Chapter 2: Nearest Neighbor Search: Theory

Ilya Razenshteyn **(CSAIL MIT)**

Nearest Neighbor Search (NNS)

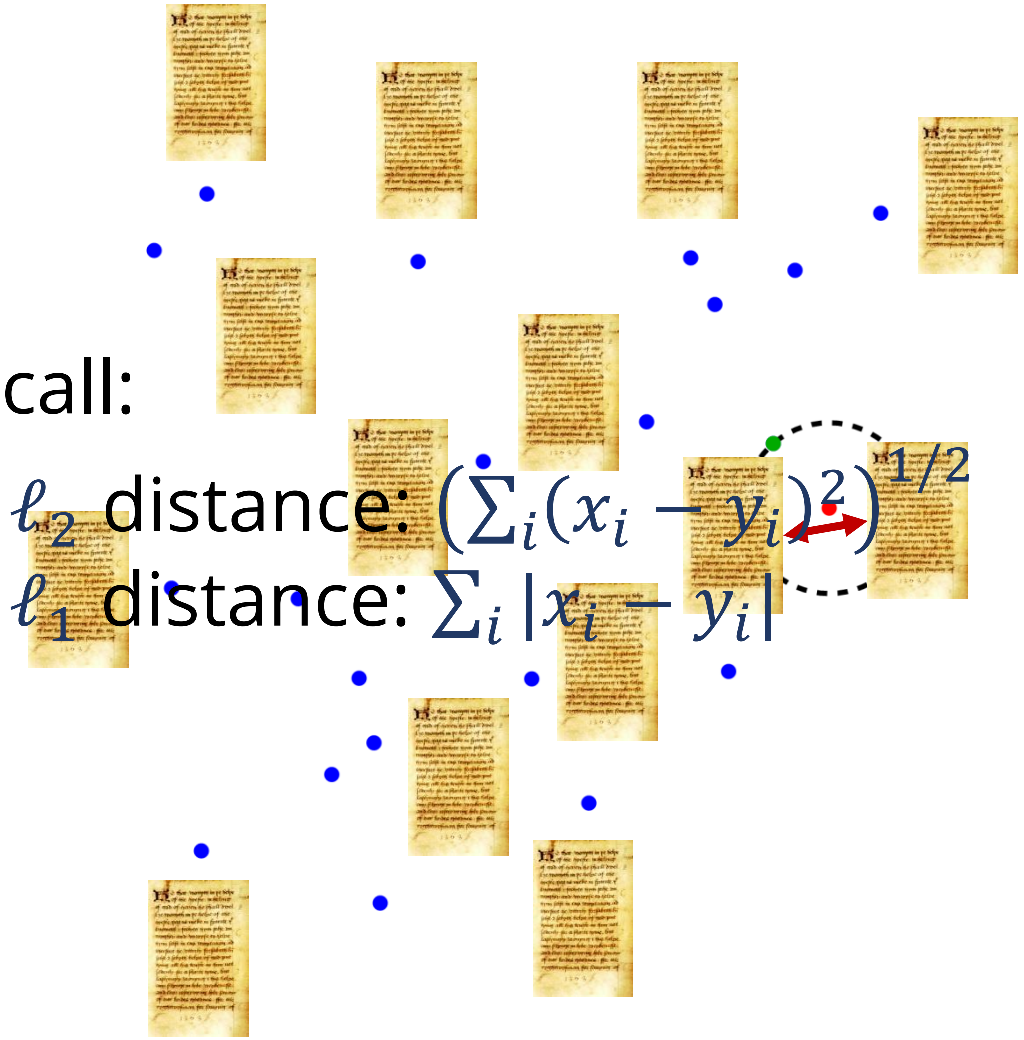
Dataset: n points in R^d
Query: a point in R^d
Goal: find the closest datapoint

Applications

- Finding similar texts/audio/images/proteins/users/etc.
- k-NN rule in machine learning
- Optimization
- Cryptanalysis (short vectors in lattices)
- Training neural networks
- …

Distances

- Euclidean/Cosine (ℓ_2), Manhattan/Hamming (ℓ_1)
- ℓ_∞, Jaccard similarity, edit distance, Earth Mover Distance (EMD), etc.

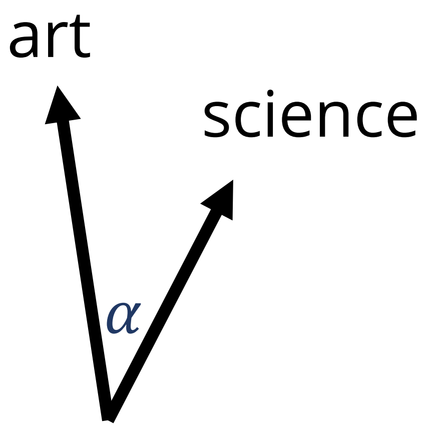


Recall:

- ℓ_2 distance: $\left(\sum_i (x_i - y_i)^2\right)^{1/2}$
- ℓ_1 distance: $\sum_i |x_i - y_i|$

An example

- Word embeddings
  - Vectors that capture semantic similarity between words
- GloVe **[Pennigton, Socher, Manning 2014]**
  - Ten nearest neighbors for “algorithms”?

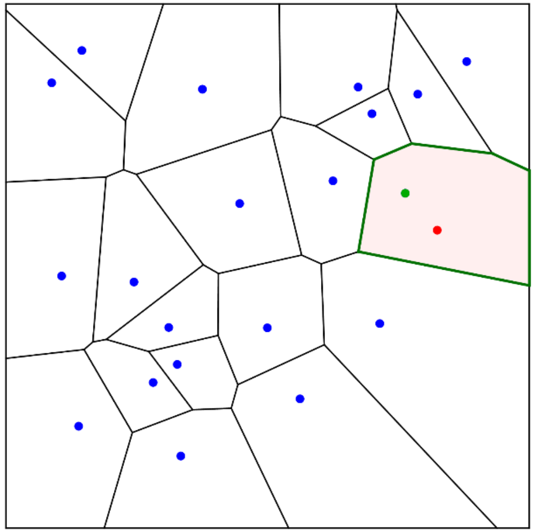


algorithm
optimization
computation
computational
implementations
probabilistic
deterministic
architectures
heuristics
methods

Setup

- Algorithm gets to know the dataset in advance
- **Preprocess** to be able to answer queries **quickly**
  - Improve upon the **linear scan**
- Main parameters: **space, query time,** preprocessing time
- **Remark:** queries *do not* belong to the dataset

Curse of dimensionality



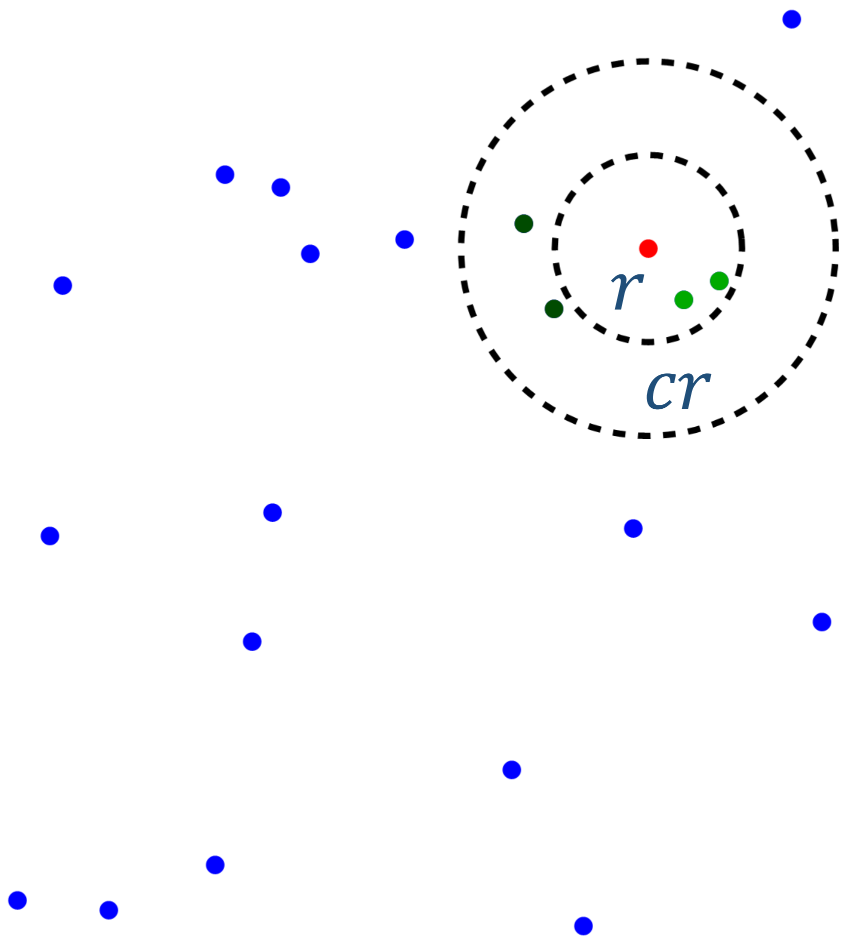
NNS becomes hard in high dimensions!

Method	Space	Query time
Linear scan	$O(dn)$ ☺	$O(dn)$ ☹
Full indexing	$n^{O(d)}$ ☹	$\mathrm{poly}(d, \log n)$ ☺

Approximate NNS

Dataset: n points in R^d
Query: a point in R^d
Goal: find a data point within factor of c from the closest

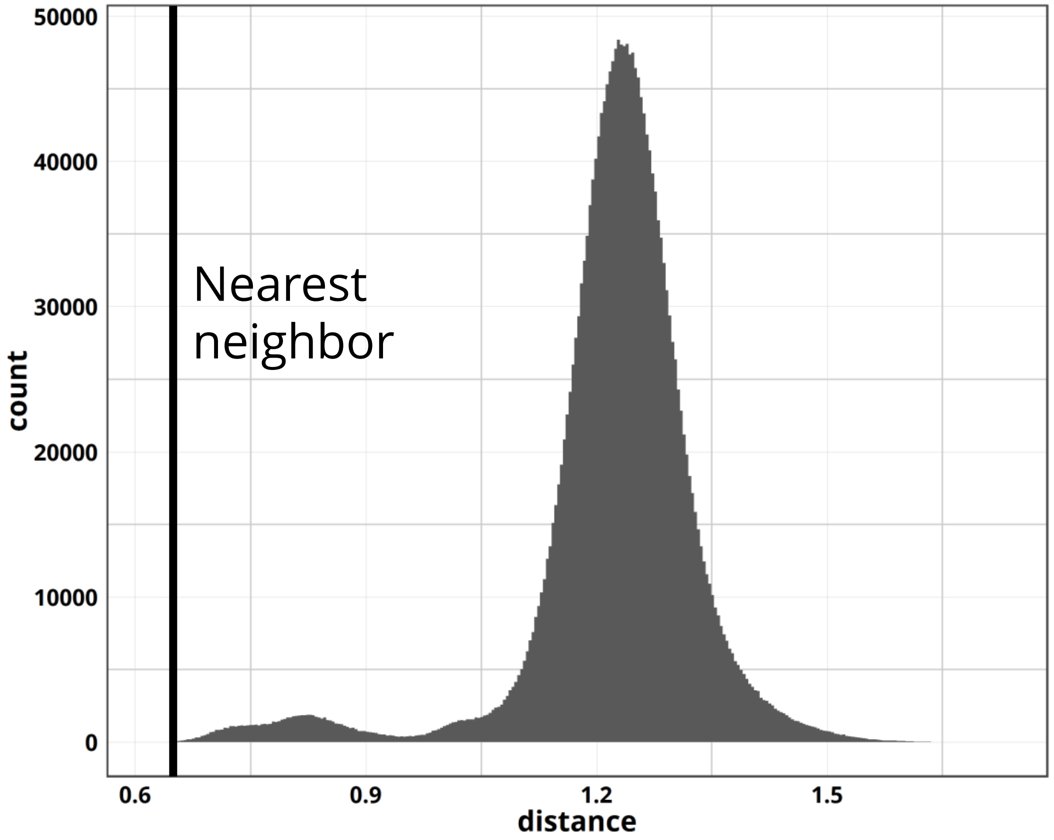
Additional data
- Approximation $c > 1$



In practice

- Want **exact** nearest neighbors ($c = 1$)
- Nearest neighbor is much closer than *most* of the data points
- The algorithms work under this "gap" assumption as well

GloVe word embeddings
[Pennigton, Socher, Manning 2014]



Related work

- (Mild) exponential dependence on d

[Arya, Mount 1993], [Clarkson 1994], [Arya, Mount, Netanyahu, Silverman, Wu 1998], [Kleinberg 1997], [Har-Peled 2002], [Arya, Fonseca, Mount 2011], …

- Polynomial dependence on d

[Kushilevitz, Ostrovsky, Rabani 1998], [Indyk, Motwani 1998], [Indyk 1998, 2001, 2002, 2004], [Gionis, Indyk, Motwani 1999], [Charikar 2002], [Datar, Immorlica, Indyk, Mirrokni 2004], [Chakrabarti, Regev 2004], [Panigrahy 2006], [Ailon, Chazelle 2006], [Andoni, Indyk 2006], [Andoni, Indyk, Nguyen, R 2014], [Bartal, Gottlieb 2014], [Kapralov 2015], [Andoni, R 2015], [Pagh 2016], [Becker, Ducas, Gama, Laarhoven 2016], [Christiani 2017], [Andoni, Laarhoven, R, Waingarten 2017], [Andoni, R, Shekel-Nosatzki 2017], [Andoni, Nguyen, Nikolov, R, Waingarten 2017], [Andoni, Nikolov, R, Waingarten 2017]

Plan

- ANN for Hamming distance (ℓ_1 on $\{0,1\}^d$)
  - Simple, classic algorithm from **[Indyk, Motwani 1998]**, will see the full analysis
  - Locality-Sensitive Hashing (LSH)
  - Space $O(n^{1+1/c} + nd)$, query time $O(dn^{1/c})$
- ANN for Euclidean distance (ℓ_2 on R^d)
  - Algorithm from **[Andoni, Laarhoven, R, Waingarten 2017]**
  - Smooth "optimal" trade-off between space and query time
  - Yields better results for Hamming as well
  - Not so simple, but modular

Hamming distance

Hamming distance between $x, y \in \{0, 1\}^d$:
number of mismatches, also $\|x - y\|_1$

0**0**1**010**01
0**1**1**101**01

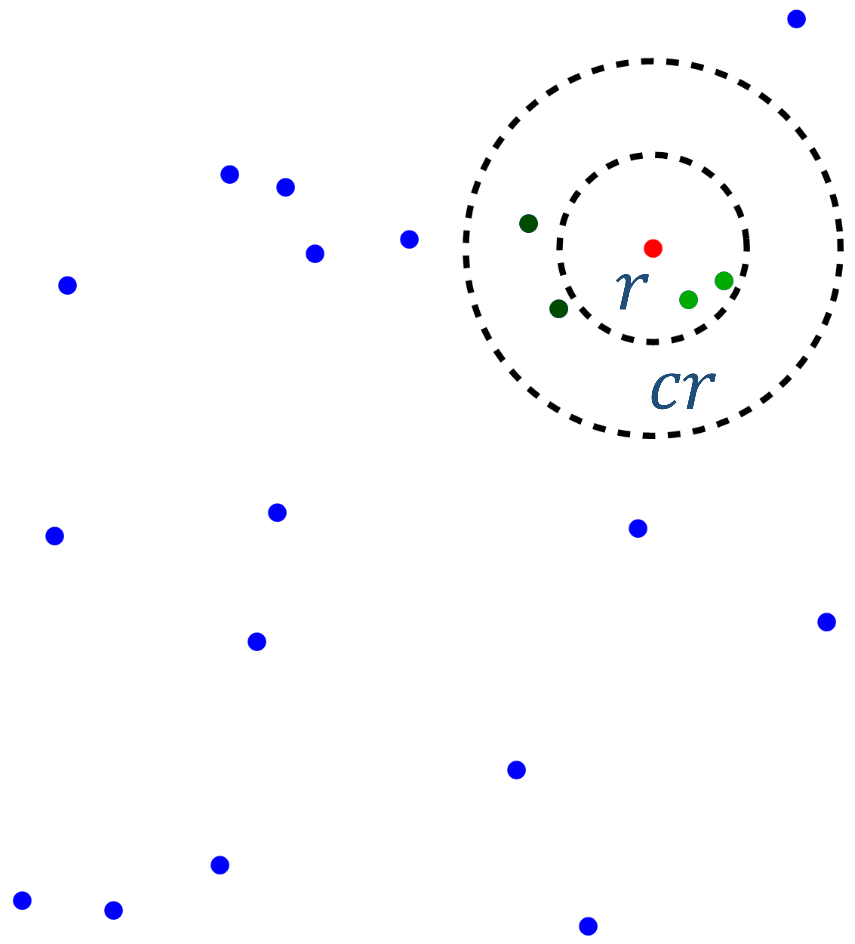
Example

- Dataset: 10M uniformly random points from $\{0,1\}^{1024}$
- One planted pair at distance 150
- Can we find it quickly?
- Naïve way: enumerate 10^{14} pairs
- Can we avoid it?

Fixed scale

Dataset: n points in $\{0,1\}^d$
Query: a point in $\{0,1\}^d$
within r from a data point
Goal: find a data point
within cr from the query

Additional data
- Approximation $c > 1$
- Distance scale $r > 0$



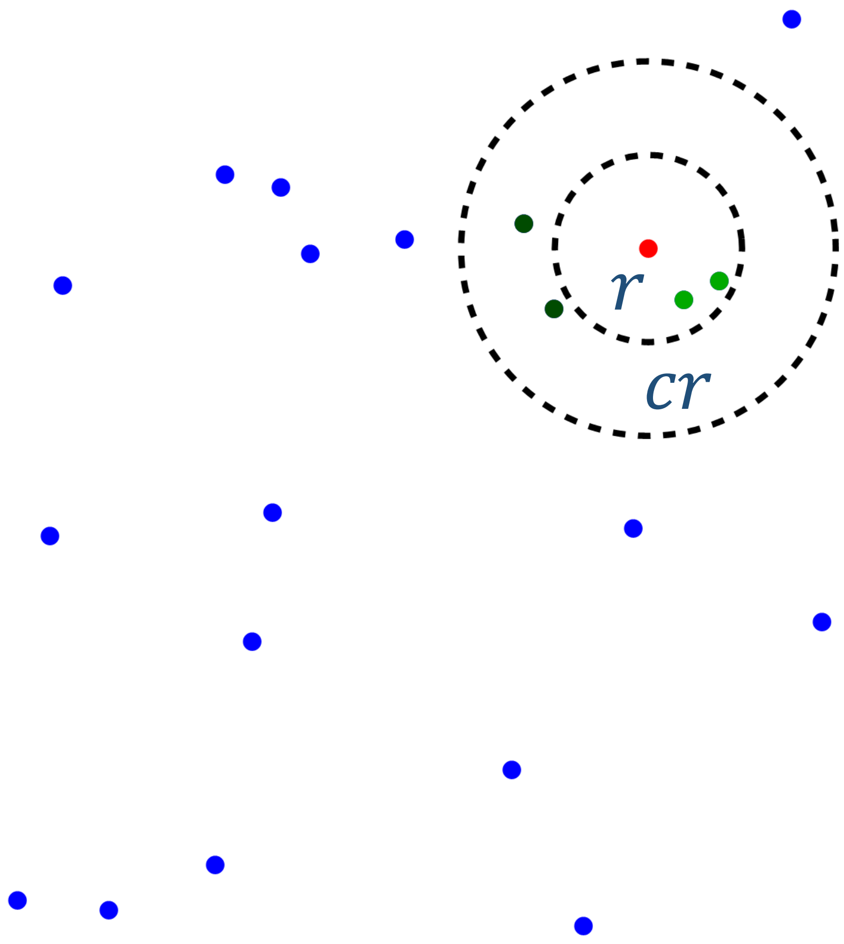
From fixed scale to the original problem

- Build a data structure for each r
- During the query stage, run binary search on the answer
- Overhead $O(d)$ in space, $O(\log d)$ in query time
- Fine print: assume that the error probability is $1 - \frac{1}{10\, d}$

Fixed scale

Dataset: n points in $\{0,1\}^d$
Query: a point in $\{0,1\}^d$
within r from a data point
Goal: find a data point
within cr from the query

Additional data
- Approximation $c > 1$
- Distance scale $r > 0$



Coordinate sampling

- **Idea:** sample K random coordinates
- Given a query, find all the data points that **match the query exactly** on the selected coordinates (can use a hash table)
- If there is any point within cr from the query, we are done

Analysis

- Number of far points (further than cr) that match the query
  - $n \cdot \left(1 - \frac{cr}{d}\right)^K$
- Set K such that this number is around 1
- It means that the query time is $O(d)$
- The probability of success is at least:
  - $\left(1 - \frac{r}{d}\right)^K \gtrsim n^{-1/c}$
- Repeat $O\left(n^{1/c}\right)$ times to get success probability 0.99

Overall algorithm for a single scale

- Sample $L = O\left(n^{1/c}\right)$ random subsets $S_1, S_2, \ldots, S_L$ of coordinates
- Each subset is of the size $\log_{\left(1-\frac{cr}{d}\right)^{-1}} n$
- Given a query, retrieve all the data points that match it exactly, when restricted on some S_i
- Stop as soon as we find something within distance cr from the query

Example

- Dataset: 10M uniformly random points from $\{0, 1\}^{1024}$
- One planted pair at distance 150
- Sample 23 coordinates, get $2^{23} \approx 10\text{M}$ buckets
- Check all pairs in each bucket
- A typical run is ≈ 40 **iterations and** $\approx 300\text{M}$ **comparisons**
- C++ code is short (150 lines with all the bells and whistles)

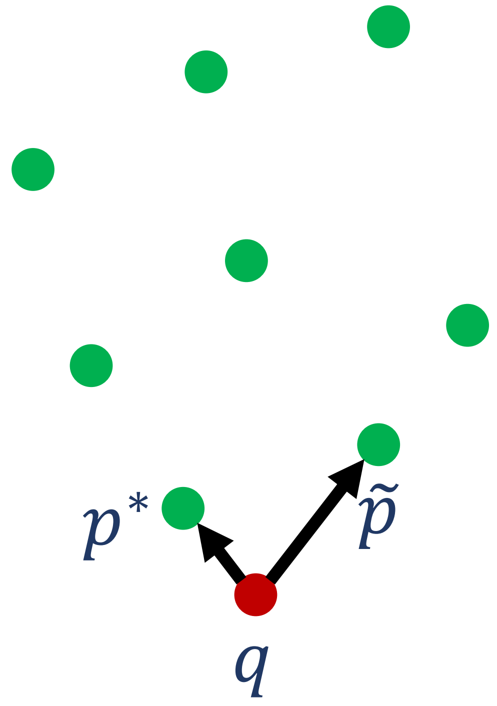
Euclidean distance

Approximate Nearest Neighbors

- **Dataset:** n points in R^d (denote by P)
- Approximation $c > 1$
- **Query:** $q \in R^d$
- **Want:** $\tilde{p} \in P$ such that

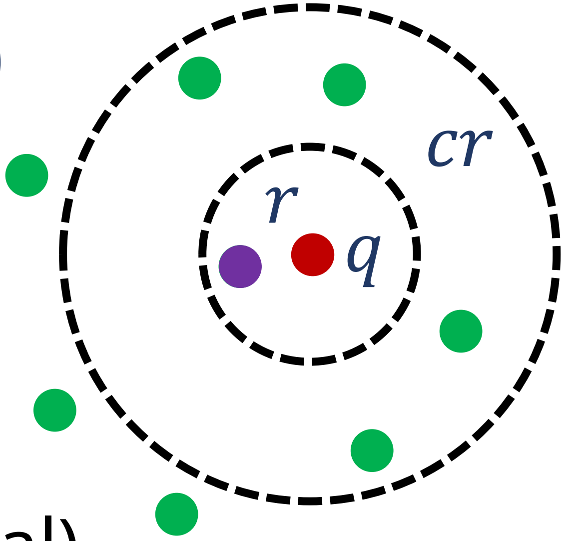
$$\|q - \tilde{p}\| \leq c \cdot \min_{p^* \in P} \|q - p^*\|$$

- Parameters: **space, query time**
- The main regime: $d = \widetilde{\Theta}(\log n)$ (assume from now on)
  - **[Johnson, Lindenstrauss 1984]** (random projections)



Approximate Near Neighbors (ANN)

- **Dataset:** n points in R^d (denote by P)
- Approximation $c > 1$, distance threshold $r > 0$
- **Query:** $q \in R^d$ such that there is $p^* \in P$ with $\|q - p^*\| \leq r$
- **Want:** $\tilde{p} \in P$ such that
$$\|q - \tilde{p}\| \leq cr$$
- **[Har-Peled, Indyk, Motwani 2012]:** (non-trivial) reduction to ANN with $(\log n)^{O(1)}$ overhead



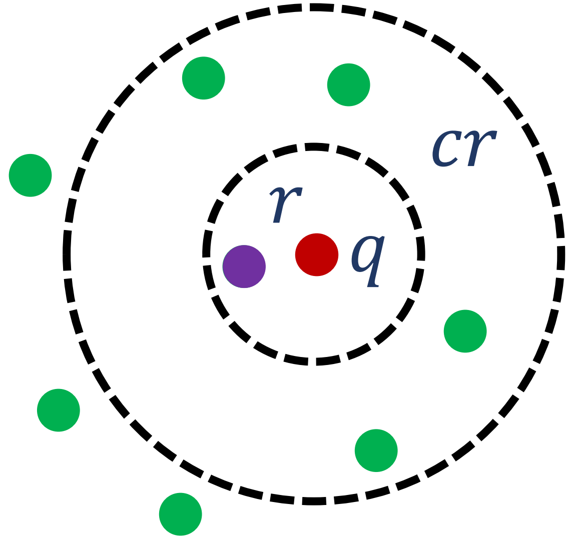
Spherical case

- Can further reduce ANN to the spherical case:

  points and queries lie on a unit sphere $S^{d-1} \subset R^d$
- *Informally:* look at the dataset from "far away"
- In practice: **cosine similarity,** interesting by itself
  - Simhash **[Charikar 2002]**

The core problem: ANN on a sphere

- **Dataset:** n points in $S^{d-1} \subset R^d$ (denote by P)
- Approximation $c > 1$, distance threshold $r > 0$
- **Query:** $q \in S^{d-1}$ such that there is $p^* \in P$ with $\|q - p^*\| \leq r$
- **Want:** $\tilde{p} \in P$ such that $\|q - \tilde{p}\| \leq cr$



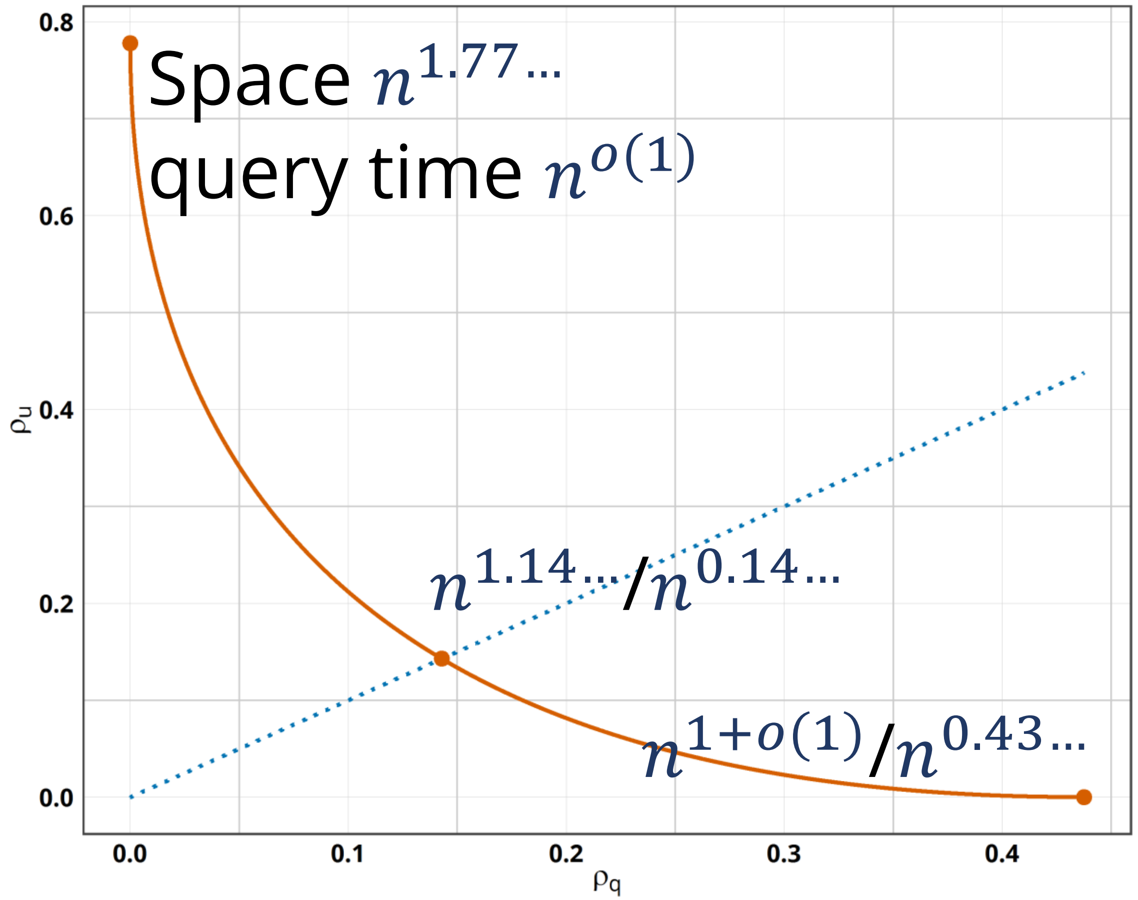
Main question

Given a space budget and desired approximation,
what is the query time one can achieve?

Our results

- Simple, modular data structure
  - Space $n^{1+\rho_u+o(1)}$, query time $n^{\rho_q+o(1)}$
- **Optimal** in a restricted model

$c = 2$

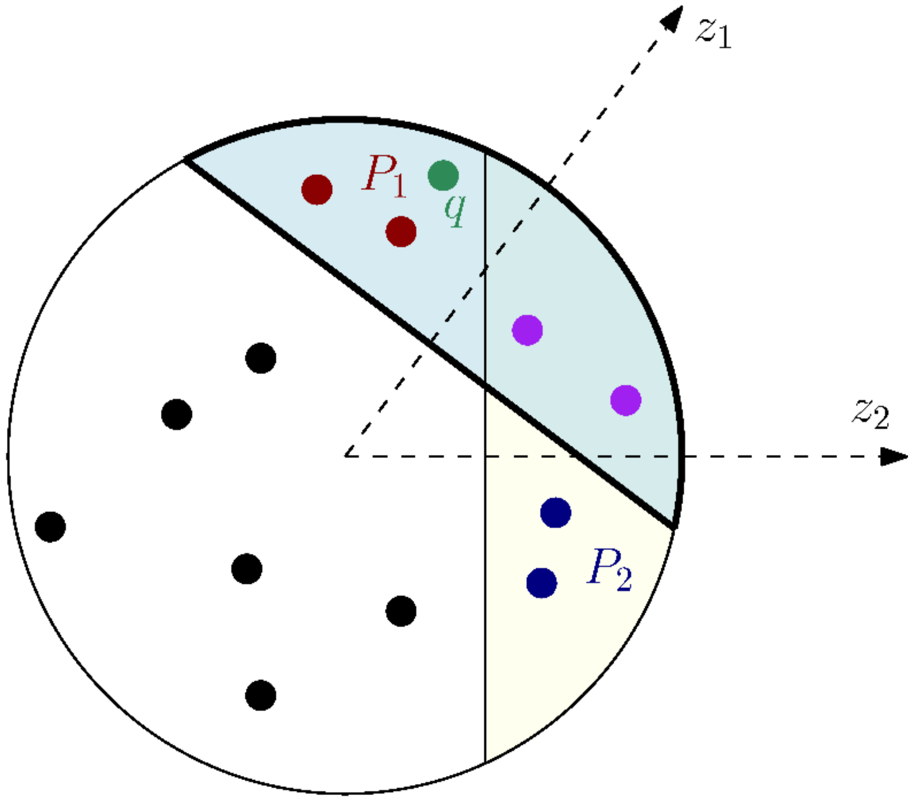


Plan

- Simple algorithm for the LSH regime (space $n^{1+\rho}$, time n^{ρ}) *assuming a magic oracle*
- Full time–space trade-off
- Getting rid of the oracle
- Data-dependent partitioning: an improved trade-off

Basic algorithm with a magic oracle

- T and η – parameters to be chosen later
- **Preprocessing**
  - Sample T Gaussian vectors $z_1, z_2, \dots, z_T \sim N(0,1)^{\otimes d}$
  - Form subsets $P_i = \{p \in P \mid \langle p, z_i \rangle \geq \eta\}$
  - Store z_i and P_i for *non-empty* P_i's
- **Query**
  - Retrieve all the caps such that $\langle q, z_i \rangle \geq \eta$
  - Search the retrieved P_i's for a point within cr from q

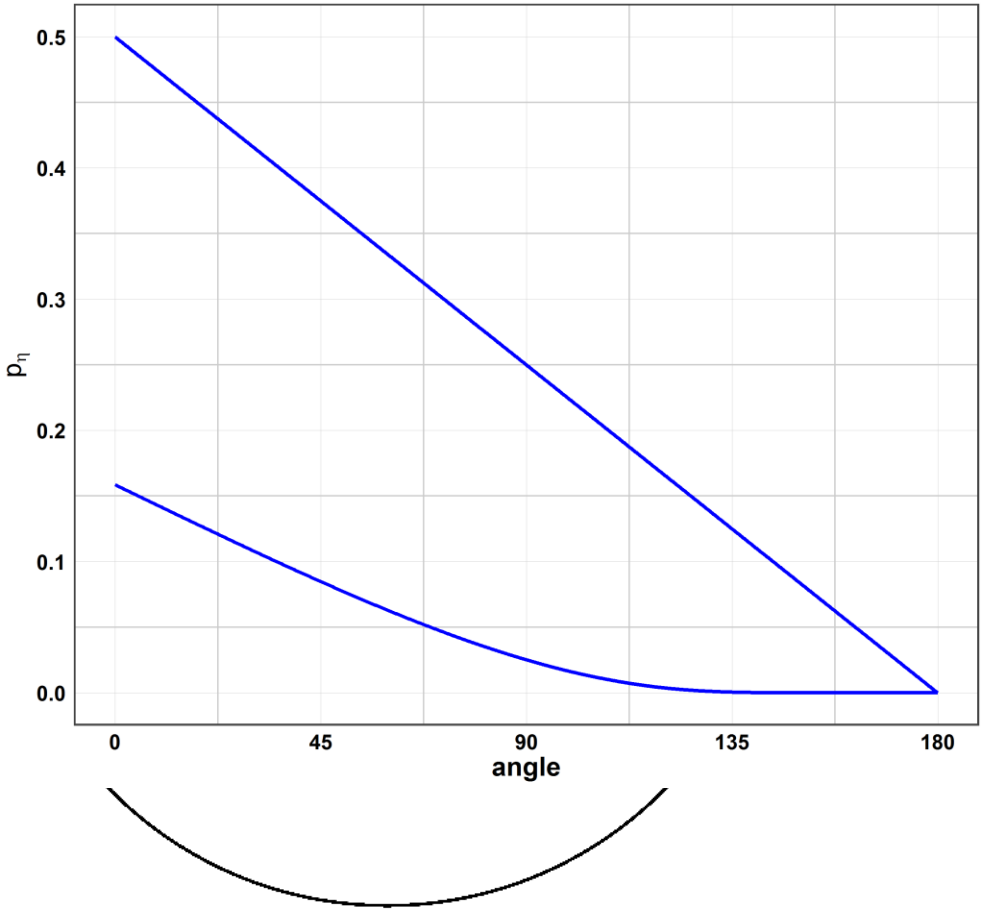


The key quantity

- Denote for two points $x, y \in S^{d-1}$ with $\|x - y\| = s$

$$p_\eta(s) = \Pr_{z \sim N(0,1)^{\otimes d}}[\langle z, x \rangle \geq \eta, \langle z, y \rangle \geq \eta]$$

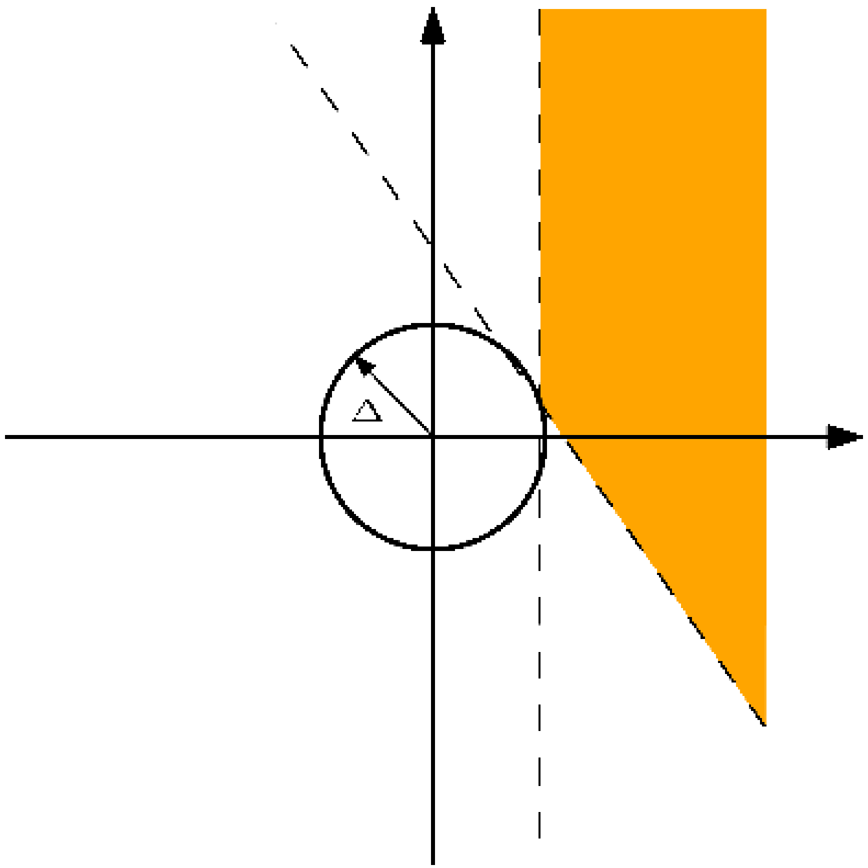
- $p_0(s) = 1 - \frac{\varphi(s)}{\pi}$, where $\varphi(s)$ is the angle for distance s (random hyperplane)
- Next: *simple* and *good* estimates on $p_\eta(s)$



Estimates on $p_\eta(s)$

$p_\eta(s)$

Trick: integrate in polar coordinates



Analysis

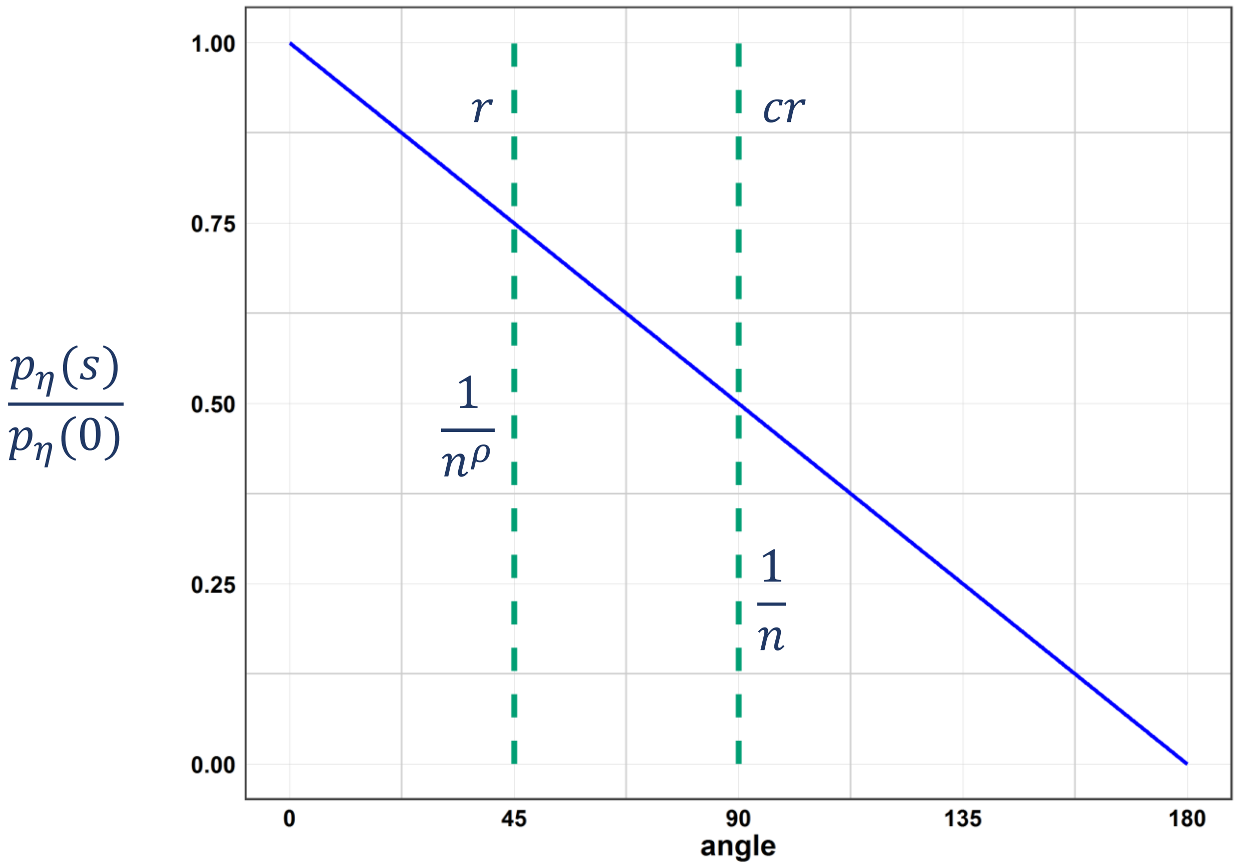
Need to set:

- Number of caps T

Que

Summary:

Recipe for choosing η



- Use estimates on $p_\eta(s)$
- $\rho(r,c) \leq \frac{1}{c^2} + o(1)$
- Space $n^{1+\frac{1}{c^2}+o(1)}$, query time $n^{\frac{1}{c^2}+o(1)}$
- Worst case: $r \to 0$

Plan

- Simple algorithm for the LSH regime (space $n^{1+\rho}$, time n^{ρ}) *assuming an unrealistic oracle*
- Full time–space trade-off
- Getting rid of the oracle
- Data-dependent partitioning: an improved trade-off

Main question

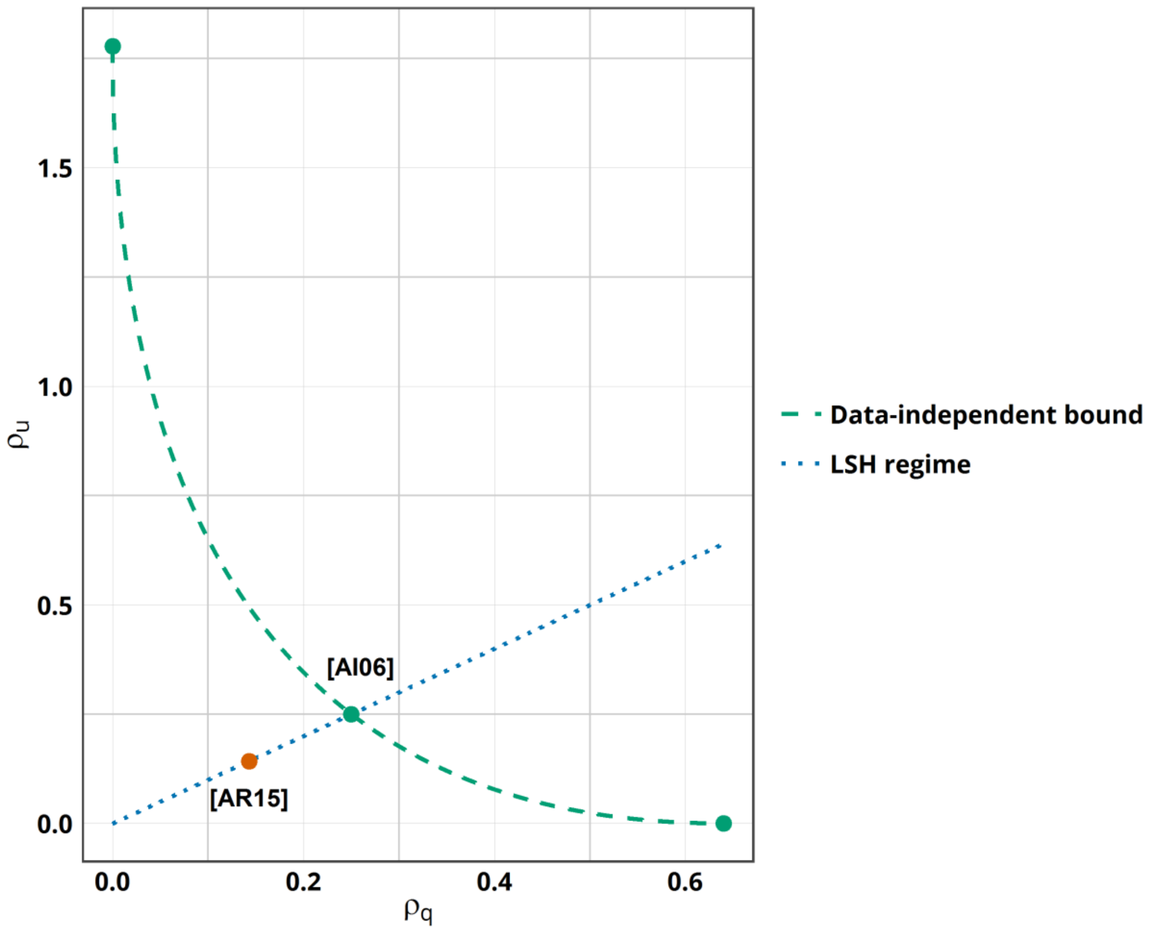
Given a space budget and desired approximation,
what is the query time one can achieve?

The full trade-off with an oracle

- T, η_u and η_q – parameters to be chosen later
- **Preprocessing**
  - Sample T Gaussian vectors $z_1, z_2, \ldots, z_T \sim N(0,1)^{\otimes d}$
  - Form subsets $P_i = \{p \in P \mid \langle p, z_i \rangle \geq \eta_u\}$
  - Store z_i and P_i for *non-empty* P_i's
- **Query**
  - Retrieve all the caps such that $\langle q, z_i \rangle \geq \eta_q$
  - Search the retrieved P_i's for a point within cr from q
- **Regimes:** $\eta_u < \eta_q$ for *faster queries*, $\eta_u > \eta_q$ for *less memory*

What we get

- Space $n^{1+\rho_u+o(1)}$, time $n^{\rho_q+o(1)}$ (plot for $c = 2$)

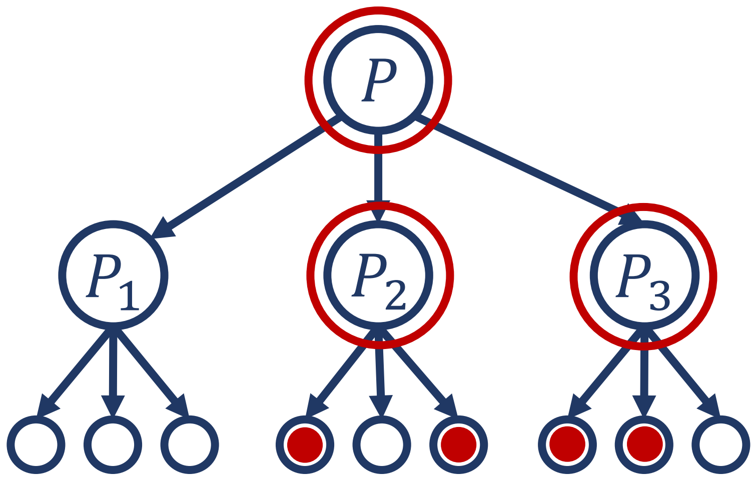


Plan

- Simple algorithm for the LSH regime (space $n^{1+\rho}$, time n^{ρ}) *assuming an unrealistic oracle*
- Full time–space trade-off
- Getting rid of the oracle
- Data-dependent partitioning: an improved trade-off

Getting rid of the oracle

- **Idea:** "gradual" partitioning, new parameter K
- **Preprocessing**
  - Sample T Gaussian vectors $z_1, z_2, \dots, z_T \sim N(0,1)^{\otimes d}$
  - Form subsets $P_i = \{p \in P \mid \langle p, z_i \rangle \geq \eta_u\}$
  - *Recurse* on non-empty P_i's
  - At level K, store P_i's explicitly
- **Query**
  - Recursively query all the caps for which $\langle q, z_i \rangle \geq \eta_q$ (search using linear scan!)
  - At level K, search the P_i's for a point within cr from q



$T = 3, K = 2$

How to set parameters

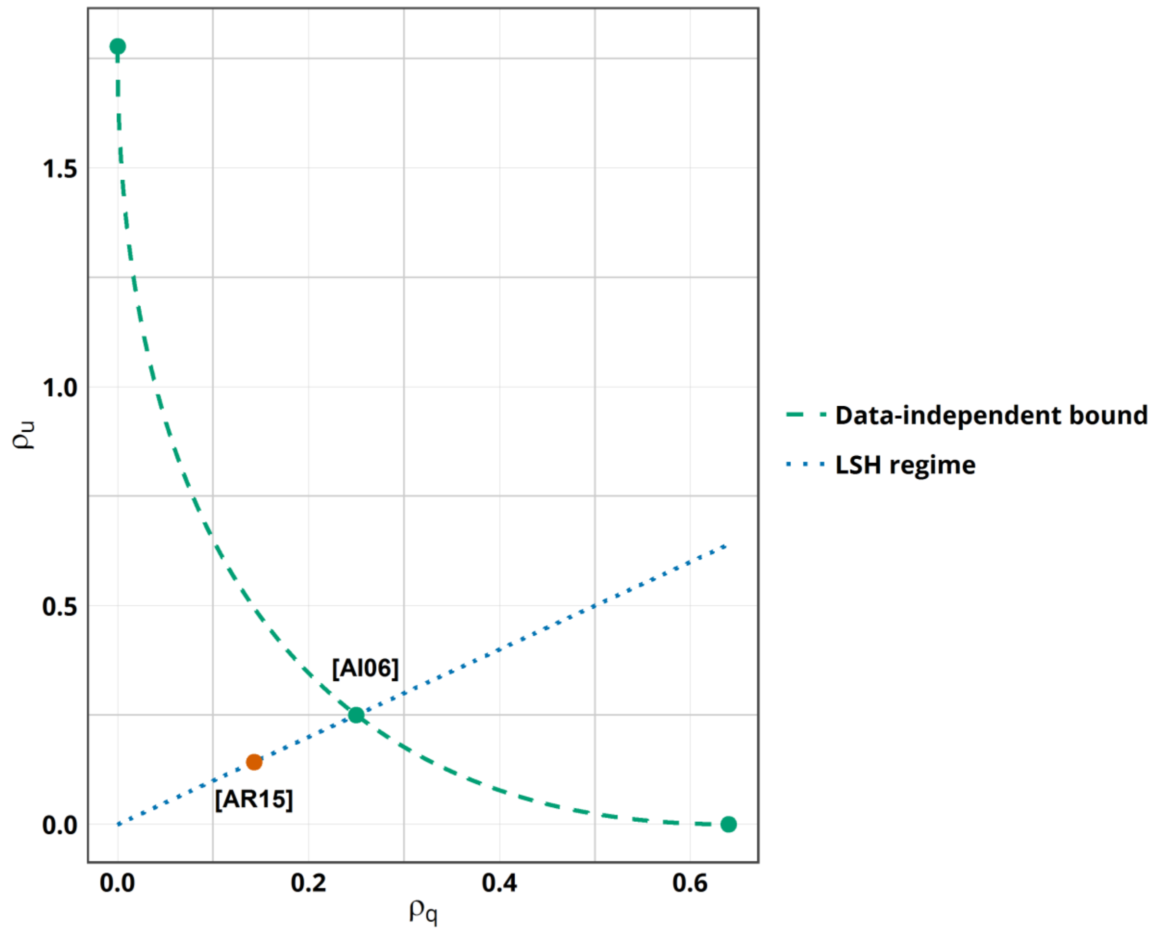
- Small K – slow point location
- Large K – bad value of $\rho(c, r)$
- A possible choice – $K \sim \sqrt{\ln n}$

Plan

- Simple algorithm for the LSH regime (space $n^{1+\rho}$, time n^{ρ}) *assuming an unrealistic oracle*
- Full time–space trade-off
- Getting rid of the oracle
- Data-dependent partitioning: an improved trade-off

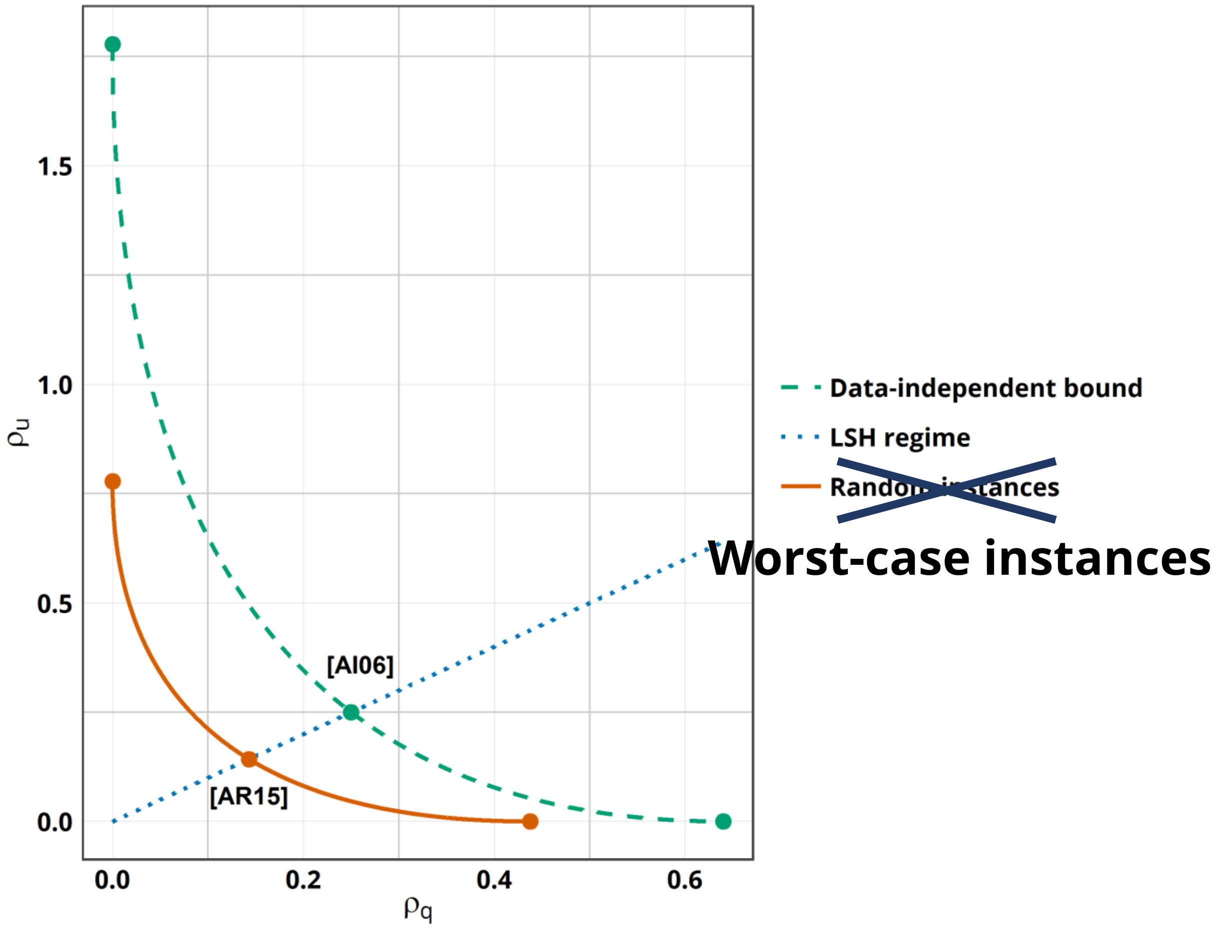
Data-dependent partitions

- So far, data-dependent LSH from **[Andoni, R 2015]** is better for the case $\rho_u = \rho_q$
- Can we get the best of both worlds?



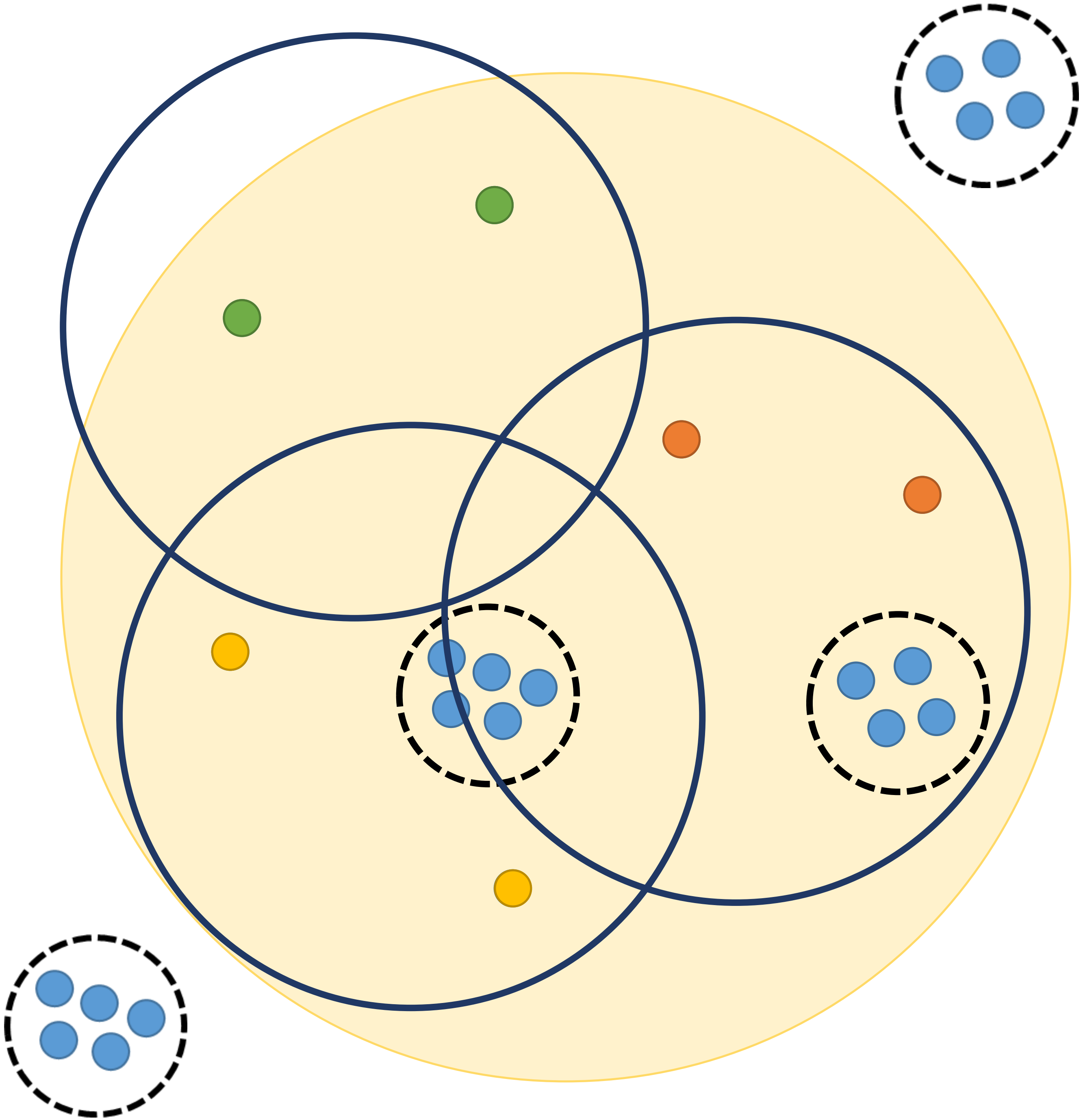
Random instances

- **Dataset:** n uniformly random unit vectors (pairwise distances concentrated around $\sqrt{2}$)
- **Queries:** planted at random within distance $r = \frac{\sqrt{2}}{c}$
- Reduction from **worst** case to **random**, can we do the same here?



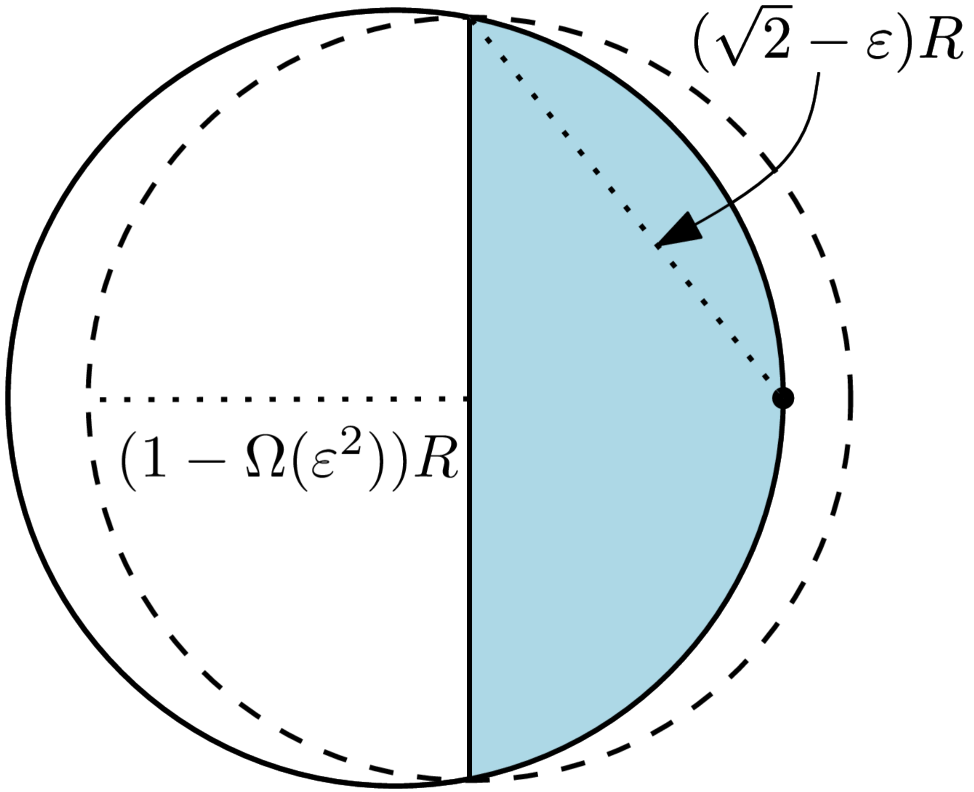
The general case

- The dataset does not look random
- Remove *structure*—clusters of small radius with $n^{1-\delta}$ points—until there are none
  - Will handle them separately
- The remainder **looks like a random set**
  - No dense areas, hence points are spread
- Sample T caps, recurse
  - Clusters can appear again
- Query **all** the clusters and **necessary** caps



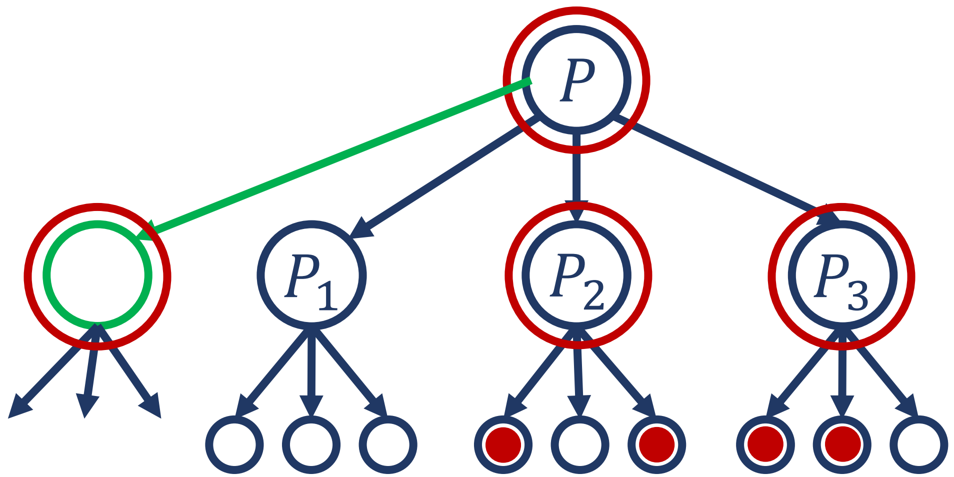
Handling clusters

- Enclose a cluster of radius $\sqrt{2} - \varepsilon$ in a ball of radius $\left(1 - \Omega\left(\varepsilon^2\right)\right)$
- Recurse with reduced radius



Overall bookkeeping

- **For clusters:** radius reduction makes the problem more isotropic
- **For the remainder:** data-independent partitioning works great (for one step)
- In terms of tree: besides cap nodes, we have **cluster nodes**, each query recurses on **all of them**



Any questions about the algorithm?