decision version + Binary search

ANN: previous results

	Query time	Space used	Preprocessing time
Vornoi	$O\left(2^d \log n\right)$	$O\left(n^{d/2}\right)$	$O\left(n^{d/2}\right)$
Kd-tree	$O\left(2^d \log n\right)$	$O(n)$	$O(n \log n)$
LSH	$O\left(n^{\rho} \log n\right)$	$O\left(n^{1+\rho}\right)$	$O\left(n^{1+\rho} \log n\right)$

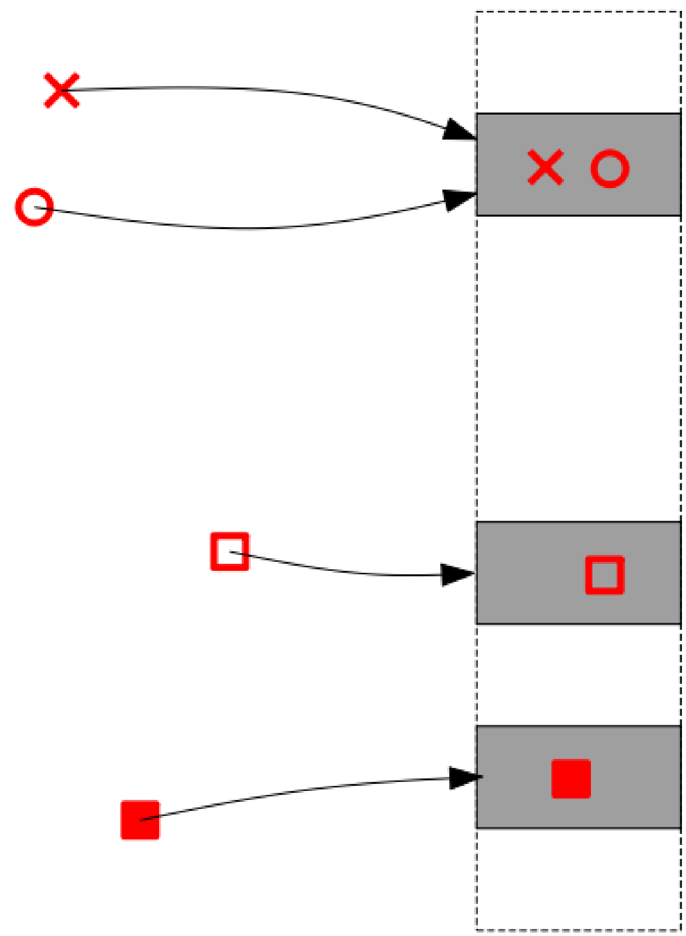
LSH: Big picture

Locality Sensitive Hashing

- **Remember: solving decision ANN**
- **Input:**
  - No. of points: n
  - Number of dimensions: d
  - Point set: P
  - Query point: q

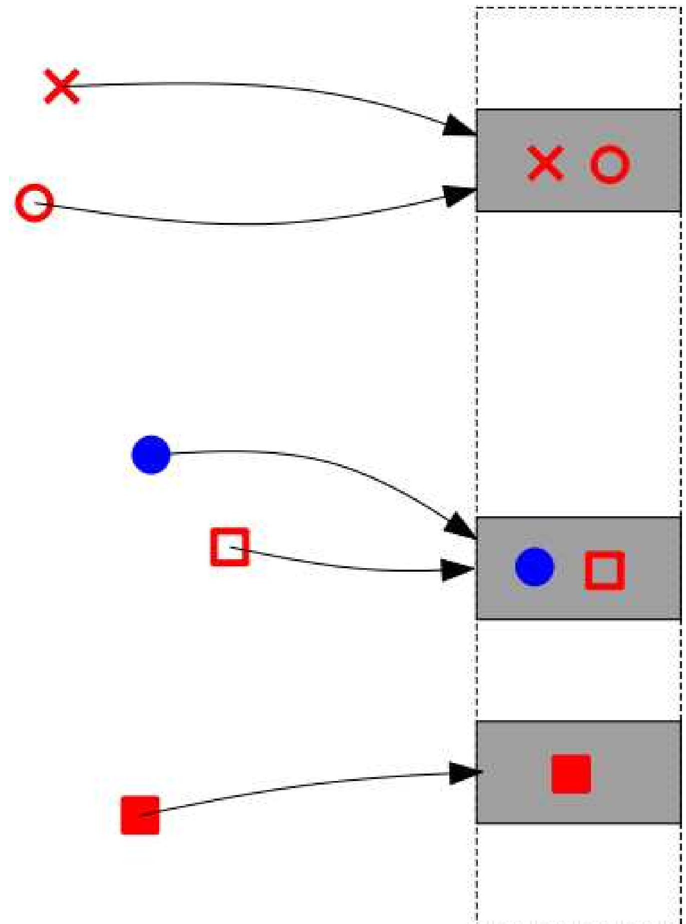
LSH: Big Picture

- **Family of hash functions:**
  - Close points to same buckets
  - Faraway points to different buckets
- **Choose a random function and hash *P***
- **Only store non-empty buckets**



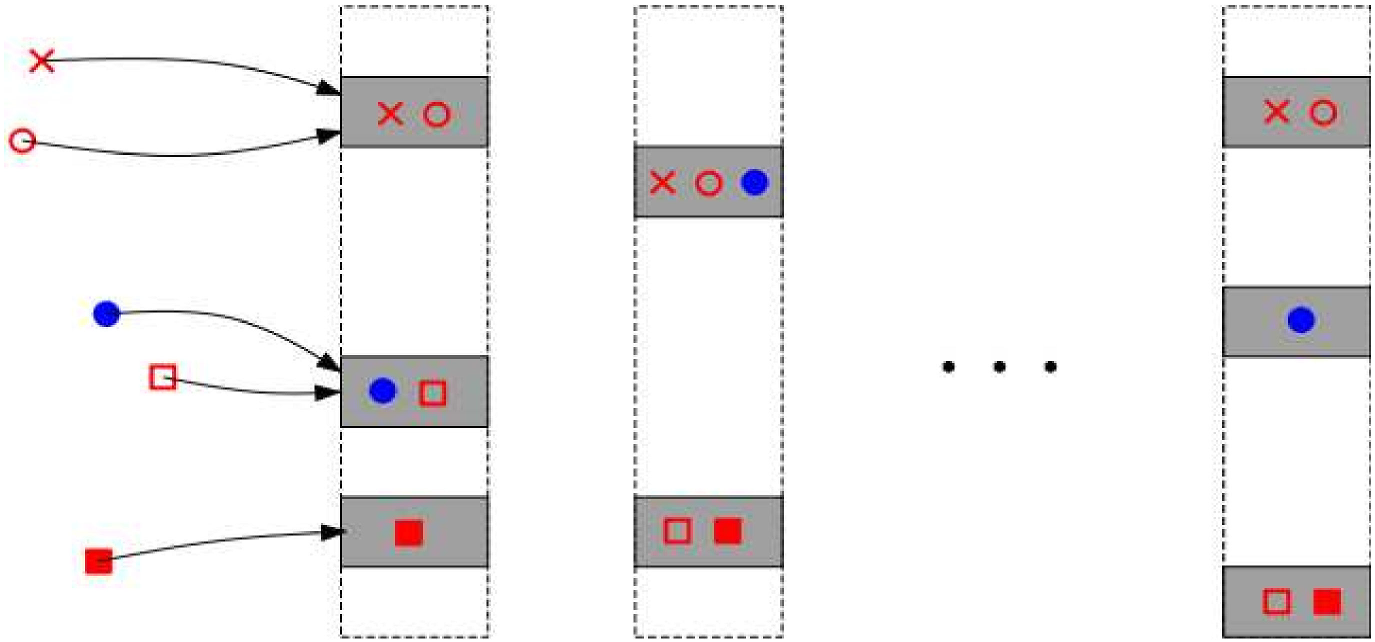
LSH: Big Picture

- **Hash *q* in the table**
- **Test every point in *q*'s bucket for ANN**
- **Problem:**
  - *q*'s bucket may be empty



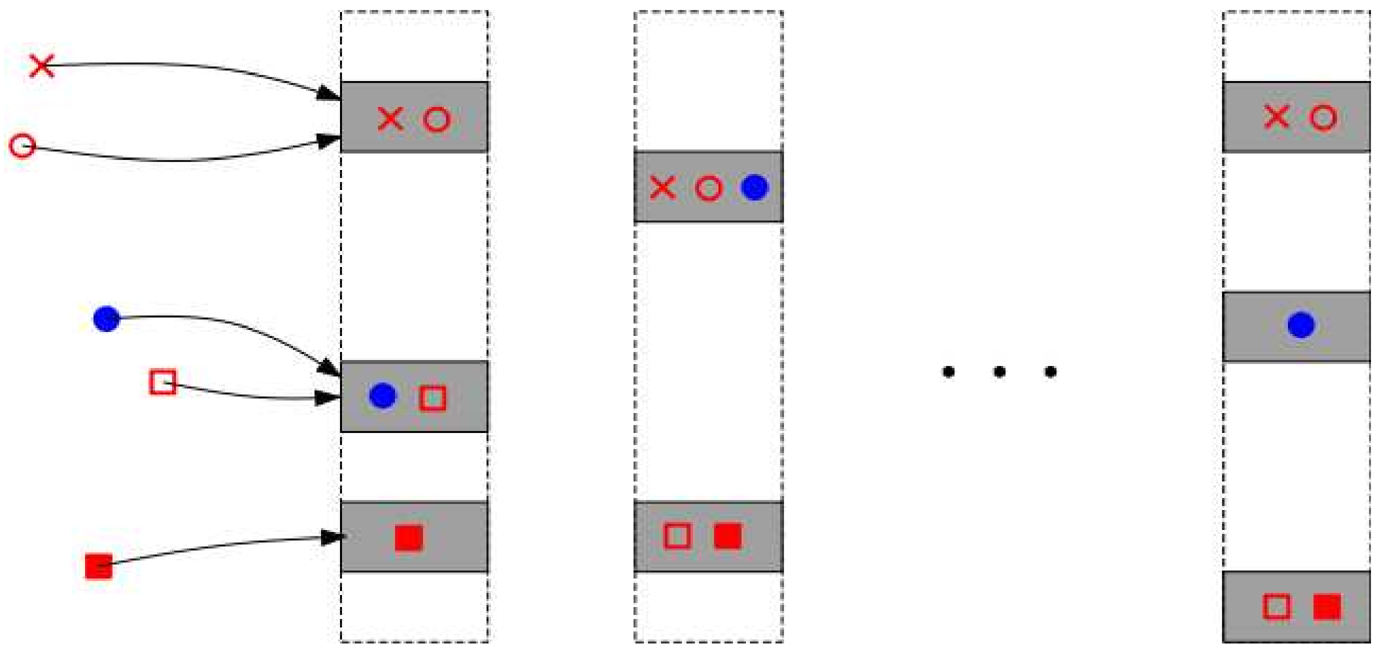
LSH: Big Picture

- **Solution:**
  - Use a number of hash tables!
  - We are done if any ANN is found



LSH: Big Picture

- **Problem:**
  - Poor resolution too many candidates!
  - Stop after reaching a limit, small probability



LSH: Big Picture

- **Want to find a hash function:**

$$\text{If } u \in B(q,r) \text{ then } \Pr\left[h(u) = h(q)\right] \geq \alpha$$

$$\text{If } u \notin B(q,R) \text{ then } \Pr\left[h(u) = h(q)\right] \leq \beta$$

$$r < R, \quad \alpha >> \beta$$

- ***h* is randomly picked from a family**
- **Choose**

$$R = r(1+\varepsilon)$$

LSH on unit Hypercube

Setup: unit hypercube

- **Points lie on hypercube: $H^d = \{0,1\}^d$**
- **Every point is a binary string**
- **Hamming distance (*r*):**
  - Number of different coordinates

$$u: \ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1$$
$$v: \ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0$$

Setup: unit hypercube

- **Points lie on hypercube: $H^d = \{0,1\}^d$**
- **Every point is a binary string**
- **Hamming distance (*r*):**
  - Number of different coordinates

$$
\begin{array}{ll}
u: & 0\ \boxed{0}\ 1\ 0\ \boxed{1}\ \boxed{1}\ 1\ 0\ \boxed{1} \\
v: & 0\ \boxed{1}\ 1\ 0\ \boxed{0}\ \boxed{0}\ 1\ 0\ \boxed{0}
\end{array}
$$

Main idea

Hash functions for hypercube

- **Define family *F*:**

$$\text{Given}: \text{Hypercube } H^d, \text{ point } b = (b_1, \ldots, b_d)$$

$$h \in F : \left\{ h_i(b) = b_i \,\middle|\, b = (b_1, \ldots, b_d) \in H^d, \text{ for } i = 1, \ldots, d \right\}$$

$$\alpha = 1 - \frac{r}{d}, \quad \beta = 1 - \frac{r(1+\varepsilon)}{d}$$

- Intuition: compare a random coordinate
- Called: $(r, r(1+\varepsilon), \alpha, \beta)$ - sensitive family

Hash functions for hypercube

- **Define family *G*:**

$$\text{Given}: b \in H^d, F$$

$$g \in G:$$

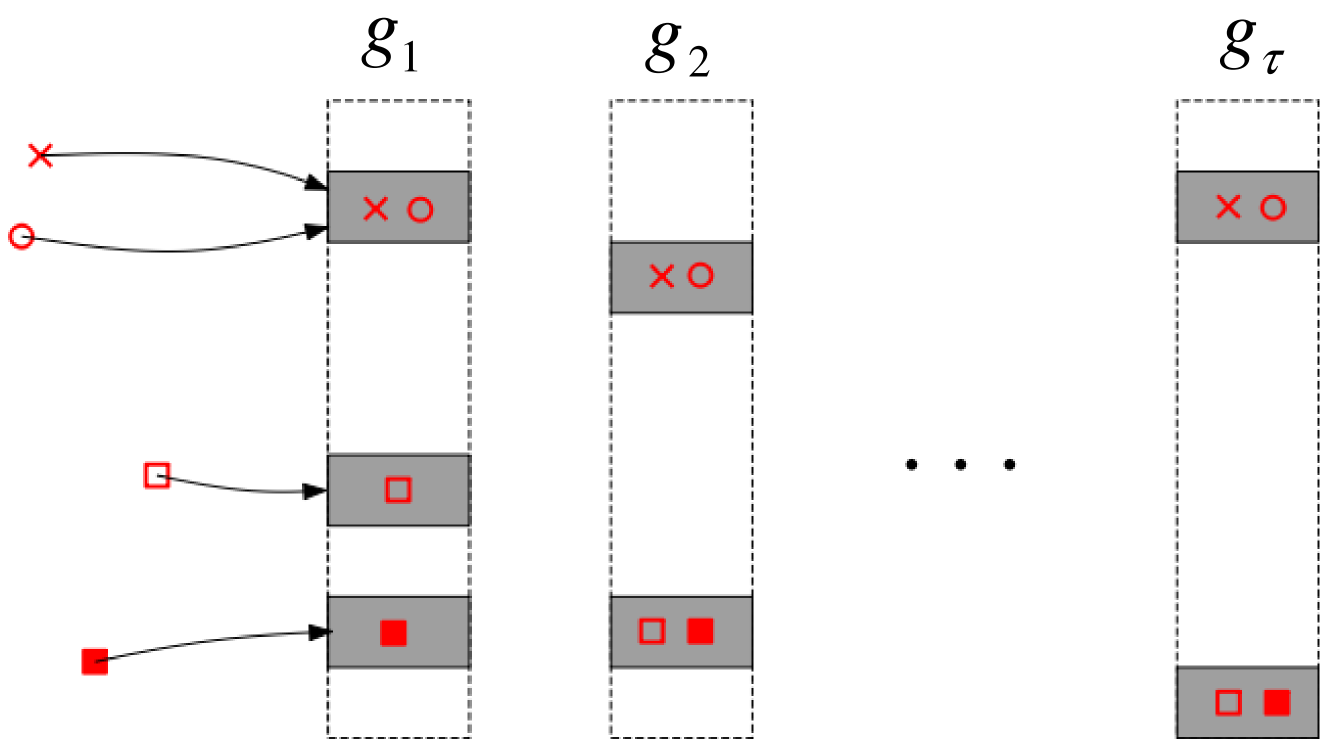
$$\left\{ g:\{0,1\}^d \to \{0,1\}^k \,\middle|\, g(b) = \left(h^1(b), \ldots, h^k(b) \right), \text{for } h^i \in F \right\}$$

$$\alpha' = \left(1 - \frac{r}{d}\right)^k = \alpha^k, \quad \beta' = \left(1 - \frac{r(1+\varepsilon)}{d}\right)^k = \beta^k$$

- Intuition: Compare *k* random coordinates
- Choose *k* later – logarithmic in *n* J-L lemma

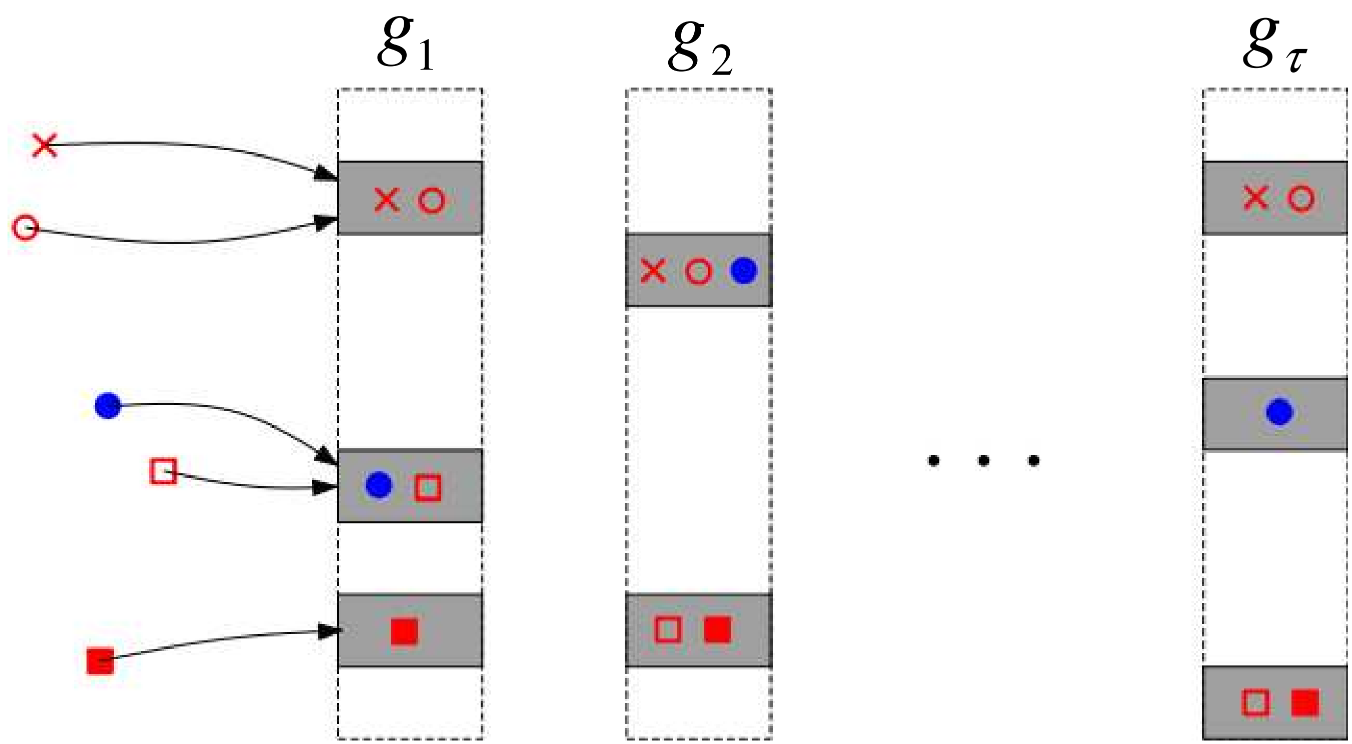
Constructing hash tables

- **Choose $g_1, \ldots, g_\tau$ uniformly at random from G**
  - Constructing τ hash tables, hash *P*
  - Will choose τ later



LSH: ANN algorithm

- **Hash *q* into each** $g_1, \ldots, g_\tau$
  - Check colliding nodes for ANN
  - Stop if more than 4τ collisions, return fail



Details…

Choosing parameters

- **Choose k and τ to ensure constant probability of:**
  - Finding an ANN if there is a NN
  - Few collisions $(< 4\tau)$ when there is no ANN

$$\text{Define} : \rho = \frac{\ln 1/\alpha}{\ln 1/\beta}$$

$$\text{Choose} : \mathrm{k} = \frac{\ln \mathrm{n}}{\ln 1/\beta} \;, \quad \tau = 2n^{\rho}$$

Analysis of LSH

- **Probability of finding an ANN if there is a NN**
  - Consider a point $p \in B(q,r)$ and a hash function $g_i \in G$

$$
\begin{aligned}
\Pr\left[g_i(p) = g_i(q)\right] &\geq \alpha^k \\
&= \alpha^{\frac{\ln n}{\ln 1/\beta}} \\
&= n^{-\frac{\ln 1/\alpha}{\ln 1/\beta}} \\
&= n^{-\rho}
\end{aligned}
$$

Analysis of LSH

- **Probability of finding an ANN if there is a NN**
  - Consider a point $p \in B(q,r)$ and a hash function $g_i \in G$

$$
\begin{aligned}
\Pr\left[g_i \text{ hashes } p \text{ and } q \text{ to different locations}\right] &\leq 1 - n^{-\rho} \\
\Pr\left[p \text{ and } q \text{ collide at least once in } \tau \text{ tables}\right] &\geq 1 - \left(1 - n^{-\rho}\right)^{\tau} \\
&\geq 1 - 1/e^2 \\
&> \frac{4}{5}
\end{aligned}
$$

Analysis of LSH

- **Probability of collision if there is no ANN**
  - Consider a point $p \notin B\big(q, r(1+\varepsilon)\big)$ and a hash function $g \in G$

$$\Pr\big[g(p) = g(q)\big] \le \beta^k$$

$$= \exp\left(\ln(\beta) . \frac{\ln n}{\ln 1/\beta} \right)$$

$$= \frac{1}{n}$$

Analysis of LSH

- **Probability of collision if there is no ANN**
  - Consider a point $p \notin B\big(q, r(1+\varepsilon)\big)$ and a hash function $g \in G$

$$
\begin{aligned}
\mathrm{E}[\text{collisions with } q \text{ in a table}] &\leq 1 \\
\mathrm{E}[\text{collisions with } q \text{ in } \tau \text{ tables}] &\leq \tau \\
\Pr[\geq 4\tau \text{ collisions}] &\leq \frac{\tau}{4\tau} = \frac{1}{4} \\
\Pr[< 4\tau \text{ collisions}] &\geq \frac{3}{4}
\end{aligned}
$$

Results

Complexity of LSH

- **Given:** $(r, r(1+\varepsilon), \alpha, \beta)$ - sensitive family for Hypercube
  - Can answer Decision-ANN with:

$$O\left(dn + n^{1+\rho}\right) \text{space}$$

$$O\left(dn^{\rho}\right) \text{query time}$$

  - Show:

$$\rho = \frac{\ln 1/\alpha}{\ln 1/\beta} = \frac{\ln\left(1 - \frac{\mathrm{r}}{\mathrm{d}}\right)}{\ln\left(1 - \frac{(1+\varepsilon)\mathrm{r}}{\mathrm{d}}\right)} \le \frac{1}{1+\varepsilon}$$

Complexity of LSH

- **Given:** $\left(r, r(1+\varepsilon), \alpha, \beta\right)$ - sensitive family for Hypercube
  - Can answer Decision-ANN with:

$$O\left(dn + n^{1+1/(1+\varepsilon)}\right) \text{ space}$$

$$O\left(dn^{1/(1+\varepsilon)}\right) \text{ query time}$$

Complexity of LSH

- **Can amplify success probability**
  - Build $O(\log n)$ structures
  - Can answer Decision-ANN with:

$$O\left(dn + n^{1+1/(1+\varepsilon)} \log n\right) \text{space}$$

$$O\left(dn^{1/(1+\varepsilon)} \log n\right) \text{query time}$$

Complexity of LSH

- **Can answer ANN on the Hypercube:**
  - Build $O\left(\varepsilon^{-1}\log n\right)$ structures with $r_i = (1+\varepsilon)^i$

$$O\left(dn + n^{1+1/(1+\varepsilon)}\varepsilon^{-1}\log^2 n\right)\text{space}$$

$$O\left(dn^{1/(1+\varepsilon)}\log\left(\varepsilon^{-1}\log n\right)\right)\text{query time}$$

LSH - Summary

- Randomized Monte-Carlo algorithm for ANN
- First truly sub-linear query time for ANN
- Need to examine only logarithmic number of coordinates
- Can be extended to any metric space if we can find a hash function for it!
- Easy to update dynamically
- Can reduce ANN in R^d to ANN on hypercube

Overview

Overview

- Introduction
- Locality Sensitive Hashing
- Hash Functions Based on *p*-Stable Distributions
  - The basic idea
  - The details (more formal)
  - Analysis, experimental results

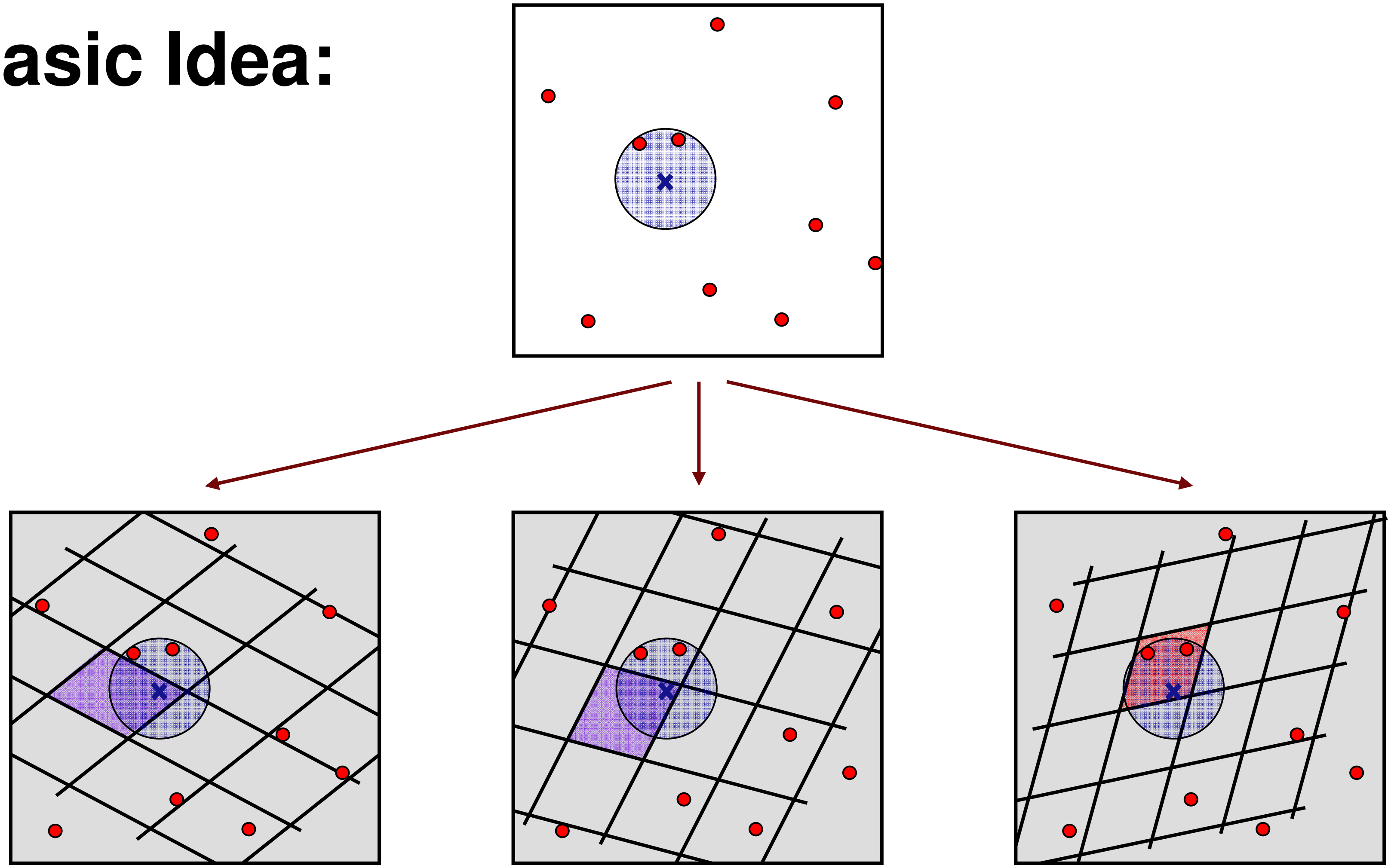
LSH by Random Projections

Idea:

- Hash function is a projection to a line
  of random orientation
- One composite hash function is a random grid
- Hashing buckets are grid cells
- Multiple grids are used for prob. amplification
- Jitter grid offset randomly (check only one cell)
- Double hashing: Do not store empty grid cells

LSH by Random Projections

Basic Idea:



LSH by Random Projections

Questions:

- What distribution should be used for the projection vectors?
- What is a good bucket size?
- Local sensitivity:
  - How many lines per grid?
  - How many hash grids overall?
  - Depends on sensitivity (as explained before)
- How efficient is this scheme?

The Details

p-Stable Distributions

Distribution for the Projection Vectors:

- Need to make the projection process formally accessible
- Mathematical tool: *p*-stable distributions

p-Stable Distributions

p-Stable Distributions:

A prob. distribution D is called p-stable $:\Leftrightarrow$

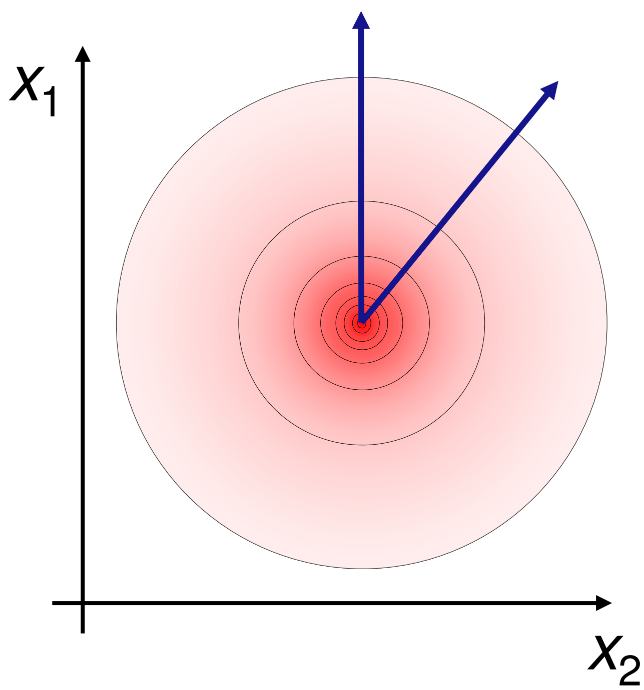
- For any $v_1, \dots, v_n \in \mathbb{R}$
- And i.i.d. random variables $X_1, \dots, X_n \sim D$

$\sum_i v_i X_i$ has the same distribution as $\left[\sum_i |v_i|^p\right]^{1/p} X$

where $X \sim D$

Gaussian Distribution

Gaussian Normal Distributions are 2-stable



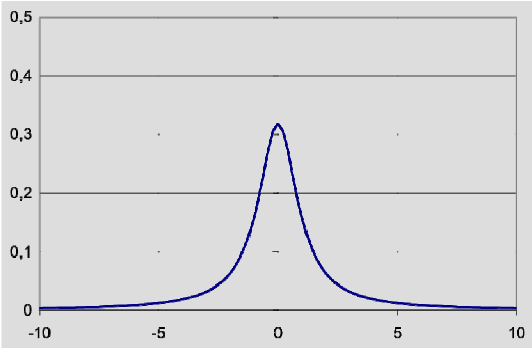
More General Distributions

Other distributions:

- Cauchy distribution $\frac{1}{\pi(1+x^2)}$

  is *1-stable*

  (must have infinite variance so that the central limit theorem is not violated)



- Distributions exists for $p \in (0,2]$
- No closed form, but can be sampled
- Sampling sufficient for LSH-algorithm

Projection

Projection Algorithm:

- Chose *p* according to metric of the space l_p
- Compute vector with entries according to a *p*-stable distribution
  [for example: Gaussian noise entries]
- Each vector v_i yields a hash function h_i
- Compute: $h_i(x) = \left\lfloor \frac{\langle v_i, x \rangle + b}{r} \right\rfloor$

random value $\in [0 \ldots r]$

bucket size

Locality Sensitive HF

Locality Sensitive Hash Functions

$H = \{h: S \to U\}$ is $(r_1, r_2, \alpha, \beta)$-sensitive :$\Leftrightarrow$

$$v \in B(q, r_1) \Rightarrow \Pr(\text{collision}(p, q)) \geq \alpha$$

$$v \notin B(q, r_2) \Rightarrow \Pr(\text{collision}(p, q)) \leq \beta$$

Performance

$$\rho = \frac{\ln \alpha}{\ln \beta}$$ (O($dn + n^{1+\rho}$) space, O(dn^{ρ}) query time)

Locality Sensitivity

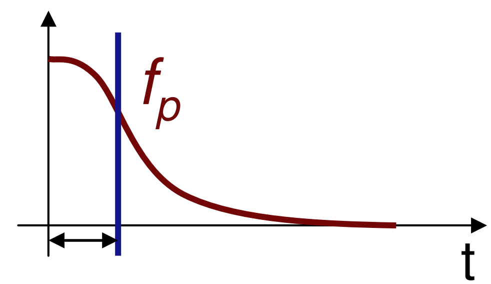
Computing the Locality "Sensitivity"

$$\left[h_i(x) = \left\lfloor \frac{\langle v_i, x \rangle + b}{r} \right\rfloor \right]$$

Distance $c = ||v_1 - v_2||_p$

cX-distributed, X from p-stable distr.

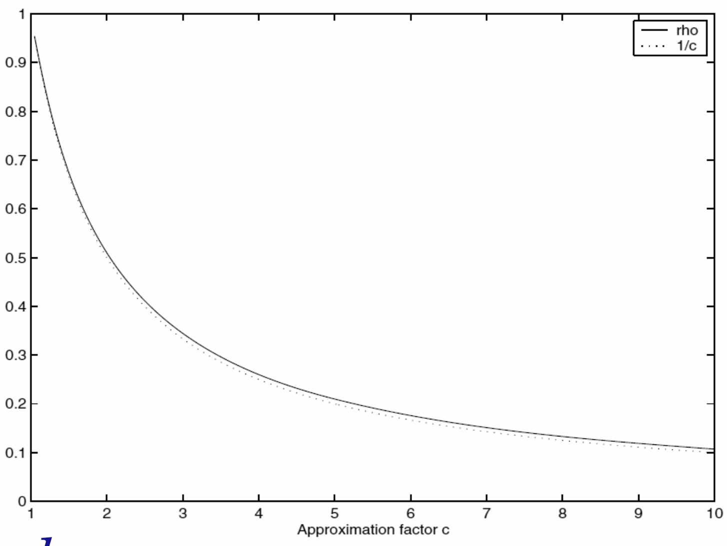
$$\underbrace{\Pr(collision)}_{=:p(c)} = \int_0^r \frac{1}{c} \underbrace{f_p\left(\frac{t}{c}\right)}_{abs.\ density} \underbrace{\left(1 - \frac{t}{r}\right)}_{hit\ bucket} dt$$



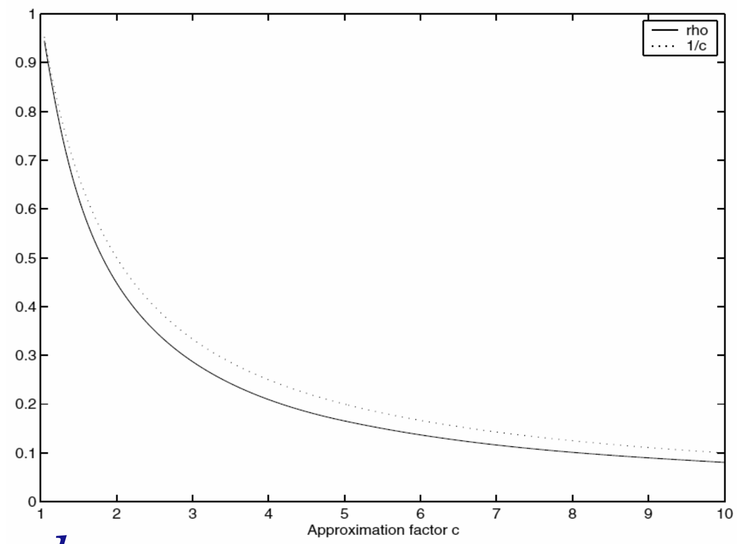
The constructed family of hash functions is $(r_1, r_2, \alpha, \beta)$-sensitive for

$\alpha = p(1)$, $\beta = p(c)$, $r_2/r_1 = c$

Numerical Computation



l_1 **[Datar et al. 04]**

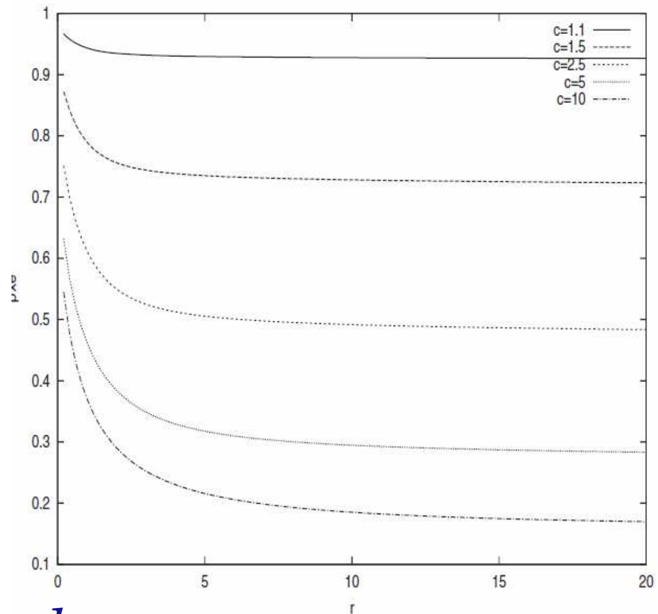


l_2 **[Datar et al. 04]**

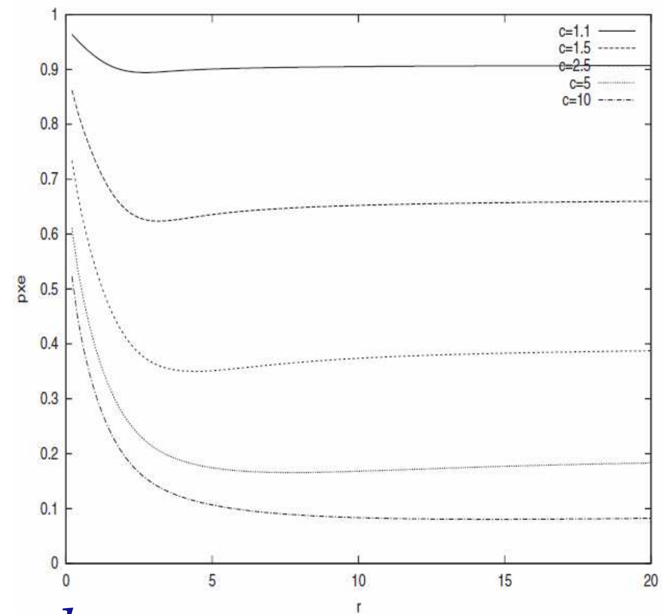
Numerical result: $\rho \sim 1/c = 1/(1+\varepsilon)$

$$\left[\rho = \frac{\ln \alpha}{\ln \beta},\ \text{O}(dn + n^{1+\rho}) \text{ space, } \text{O}(dn^{\rho}) \text{ query time} \right]$$

Numerical Computation



l_1 **[Datar et al. 04]**



l_2 **[Datar et al. 04]**

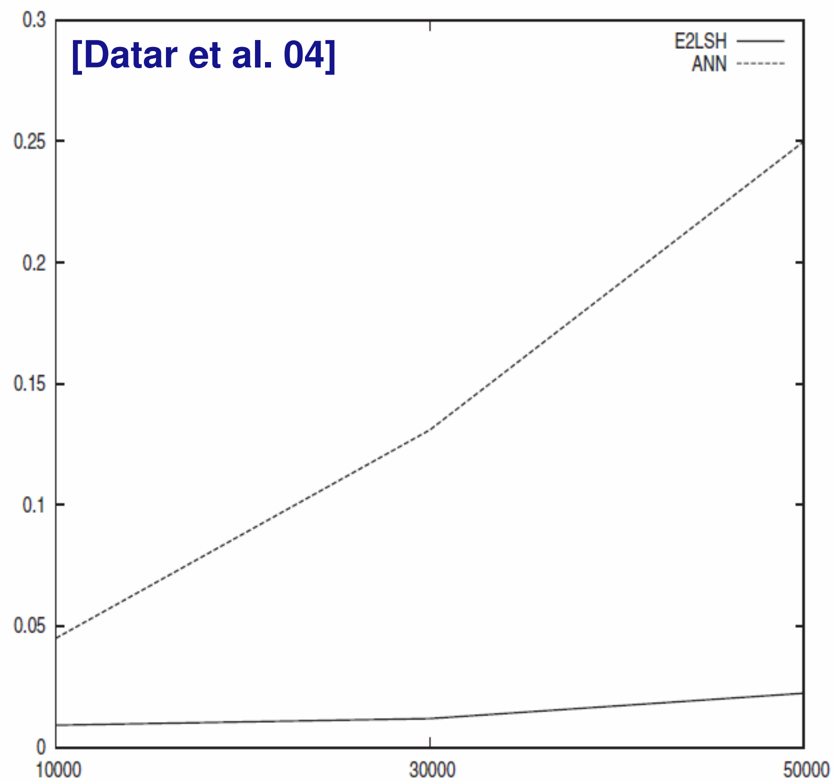
Width Parameter *r*

- Intuitively: In the range of ball radius
- Num. result: not too small (too large increases *k*)
- Practice: factor 4 (E2LSH manual)

Experimental Results

LSH vs. ANN

Comparison with ANN (Mount, Arya, kD/BBD-trees)



MNIST handwritten digits, $60\,000 \times 28^2$ pix ($d = 784$)

LSH vs. ANN

Remarks:

- ANN with $c = 10$ is comparably fast and 65% correct, but there are no guarantees [Indyk]
- LSH needs more memory:
  1.2GB vs. 360MB [Indyk]
- Empirically, LSH shows linear performance when forced to use linear memory [Goldstein et al. 05]
- Benchmark searches only for points in the data set, LSH is much slower for negative results
  [Goldstein et al. 05, report ~1.5 ord. of mag.]