# CS-GY 6763: Lecture 6
# Near-neighbor search in high dimensions

NYU Tandon School of Engineering, Prof. Christopher Musco

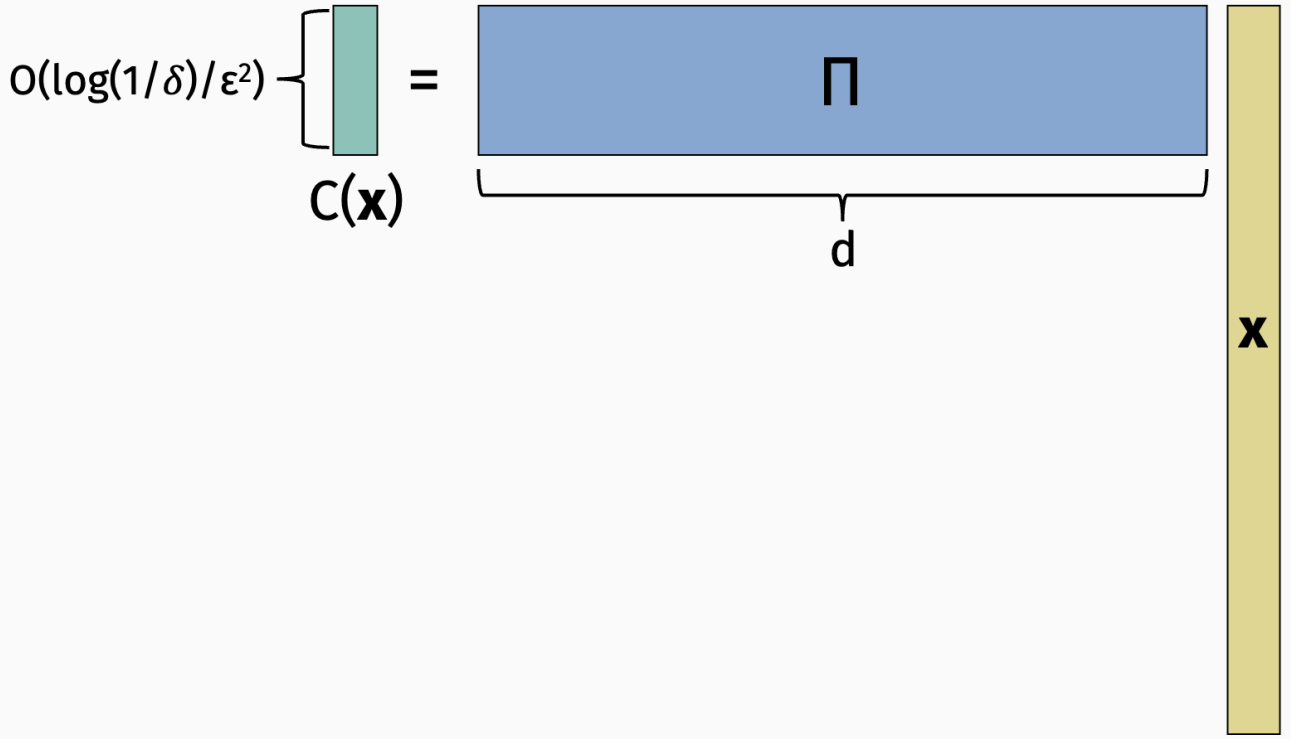
# LAST CLASS

**Dimensionality reduction:** Given vectors $\mathbf{x}, \mathbf{y}$, compute small space compressions $C(\mathbf{x})$ and $C(\mathbf{y})$ that can be used to estimate the distance or similarity between $\mathbf{x}$ and $\mathbf{y}$.

## Lemma (Distributional JL Lemma)

*Let* $\mathbf{\Pi}$ *be a random matrix that compresses to* $k = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ *rows. Then with probability* $(1-\delta)$*:*

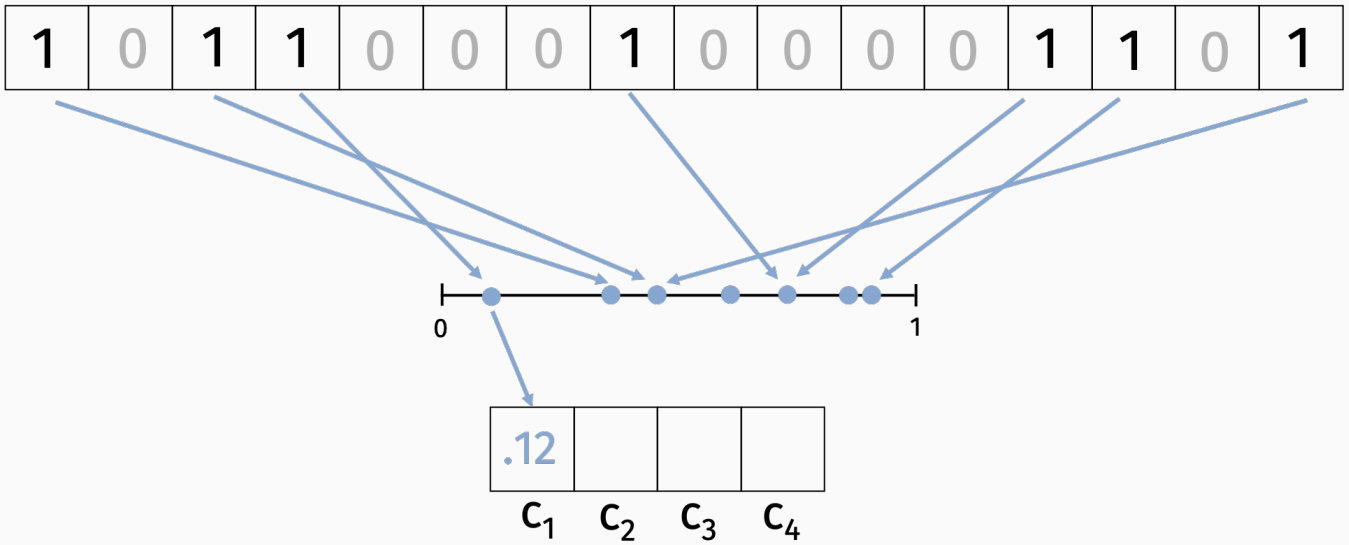
$$(1-\epsilon)\|\mathbf{x}-\mathbf{y}\|_2^2 \leq \|\mathbf{\Pi}\mathbf{x}-\mathbf{\Pi}\mathbf{y}\|_2^2 \leq (1+\epsilon)\|\mathbf{x}-\mathbf{y}\|_2^2$$



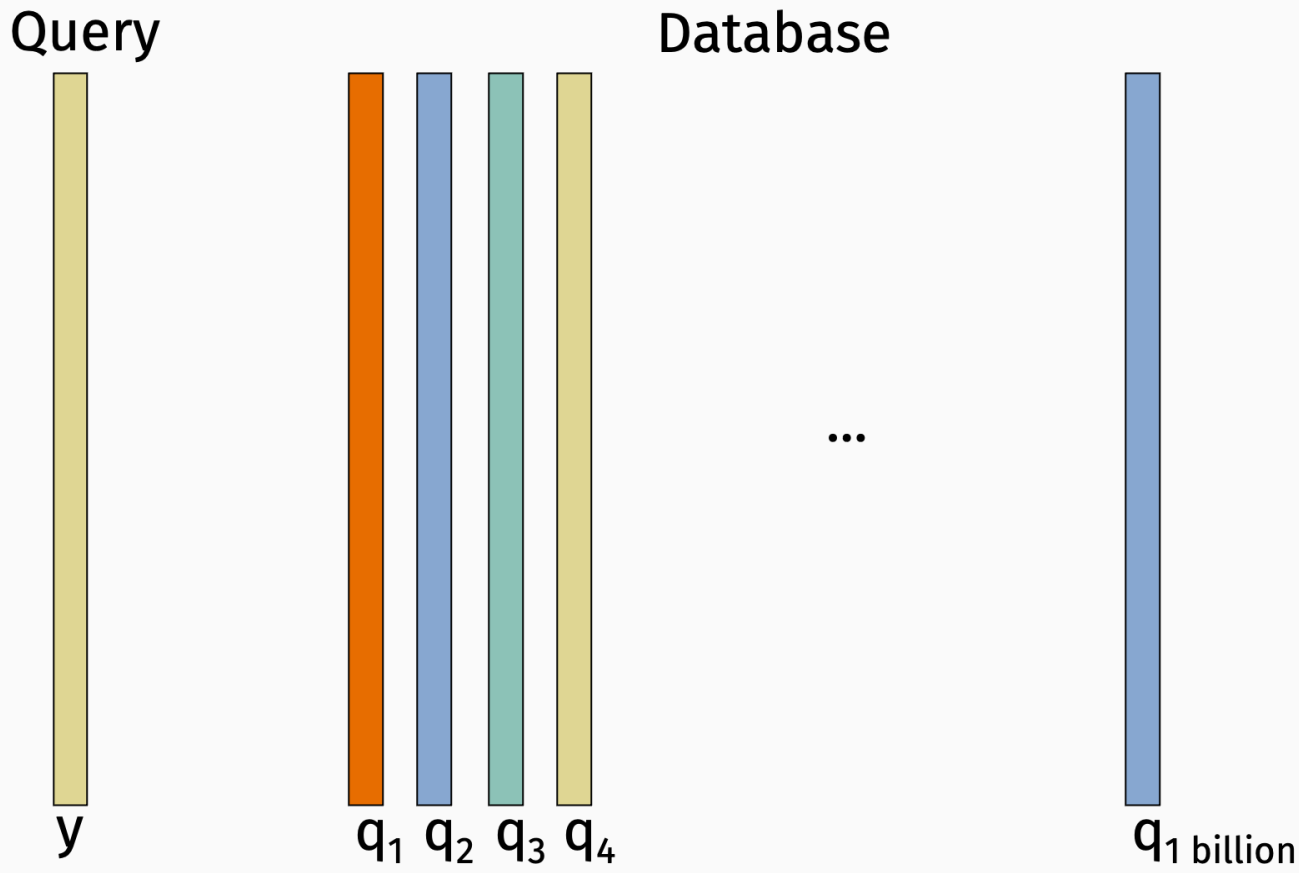
## Lemma (MinHash)

*Let $C$ be a length $k = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ MinHash sketch. Then with probability $(1-\delta)$, we can return an estimate $\tilde{J}$ based on $C(\mathbf{x})$ and $C(\mathbf{y})$ with:*

$$J(\mathbf{x}, \mathbf{y}) - \epsilon \leq \tilde{J} \leq J(\mathbf{x}, \mathbf{y}) + \epsilon.$$



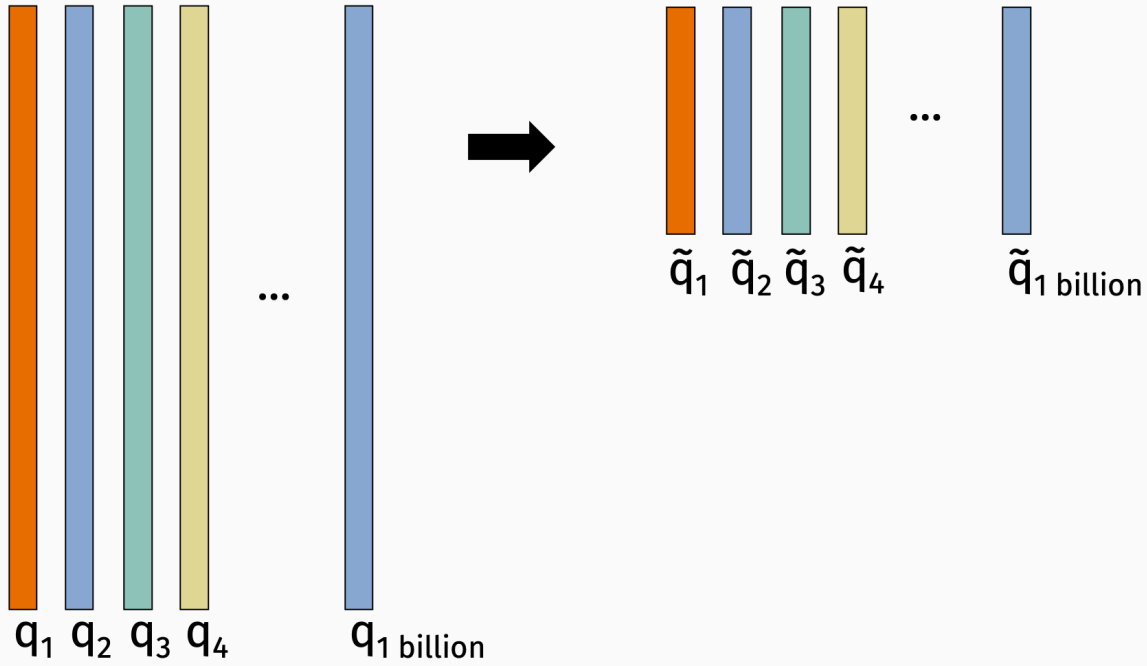
# KEY APPLICATION: MODERN VECTOR SEA SEARCH




Cost of naive algorithm is $O(nd)$.

# KEY APPLICATION: MODERN VECTOR SEA SEARCH

Dimensionality reduction reduces search cost to $O(nk)$ and reduces space requirements.




All modern vector search systems use "fancier" versions of methods studied in this class:

- Quantized JL/SimHash
- b-bit MinHash
- Product Quantization
- PCA-based methods

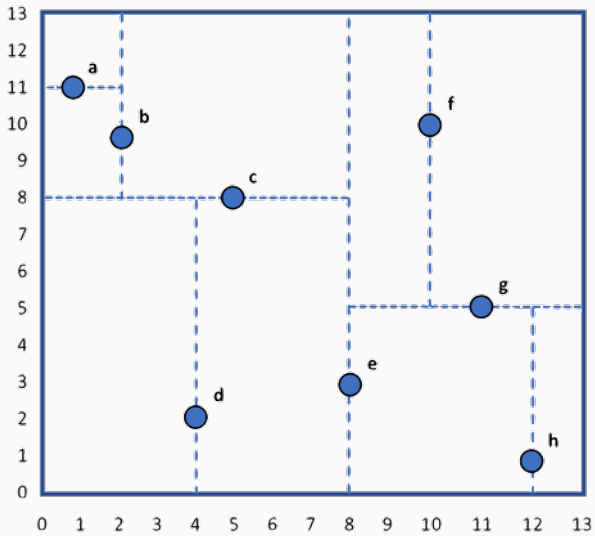
Dimensionality reduction methods are typically paired with vector indexing methods.

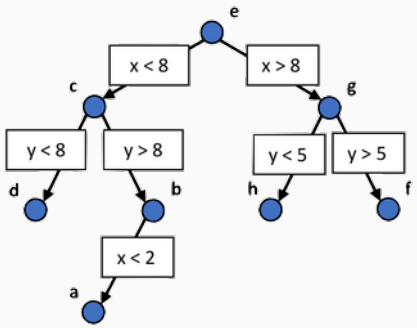
**Goal of Dimensionality Reduction:** Reduce dependence on $d$ in $O(nd)$ search cost. Reduce space complexity.

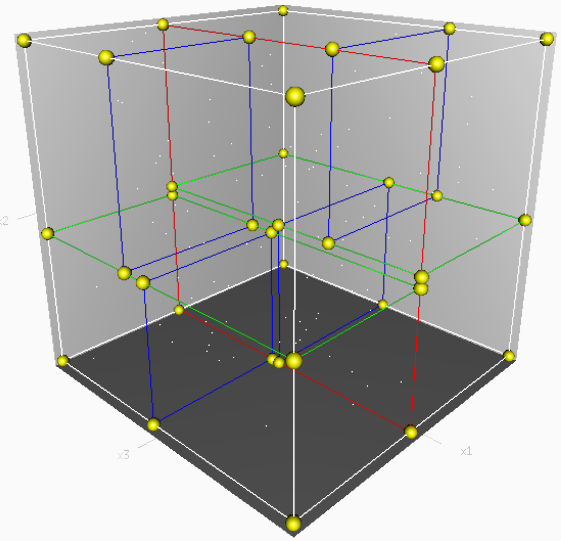
**Goal of Vector Indexing:** Reduce dependence on $n$ in $O(nd)$ search cost. Often at the cost of added space complexity.

# BEYOND A LINEAR SCAN

This problem can already be solved in low-dimensions using space partitioning approaches (namely, kd-trees).

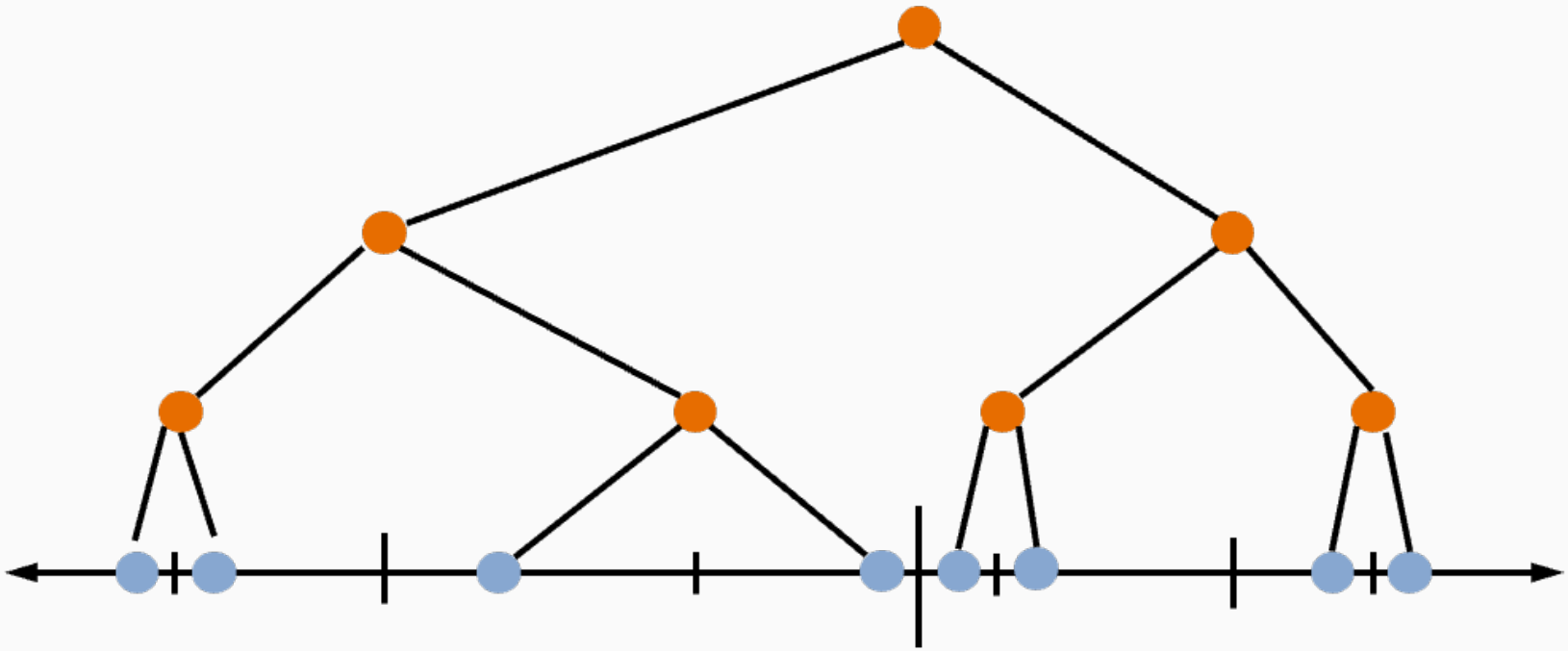


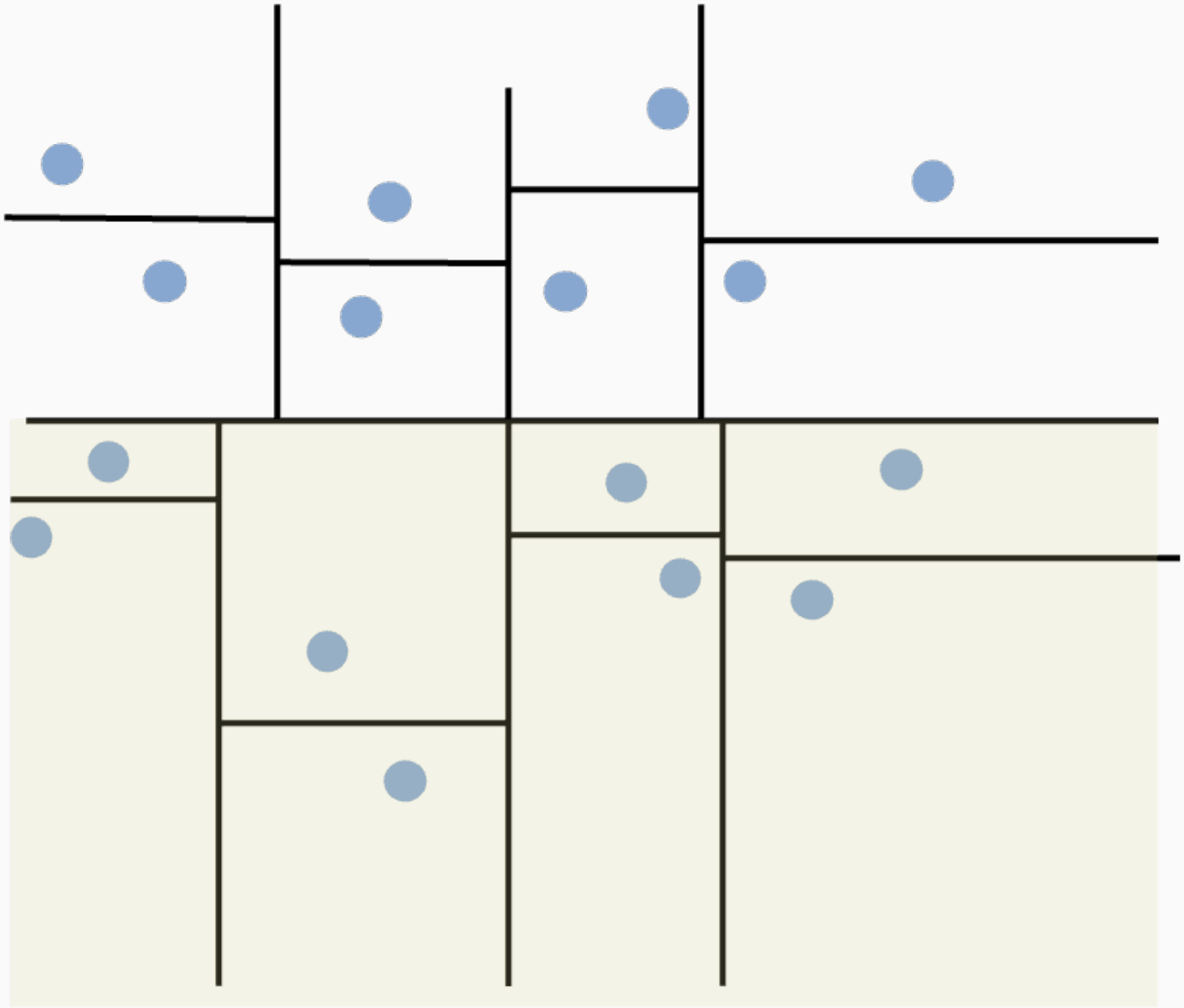



Search time is roughly $O(d \cdot \log n \cdot 2^d))$, which is only sublinear for $d = o(\log n)$.

# ISSUE WITH KD-TREES



# ISSUE WITH KD-TREES



# HIGH DIMENSIONAL NEAR NEIGHBOR SEARCH

Only been attacked much more recently:

- Locality-sensitive hashing [Indyk, Motwani, 1998]
- Spectral hashing [Weiss, Torralba, and Fergus, 2008]
- Vector quantization [Jégou, Douze, Schmid, 2009]
- Graph-based vector search [Malkov, Yashunin, 2016, Subramanya et al., 2019]

**Key ideas behind all of these methods:**

1. Trade worse space-complexity + preprocessing time for better time-complexity. I.e., preprocess database in data structure that uses $\Omega(n)$ space.
2. Allow for approximation.

# Intuitively why do preprocessing and space help?

**Question:** Suppose you want to search over points in $[-1, 1]^d$ and to achieve accuracy $\epsilon$. I.e., for a given $\mathbf{y} \in [-1, 1]^d$, you want to find $\tilde{\mathbf{q}}$ with $\|\mathbf{y} - \tilde{\mathbf{q}}\|_2 \leq \min_i \|\mathbf{y} - \mathbf{q}_i\|_2 + \epsilon$.

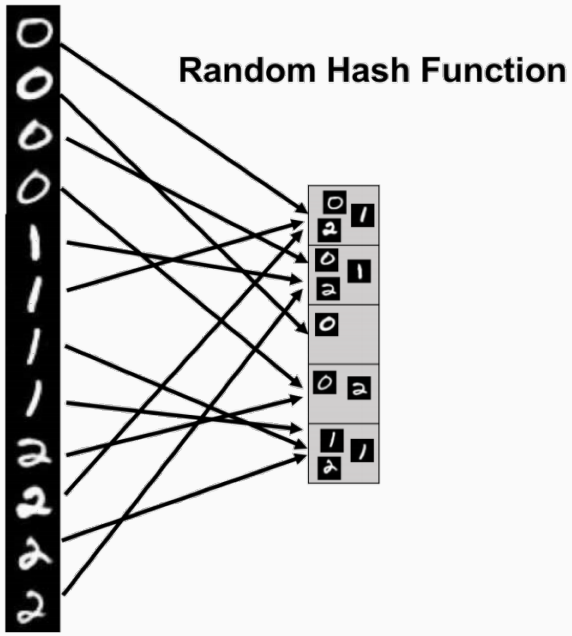
Can you construct a data structure that supports $O(1)$ time search but uses exponential space?

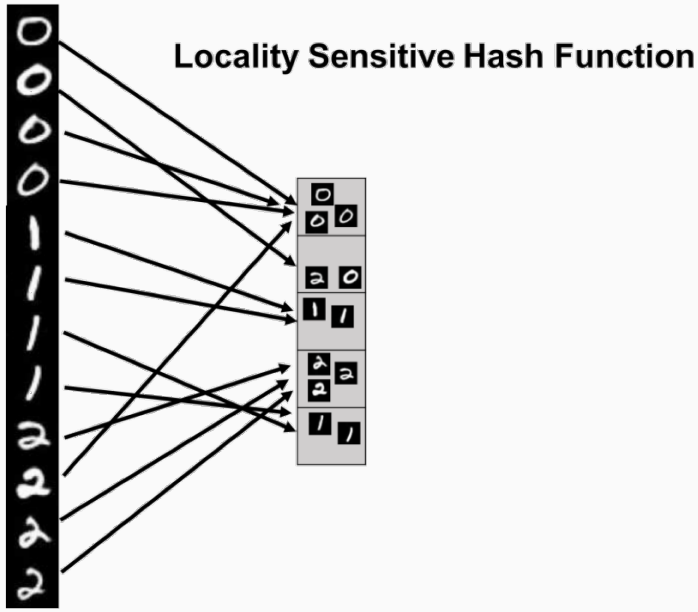
# LOCALITY SENSITIVE HASH FUNCTIONS

Let $h : \mathbb{R}^d \to \{1, \ldots, m\}$ be a random hash function.

We call $h$ locality sensitive for similarity function $s(\mathbf{q}, \mathbf{y})$ if $\Pr[h(\mathbf{q}) == h(\mathbf{y})]$ is:

- Higher when $\mathbf{q}$ and $\mathbf{y}$ are more similar, i.e. $s(\mathbf{q}, \mathbf{y})$ is higher.
- Lower when $\mathbf{q}$ and $\mathbf{y}$ are more dissimilar, i.e. $s(\mathbf{q}, \mathbf{y})$ is lower.

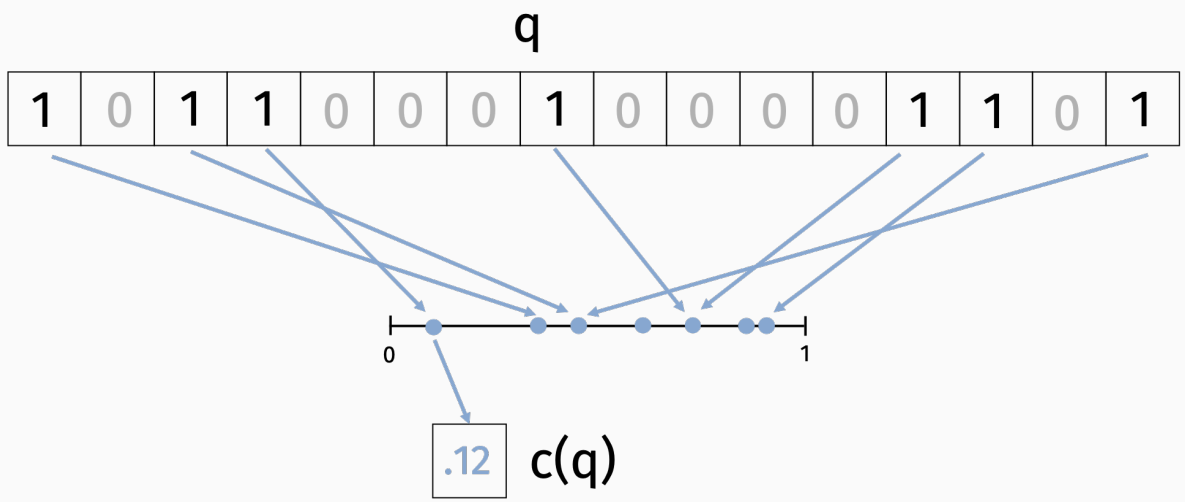


# LOCALITY SENSITIVE HASH FUNCTIONS

LSH for $s(\mathbf{q}, \mathbf{y})$ equal to Jaccard similarity:

- Let $c : \{0,1\}^d \to [0,1]$ be a single instantiation of MinHash.
- Let $g : [0,1] \to \{1, \ldots, m\}$ be a uniform random hash function.
- Let $h(\mathbf{q}) = g(c(\mathbf{q}))$.



# LOCALITY SENSITIVE HASH FUNCTIONS

LSH for Jaccard similarity:

- Let $c : \{0,1\}^d \to [0,1]$ be a single instantiation of MinHash.
- Let $g : [0,1] \to \{1, \ldots, m\}$ be a uniform random hash function.
- Let $h(\mathbf{x}) = g(c(\mathbf{x}))$.

If $J(\mathbf{q}, \mathbf{y}) = v$,

$$\Pr[h(\mathbf{q}) == h(\mathbf{y})] =$$

Basic approach for LSH-based near neighbor search in a database.

**Pre-processing:**

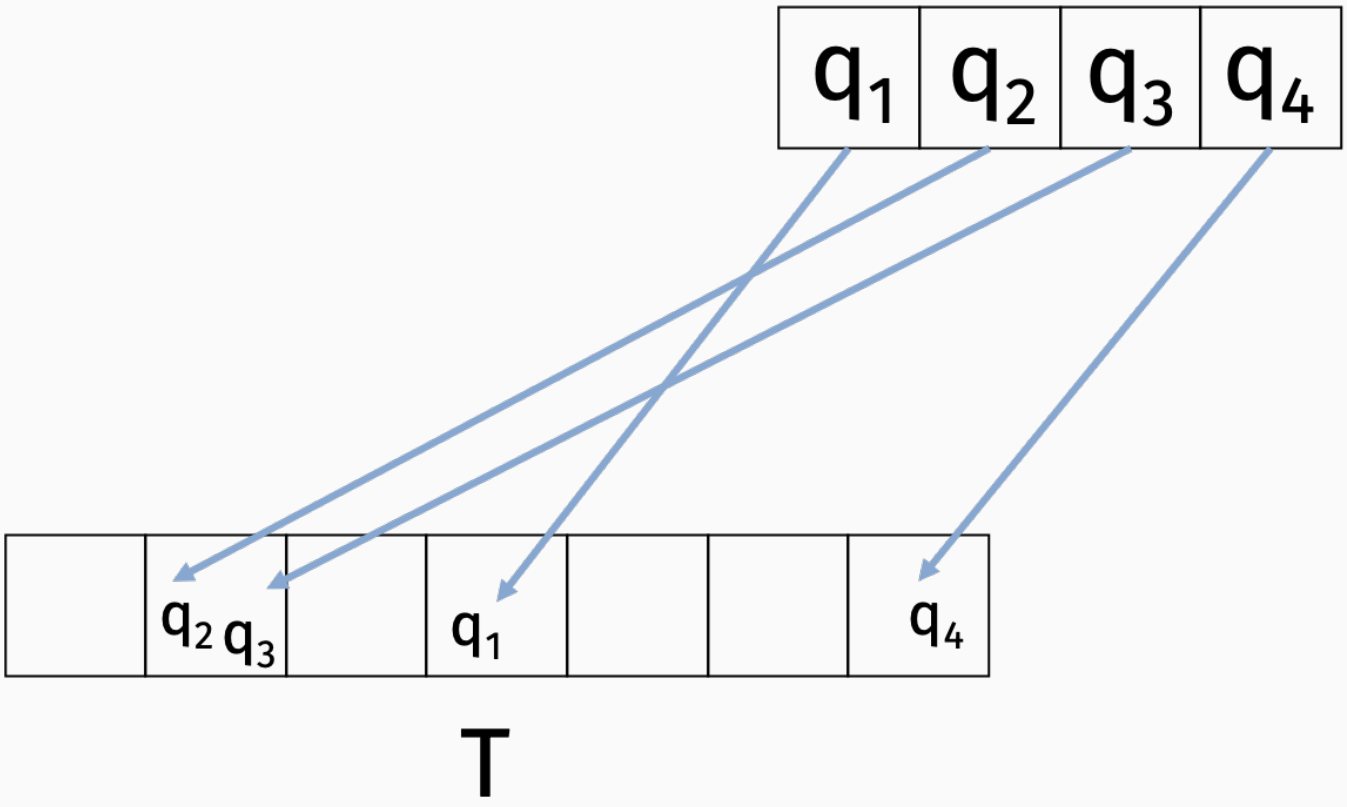
- Select random LSH function $h : \{0,1\}^d \to 1, \ldots, m$.
- Create table $T$ with $m = O(n)$ slots.[1]
- For $i = 1, \ldots, n$, insert $\mathbf{q}_i$ into $T(h(\mathbf{q}_i))$.

**Query:**

- Want to find near neighbors of input $\mathbf{y} \in \{0,1\}^d$.
- Linear scan through all vectors $\mathbf{q} \in T(h(\mathbf{y}))$ and return any that are close to $\mathbf{y}$. Time required is $O(d \cdot |T(h(\mathbf{y})|)$.

[1] Enough to make the $O(1/m)$ term negligible.

# NEAR NEIGHBOR SEARCH



Two main considerations:

- **False Negative Rate**: What's the probability we do not find a vector that is close to $\mathbf{y}$?
- **False Positive Rate**: What's the probability that a vector in $T(h(\mathbf{y}))$ is not close to $\mathbf{y}$?

A higher false negative rate means we miss near neighbors.

A higher false positive rate means increased runtime – we need to compute $S(\mathbf{q}, \mathbf{y})$ for every $\mathbf{q} \in T(h(\mathbf{y}))$ to check if it's actually close to $\mathbf{y}$.

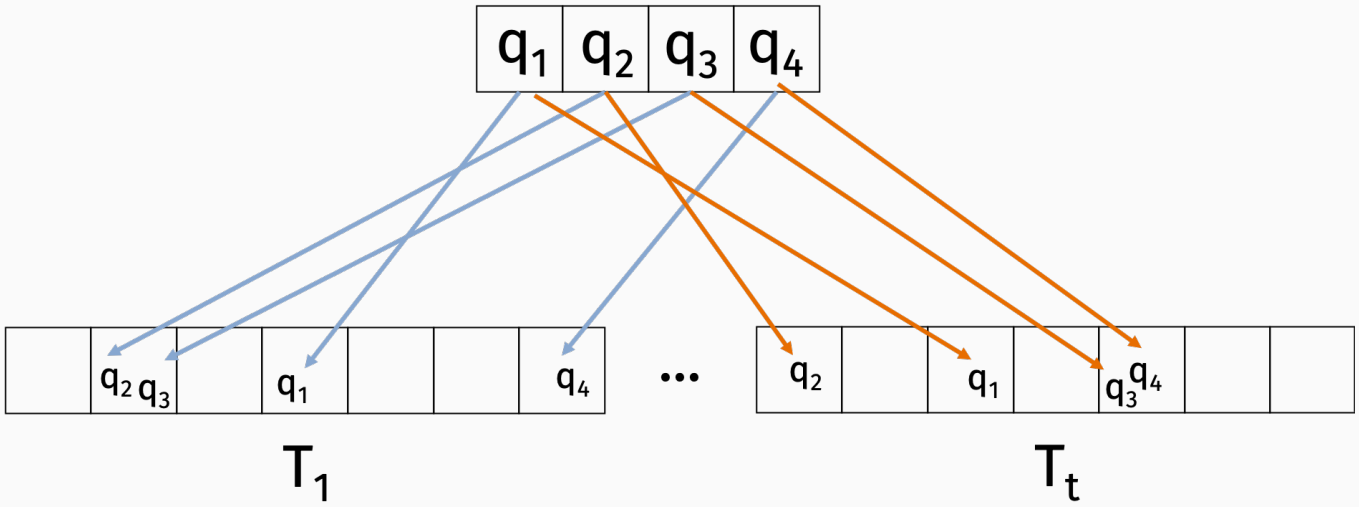
**Note:** The meaning of "close" and "not close" is application dependent. E.g. we might specify that we want to find anything with Jaccard similarity $> .4$, but not with Jaccard similarity $< .2$.

# REDUCING FALSE NEGATIVE RATE

Let's use Jaccard similarity as a running example. We will discuss LSH for inner product/Euclidean distance as well. Suppose the nearest database point $\mathbf{q}$ has $J(\mathbf{y}, \mathbf{q}) = .4$.

What's the probability we do not find q?

# REDUCING FALSE NEGATIVE RATE




Pre-processing:

- Select $t$ independent LSH's $h_1, \ldots, h_t : \{0,1\}^d \to 1, \ldots, m$.
- Create tables $T_1, \ldots, T_t$, each with $m$ slots.
- For $i = 1, \ldots, n$, $j = 1, \ldots, t$,
  - Insert $\mathbf{q}_i$ into $T_j(h_j(\mathbf{q}_i))$.

# REDUCING FALSE NEGATIVE RATE

Query:

- Want to find near neighbors of input $\mathbf{y} \in \{0,1\}^d$.
- Linear scan through all vectors in $T_1(h_1(\mathbf{y})) \cup T_2(h_2(\mathbf{y})) \cup \ldots, T_t(h_t(\mathbf{y}))$.

Suppose the nearest database point $\mathbf{q}$ has $J(\mathbf{y}, \mathbf{q}) = .4$.

What's the probability we find q?

(10, 99%)

# WHAT HAPPENS TO FALSE POSITIVES?

Suppose there is some other database point $\mathbf{z}$ with $J(\mathbf{y}, \mathbf{z}) = .2$.

What is the probability we will need to compute $J(\mathbf{z}, \mathbf{y})$ in our hashing scheme with one table? I.e. the probability that $\mathbf{y}$ hashes into at least one bucket containing $\mathbf{z}$.

In the new scheme with $t = 10$ tables?

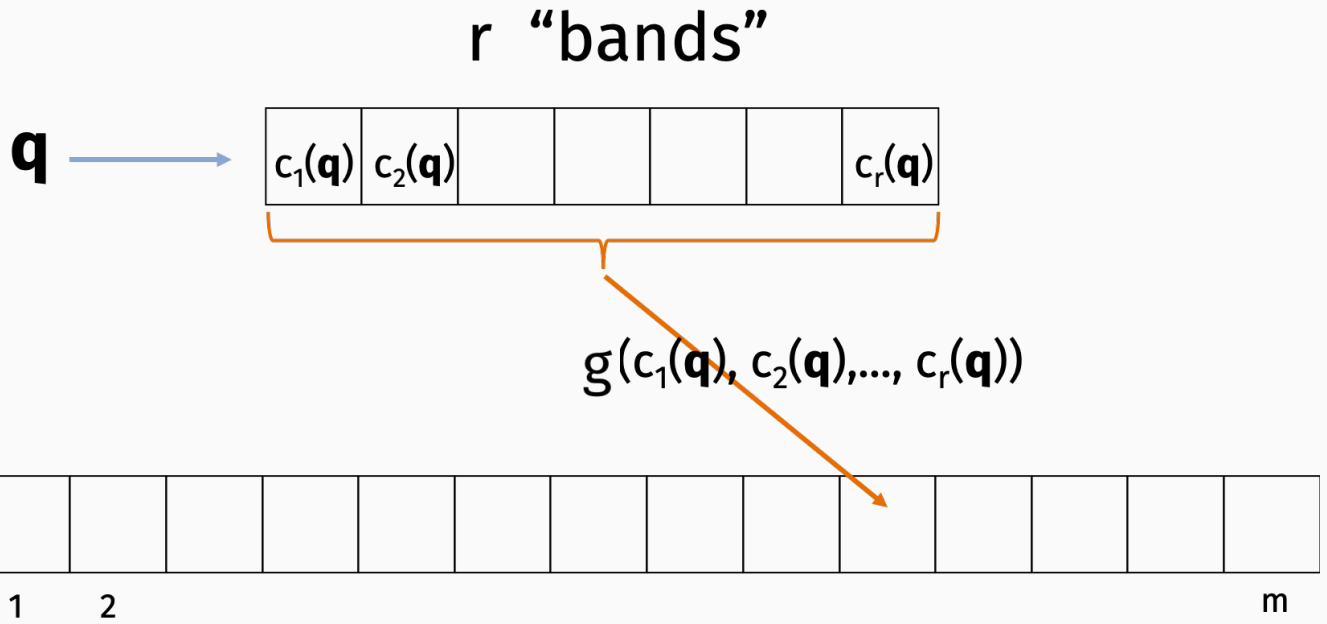
(89%)

# REDUCING FALSE POSITIVES

**Change our locality sensitive hash function.**

Tunable LSH for Jaccard similarity:

- Choose parameter $r \in \mathbb{Z}^+$.
- Let $c_1, \ldots, c_r : \{0,1\}^d \to [0,1]$ be independnt random MinHash's.
- Let $g : [0,1]^r \to \{1, \ldots, m\}$ be a uniform random hash function.
- Let $h(\mathbf{x}) = g(c_1(\mathbf{x}), \ldots, c_r(\mathbf{x}))$.



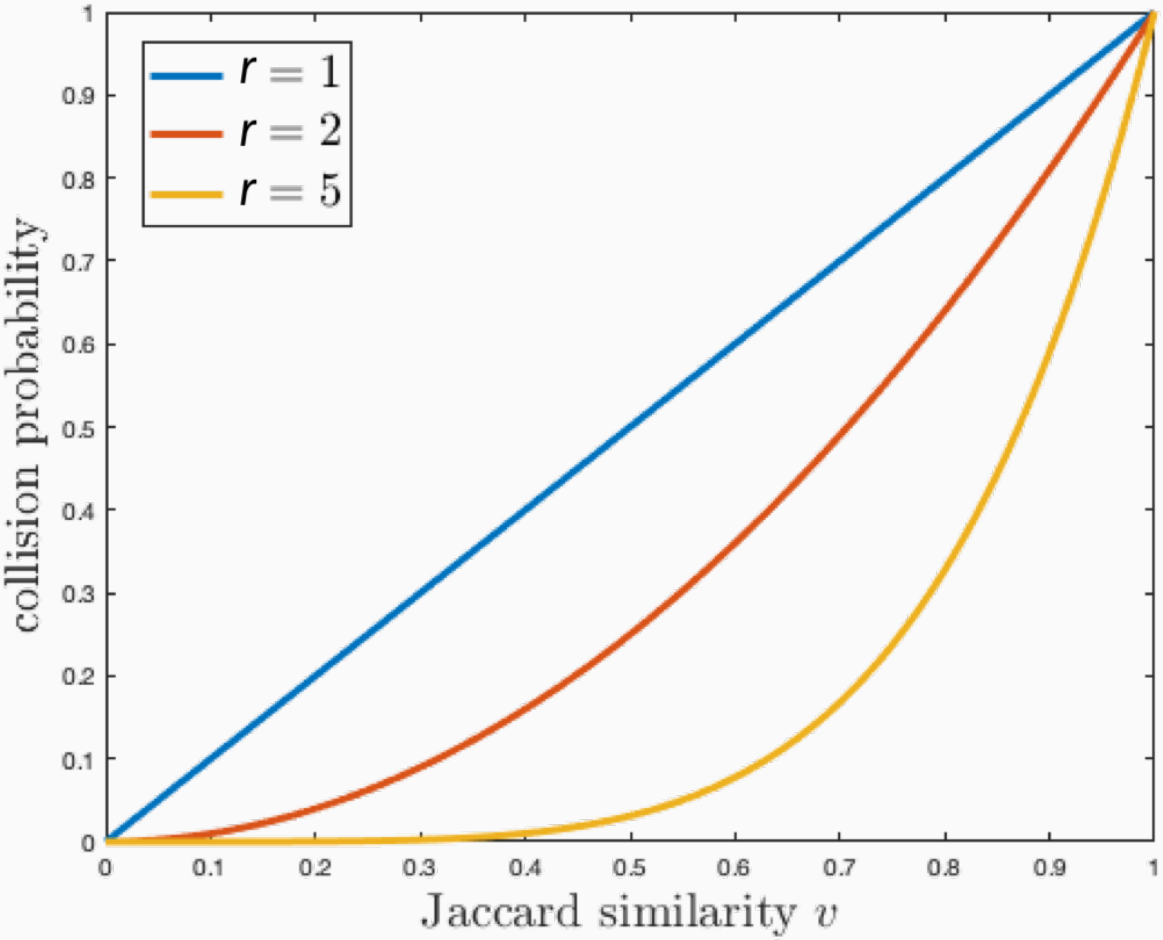
# REDUCING FALSE POSITIVES

Tunable LSH for Jaccard similarity:

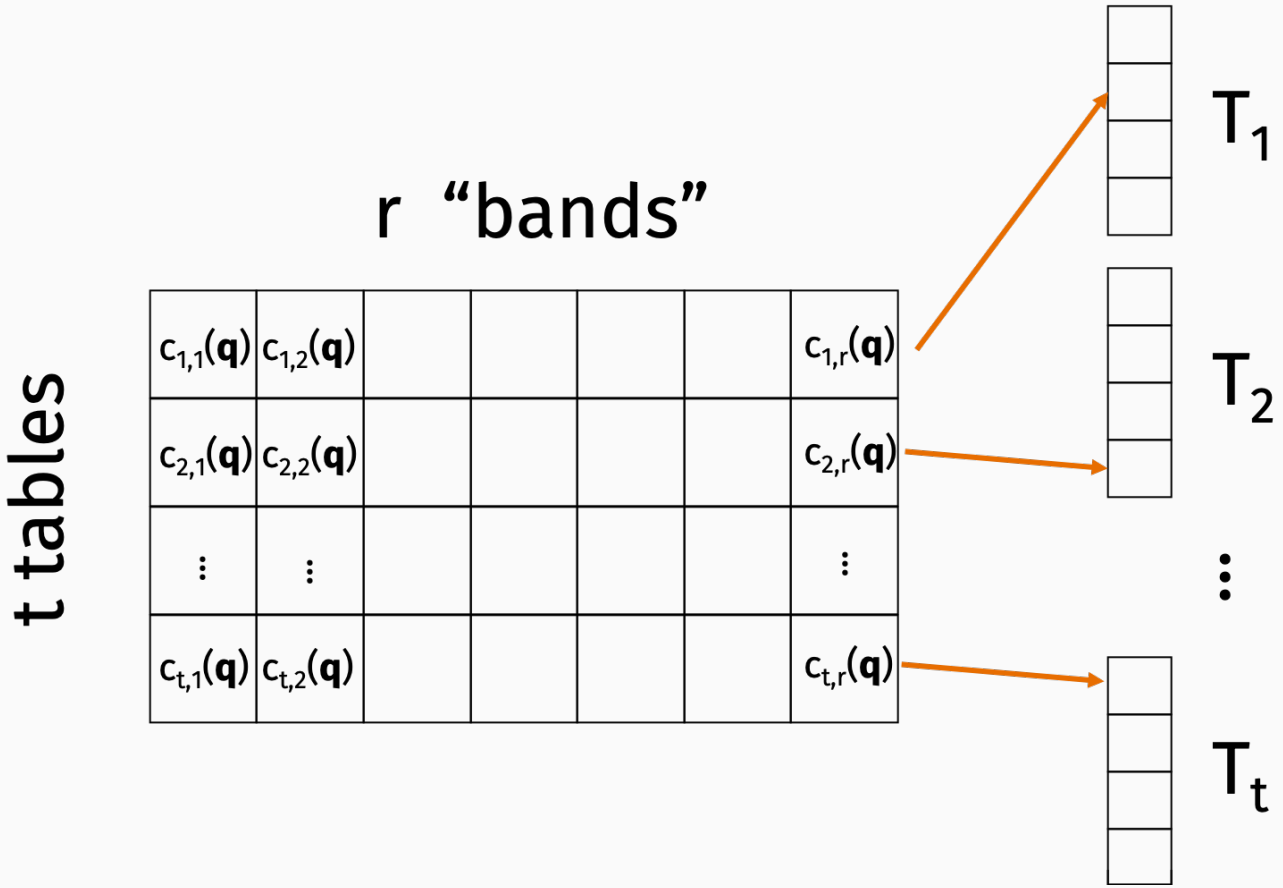
- Choose parameter $r \in \mathbb{Z}^+$.
- Let $c_1, \ldots, c_r : \{0,1\}^d \to [0,1]$ be random MinHash.
- Let $g : [0,1]^r \to \{1, \ldots, m\}$ be a uniform random hash function.
- Let $h(\mathbf{x}) = g(c_1(\mathbf{x}), \ldots, c_r(\mathbf{x}))$.

If $J(\mathbf{q}, \mathbf{y}) = v$, then $\Pr[h(\mathbf{q}) == h(\mathbf{y})] =$

# TUNABLE LSH


r = 1
r = 2
r = 5
collision probability
Jaccard similarity v
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Full LSH cheme has two parameters to tune:



Effect of **increasing number of tables** $t$ on:

False Negatives

False Positives

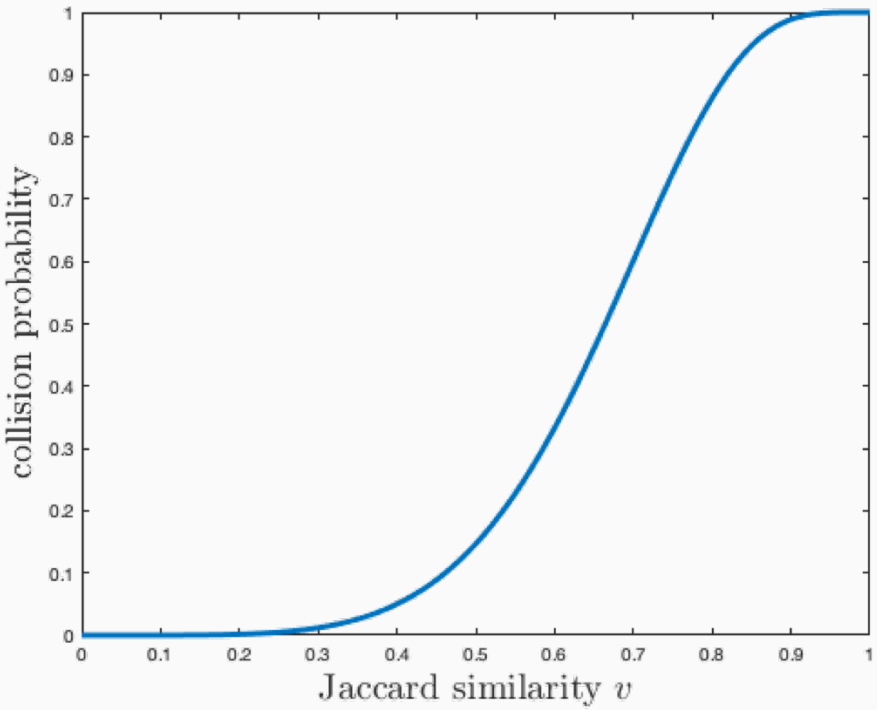
Effect of **increasing number of bands** $r$ on:

False Negatives

False Positives

# S-CURVE TUNING

Probability we check $\mathbf{q}$ when querying $\mathbf{y}$ if $J(\mathbf{q}, \mathbf{y}) = v$:

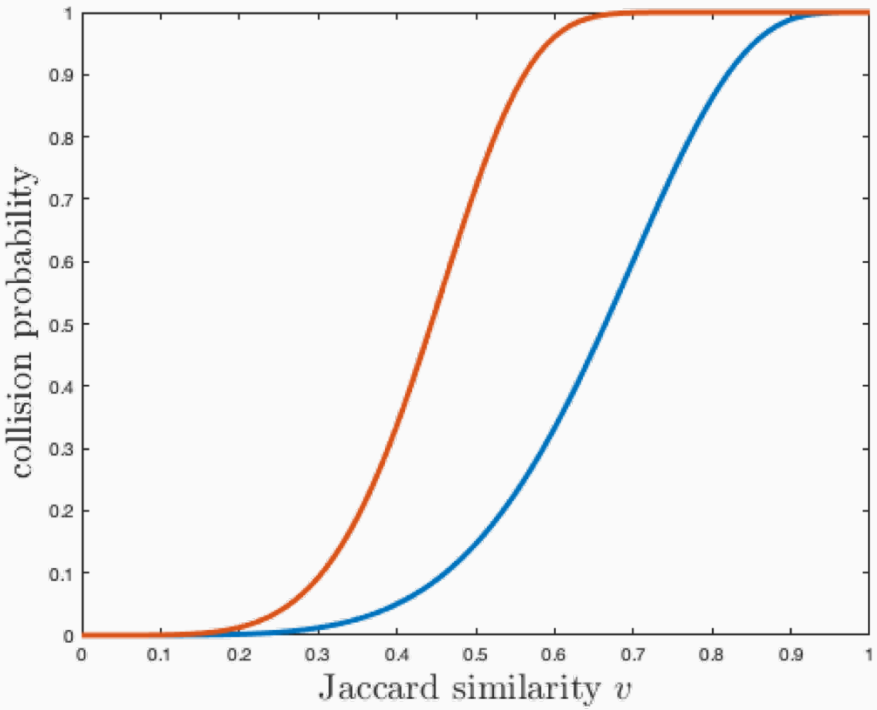



$r = 5, t = 5$

# S-CURVE TUNING

Probability we check $\mathbf{q}$ when querying $\mathbf{y}$ if $J(\mathbf{q}, \mathbf{y}) = v$:

$$\approx 1 - (1 - v^r)^t$$

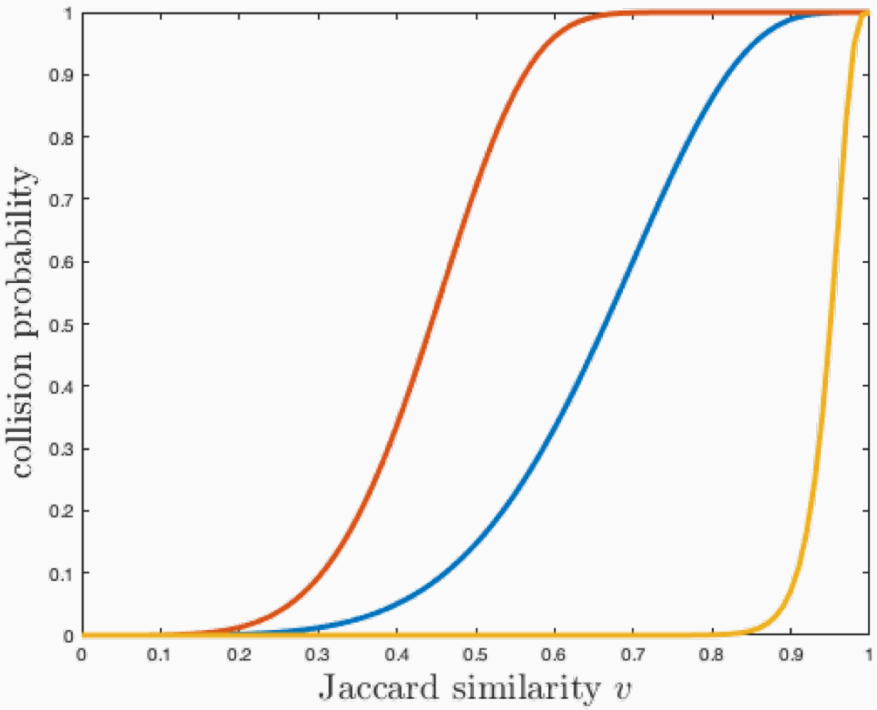



$$r = 5, t = 40$$

# S-CURVE TUNING

Probability we check $\mathbf{q}$ when querying $\mathbf{y}$ if $J(\mathbf{q}, \mathbf{y}) = v$:

$$\approx 1 - (1 - v^r)^t$$

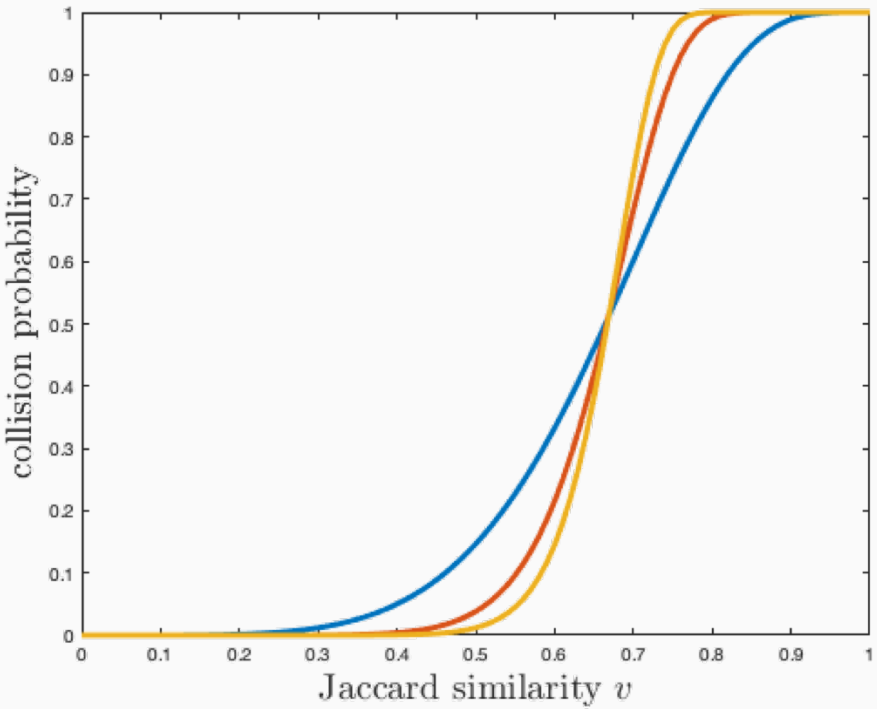



$$r = 40, t = 5$$

# S-CURVE TUNING

Probability we check $\mathbf{q}$ when querying $\mathbf{y}$ if $J(\mathbf{q}, \mathbf{y}) = v$:

$$1 - (1 - v^r)^t$$




Increasing both $r$ and $t$ gives a steeper curve.

Better for search, but worse space complexity.

# FIXED THRESHOLD

**Use Case 1:** Fixed threshold.

- Shazam wants to find match to audio clip $\mathbf{y}$ in a database of 10 million clips.
- There are 10 true matches with $J(\mathbf{y}, \mathbf{q}) > .9$.
- There are 10,000 near matches with $J(\mathbf{y}, \mathbf{q}) \in [.7, .9]$.
- All other items have $J(\mathbf{y}, \mathbf{q}) < .7$.

With $r = 25$ and $t = 40$,

- Hit probability for $J(\mathbf{y}, \mathbf{q}) > .9$ is $\gtrsim 1 - (1 - .9^{25})^{40} = .95$
- Hit probability for $J(\mathbf{y}, \mathbf{q}) \in [.7, .9]$ is $\lesssim 1 - (1 - .9^{25})^{40} = .95$
- Hit probability for $J(\mathbf{y}, \mathbf{q}) < .7$ is $\lesssim 1 - (1 - .7^{25})^{40} = .005$

**Upper bound on total number of items checked:**

$$10 + .95 \cdot 10,000 + .005 \cdot 9,989,990 \approx 60,000 \ll 10,000,000.$$

# FIXED THRESHOLD

Space complexity: 40 hash tables $\approx 40 \cdot O(n)$.

**Directly trade space for fast search.**

Possible to prove concrete worst-case results for distance functions that satisfy triangle inequality.

**Theorem (Indyk, Motwani, 1998. Point Location in Ball)**

*Fix a distance $R$. If there exists some $q$ with $\|\mathbf{q} - \mathbf{y}\|_0 \leq R$, return a vector $\tilde{\mathbf{q}}$ with $\|\tilde{\mathbf{q}} - \mathbf{y}\|_0 \leq C \cdot R$ in:*

- *Time: $O\left(n^{1/C}\right)$.*
- *Space: $O\left(n^{1+1/C} + nd\right)$.*

$\|\mathbf{q} - \mathbf{y}\|_0 =$ "hamming distance" = number of elements that differ between $\mathbf{q}$ and $\mathbf{y}$.

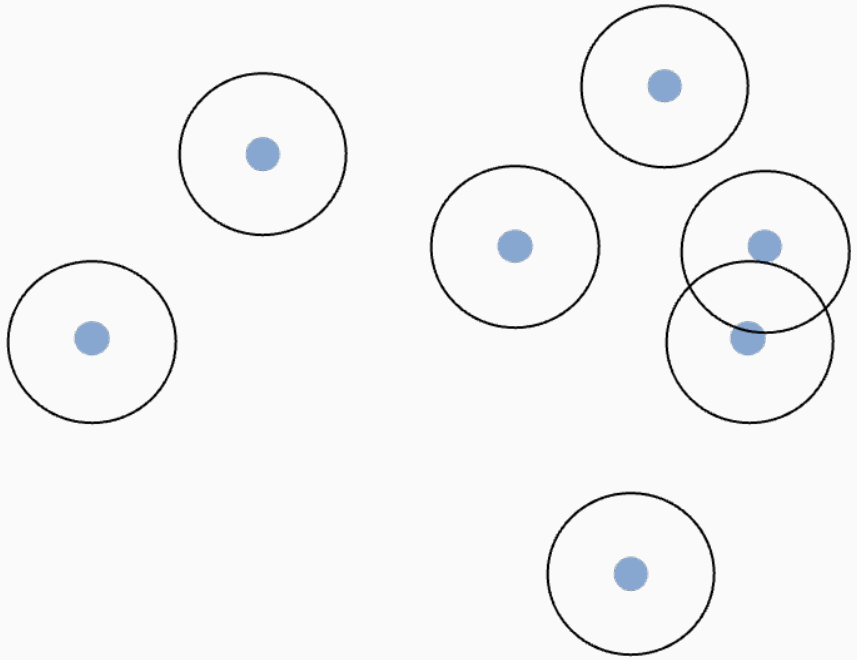
If there is no point at distance $R$, algorithm does not need to return anything.

# LSH based nearest-neighbor in theory

To obtain a nearest-neighbor search algorithm build multiple data structures for exponentially growing distances:

$$R \qquad 2R \qquad 4R \qquad 8R \qquad \ldots$$

Search from most accurate level to least accurate.

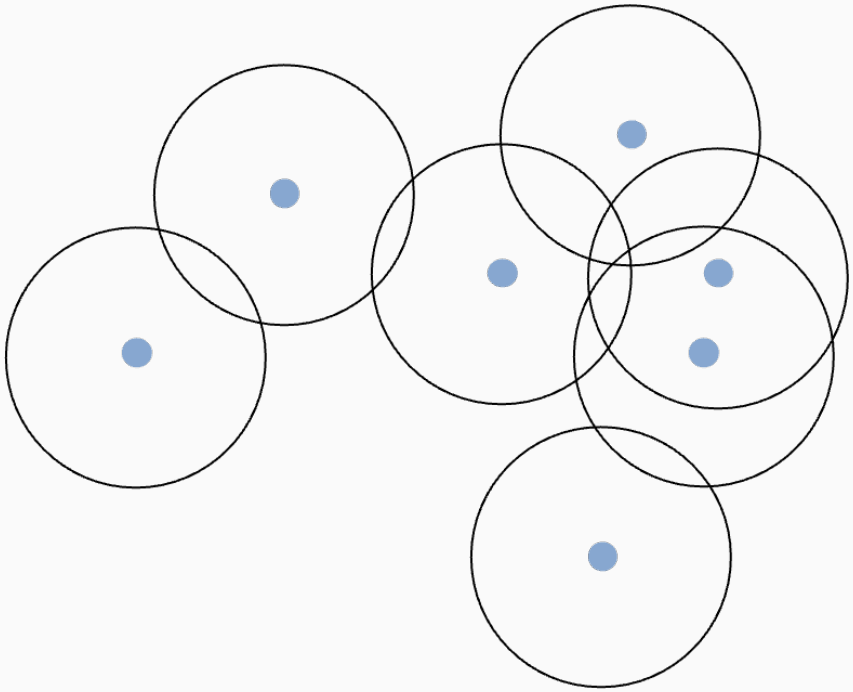


# LSH based nearest-neighbor in theory

To obtain a nearest-neighbor search algorithm build multiple data structures for exponentially growing distances:

$$R \qquad 2R \qquad 4R \qquad 8R \qquad \ldots$$

Search from most accurate level to least accurate.

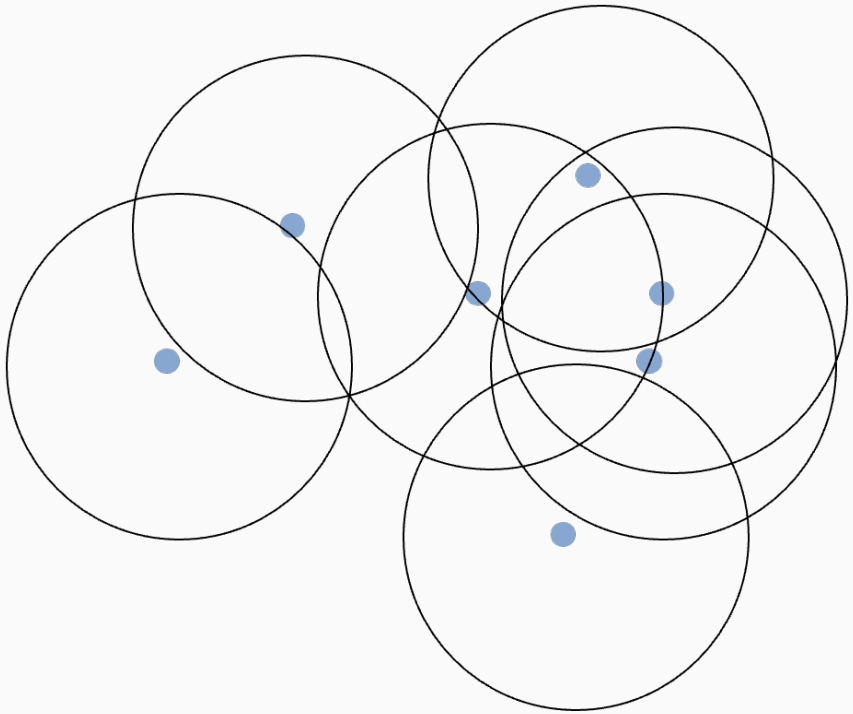


# LSH based nearest-neighbor in theory

To obtain a nearest-neighbor search algorithm build multiple data structures for exponentially growing distances:

$$R \qquad 2R \qquad 4R \qquad 8R \qquad \ldots$$

Search from most accurate level to least accurate.



Total number of levels = $O(\log(d_{\max}/d_{\min}))$, where $d_{\max} = \max_{i,j} \|\mathbf{q}_i - \mathbf{q}_j\|$ and $d_{\min} = \min_{i,j} \|\mathbf{q}_i - \mathbf{q}_j\|$. $d_{\max}/d_{\min}$ is called the **dynamic range**.

**Theorem (Indyk, Motwani, 1998)**

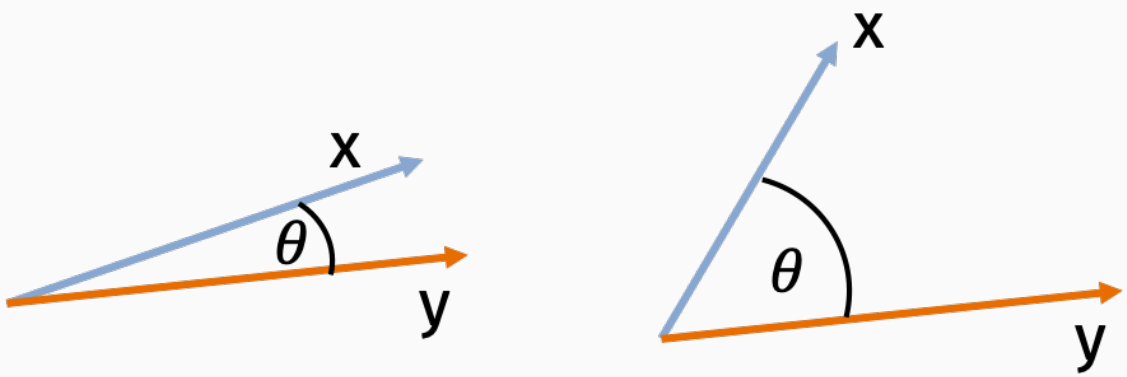
*Let $q$ be the closest database vector to $\mathbf{y}$. Return a vector $\tilde{\mathbf{q}}$ with $\|\tilde{\mathbf{q}} - \mathbf{y}\|_0 \leq C \cdot \|\mathbf{q} - \mathbf{y}\|_0$ in:*

- *Time: $\tilde{O}\left(n^{1/C}\right)$.*
- *Space: $\tilde{O}\left(n^{1+1/C} + nd\right)$.*

Similar results can be proven for other metrics, including Euclidean distance. But you need a good LSH function.

Good locality sensitive hash functions exists for other similarity measures.

**Cosine similarity** $\cos(\theta(x, y)) = \frac{\langle x,y\rangle}{\|x\|_2\|y\|_2}$:




$$-1 \leq \cos(\theta(x, y)) \leq 1.$$

# COSINE SIMILARITY

Cosine similarity is natural "inverse" for Euclidean distance when $\|\mathbf{x}\|_2^2 = \|\mathbf{y}\|_2^2 = 1$ (often the case for ML-based embeddings).

LSH functions also exist for Euclidean distance, but are a bit more complex to describe/analyze. See [Andoni, Indyk, 2006] if you are interested.

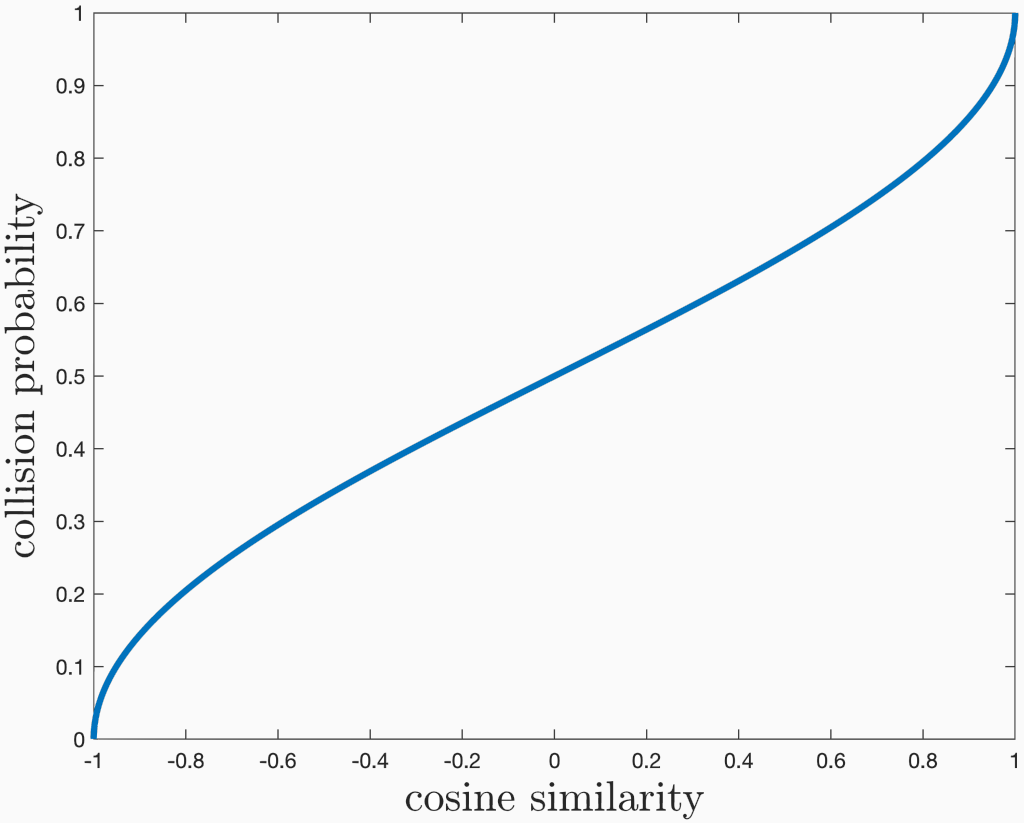
Locality sensitive hash for **cosine similarity**:

- Let $\mathbf{g} \in \mathbb{R}^d$ be randomly chosen with each entry $\mathcal{N}(0,1)$.
- Let $f : \{-1,1\} \to \{1,\ldots,m\}$ be a uniformly random hash function.
- $h : \mathbb{R}^d \to \{1,\ldots,m\}$ is definied $h(\mathbf{x}) = f(\mathsf{sign}(\langle \mathbf{g}, \mathbf{x} \rangle))$.

If $\cos(\theta(\mathbf{x},\mathbf{y})) = v$, what is $\Pr[h(\mathbf{x}) == h(\mathbf{y})]$?

**Theorem (to be proven):** If $\cos(\theta(\mathbf{x}, \mathbf{y})) = v$, then

$$\Pr[h(\mathbf{x}) == h(\mathbf{y})] = 1 - \frac{\theta}{\pi} + \frac{\theta/\pi}{m} = 1 - \frac{\cos^{-1}(v)}{\pi} + \frac{\theta/\pi}{m}$$



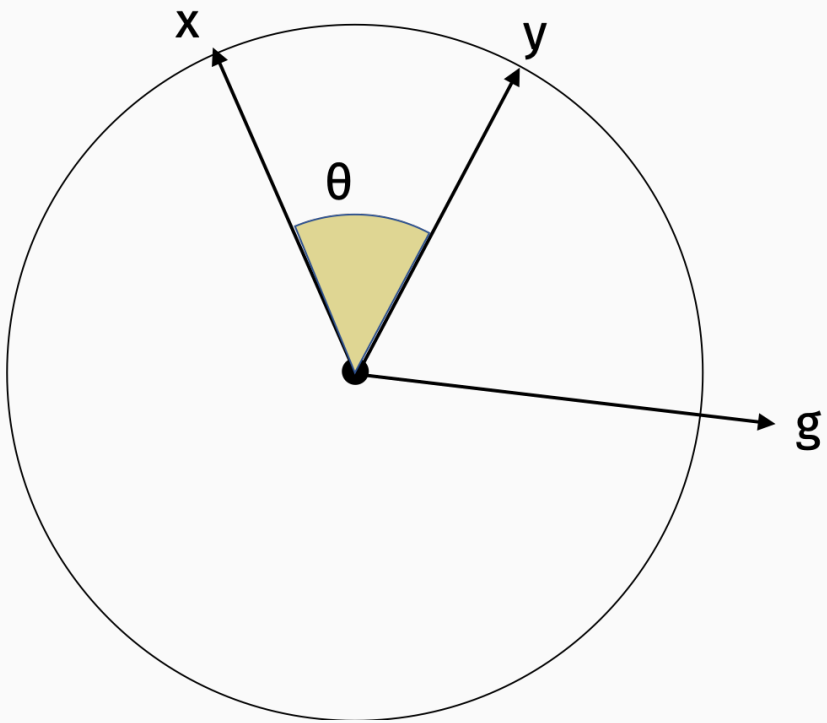
# SIMHASH

SimHash can be banded, just like our MinHash based LSH function for Jaccard similarity:

- Let $\mathbf{g}_1, \ldots, \mathbf{g}_r \in \mathbb{R}^d$ be randomly chosen with each entry $\mathcal{N}(0, 1)$.
- Let $f : \{-1, 1\}^r \to \{1, \ldots, m\}$ be a uniformly random hash function.
- $h : \mathbb{R}^d \to \{1, \ldots, m\}$ is defined $h(\mathbf{x}) = f([\mathsf{sign}(\langle \mathbf{g}_1, \mathbf{x} \rangle), \ldots, \mathsf{sign}(\langle \mathbf{g}_r, \mathbf{x} \rangle)])$.

$$\Pr[h(\mathbf{x}) == h(\mathbf{y})] \approx \left(1 - \frac{\theta}{\pi}\right)^r$$

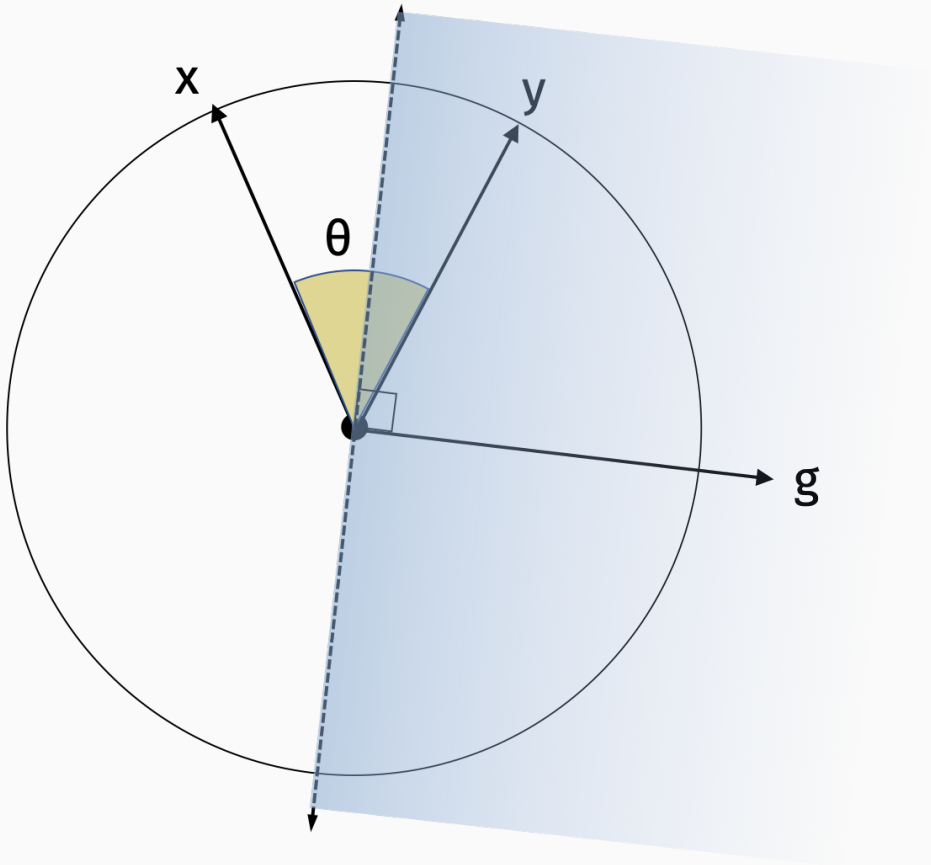
**To prove:** $\Pr[h(\mathbf{x}) == h(\mathbf{y})] \approx 1 - \frac{\theta}{\pi}$, where $h(\mathbf{x}) = f(\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle))$ and $f$ is uniformly random hash function.




$$\Pr[h(\mathbf{x}) == h(\mathbf{y})] = z + \frac{1-z}{m} \approx z.$$

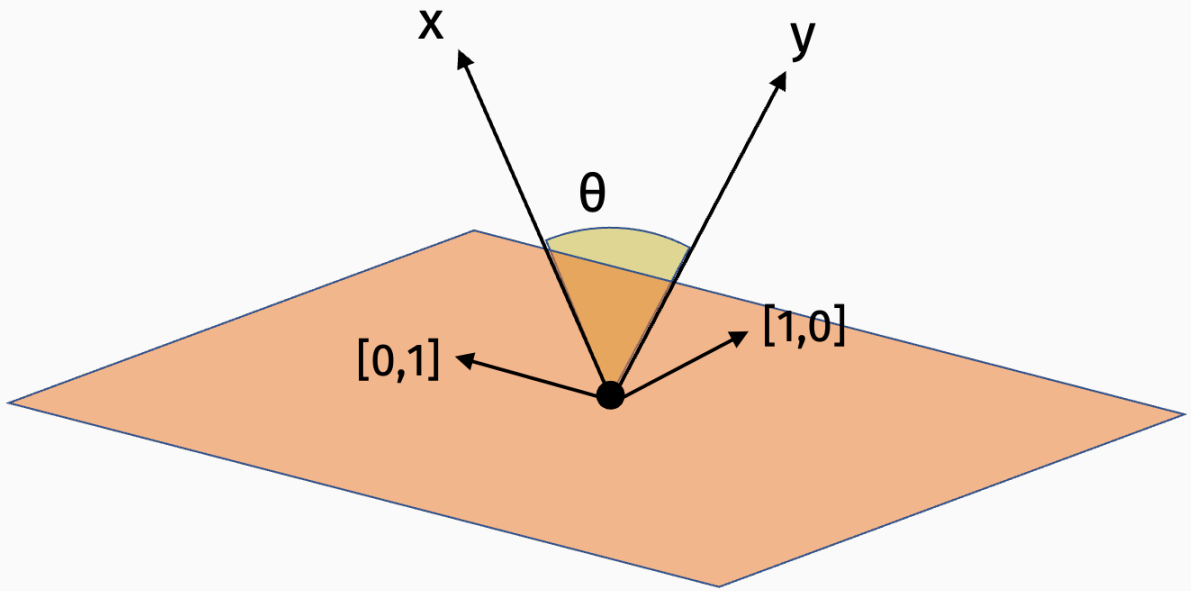
where $z = \Pr[\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle) == \text{sign}(\langle \mathbf{g}, \mathbf{y} \rangle)]$

# SIMHASH ANALYSIS 2D

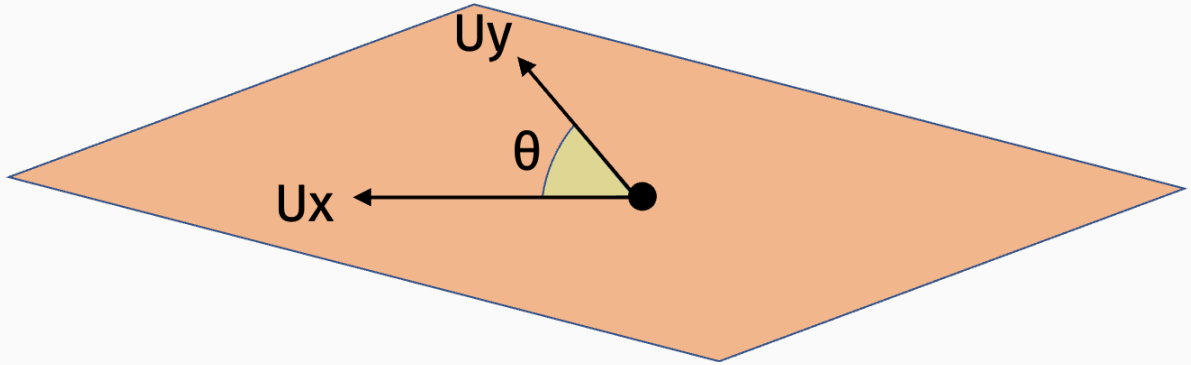



$\Pr[\mathsf{sign}(\langle \mathbf{g}, \mathbf{x} \rangle) == \mathsf{sign}(\langle \mathbf{g}, \mathbf{y} \rangle)] =$ probability $\mathbf{x}$ and $\mathbf{y}$ are on the same side of hyperplane orthogonal to $\mathbf{g}$.

# SIMHASH ANALYSIS HIGHER DIMENSIONS




There is always some rotation matrix **U** such that **Ux**, **Uy** are spanned by the first two-standard basis vectors and have the same cosine similarity as **x** and **y**.




There is always some rotation matrix $\mathbf{U}$ such that $\mathbf{x}, \mathbf{y}$ are spanned by the first two-standard basis vectors.

**Note:** A rotation matrix $\mathbf{U}$ has the property that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. I.e., $\mathbf{U}^T$ is a rotation matrix itself, which reverses the rotation of $\mathbf{U}$.

Claim:

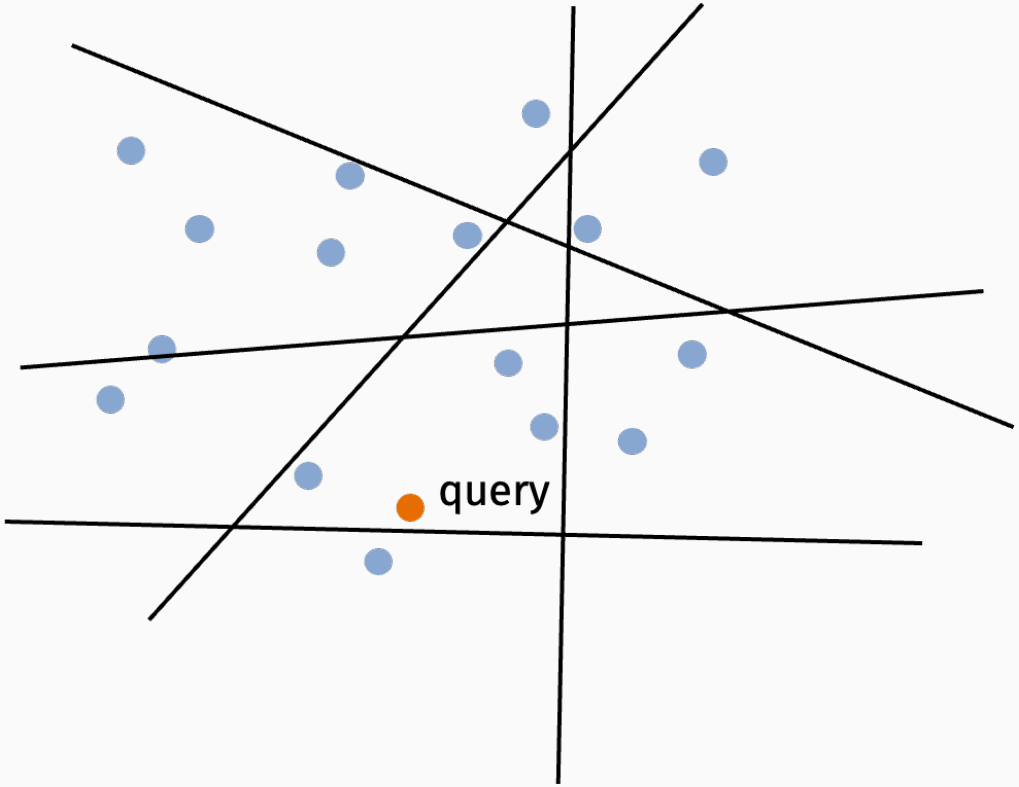
$$\begin{aligned}\Pr[\mathsf{sign}(\langle \mathbf{g}, \mathbf{x}\rangle) == \mathsf{sign}(\langle \mathbf{g}, \mathbf{y}\rangle) &= \Pr[\mathsf{sign}(\langle \mathbf{g}, \mathbf{U}\mathbf{x}\rangle) == \mathsf{sign}(\langle \mathbf{g}, \mathbf{U}\mathbf{y}\rangle)] \\ &= \Pr[\mathsf{sign}(\langle \mathbf{g}[1,2], (\mathbf{U}\mathbf{x})[1,2]\rangle) == \mathsf{sign}(\langle \mathbf{g}[1,2], (\mathbf{U}\mathbf{y}[1,2]\rangle)] \\ &= 1 - \frac{\theta}{\pi}.\end{aligned}$$

The first step is the trickiest here. Why does it hold?

# NEAREST-NEIGHBOR SEARCH IN PRACTICE

LSH is widely used in practice, but is starting to get replaced by other methods. Most of these are data dependent in some way.

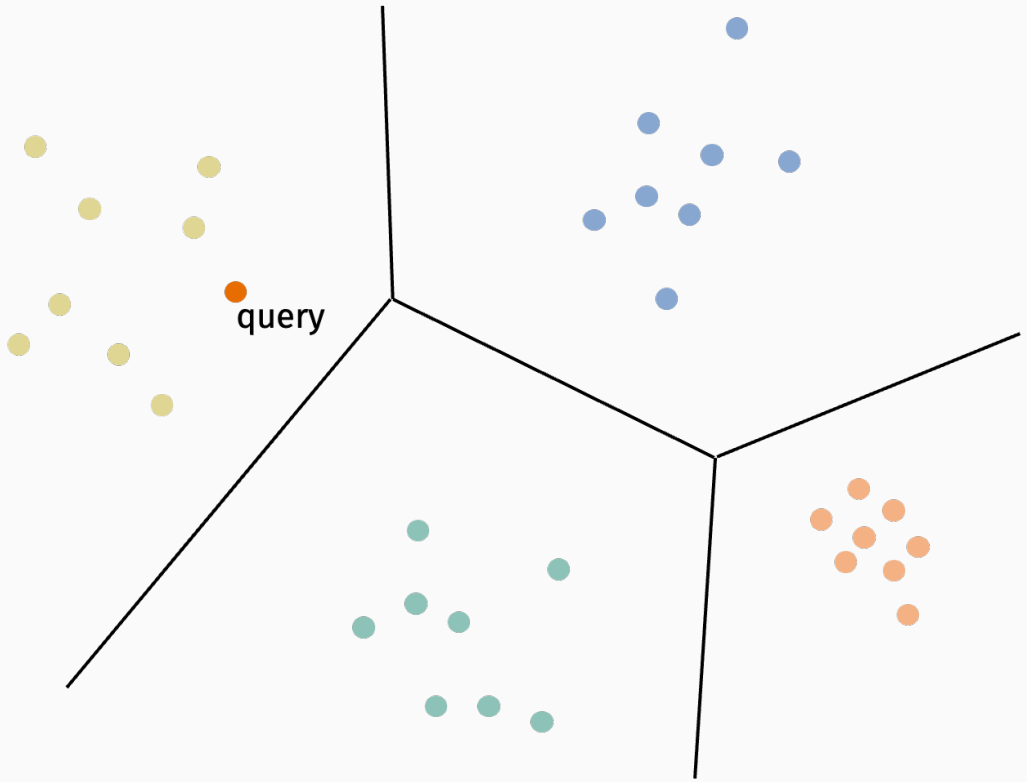
**Starting point:** Think of LSH as a randomized space-partitioning method.



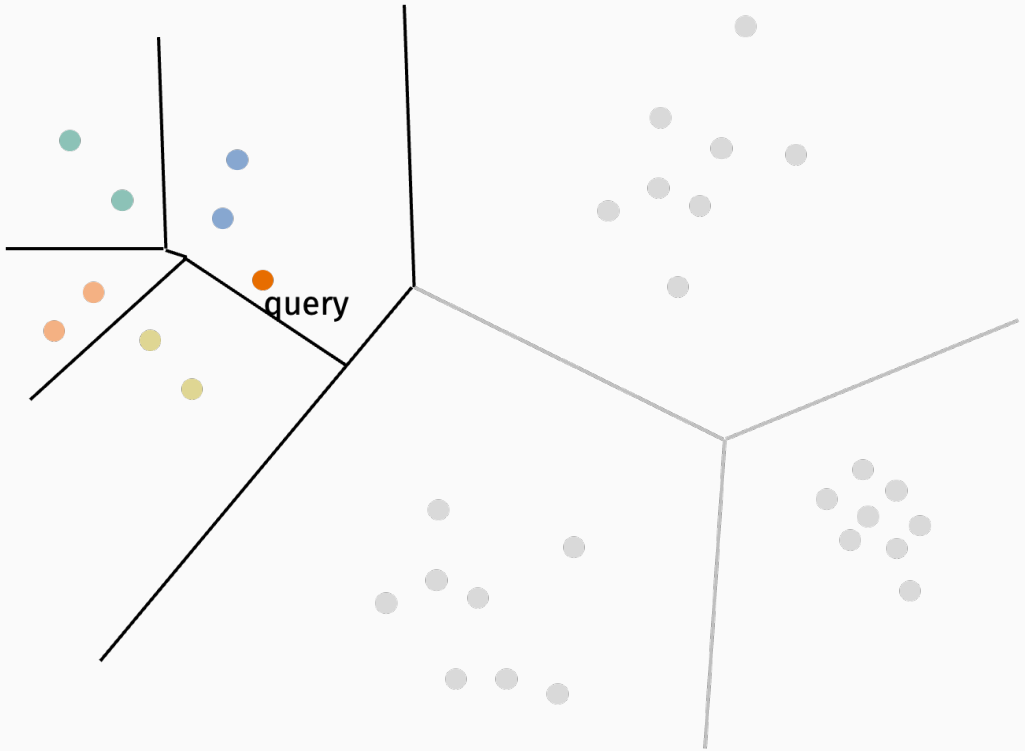
# NEAREST-NEIGHBOR SEARCH IN PRACTICE

In practice, we can often get partitions with better margin but partitioning in a data-dependent way.

**Common approach:** Split data using $k$-means clustering.



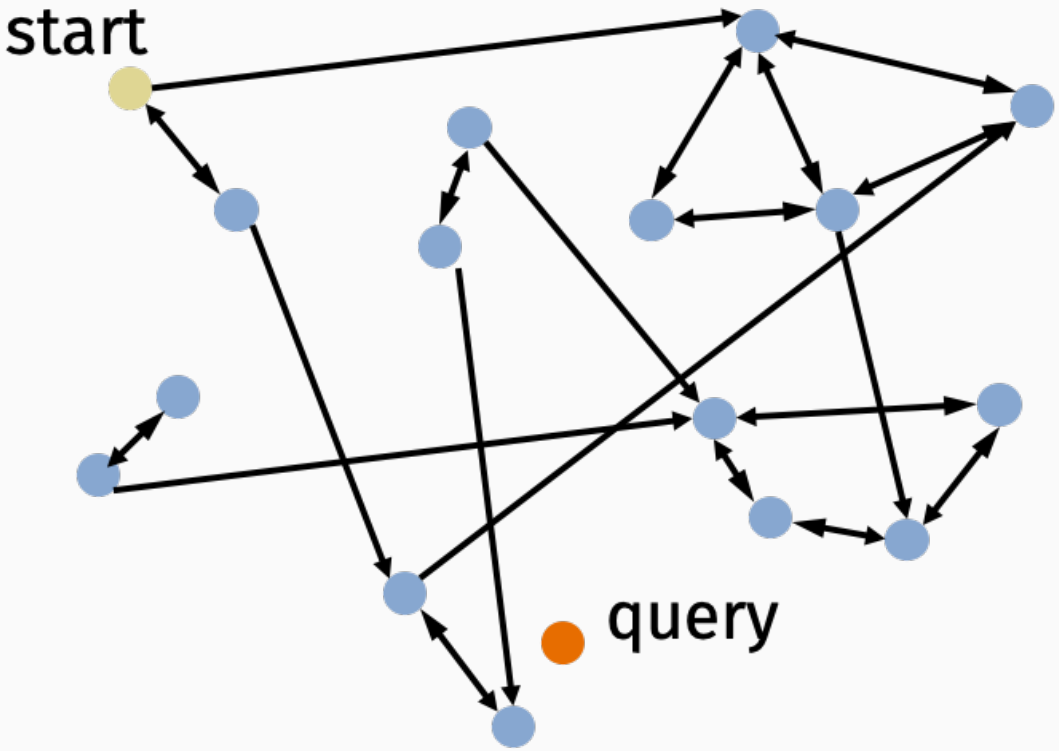
**Common approach:** Split data using $k$-means clustering.




Main approach behind “k-means tree” and “inverted file index” based near-neighbor search methods like Meta’s FAISS library and Google’s SCANN.

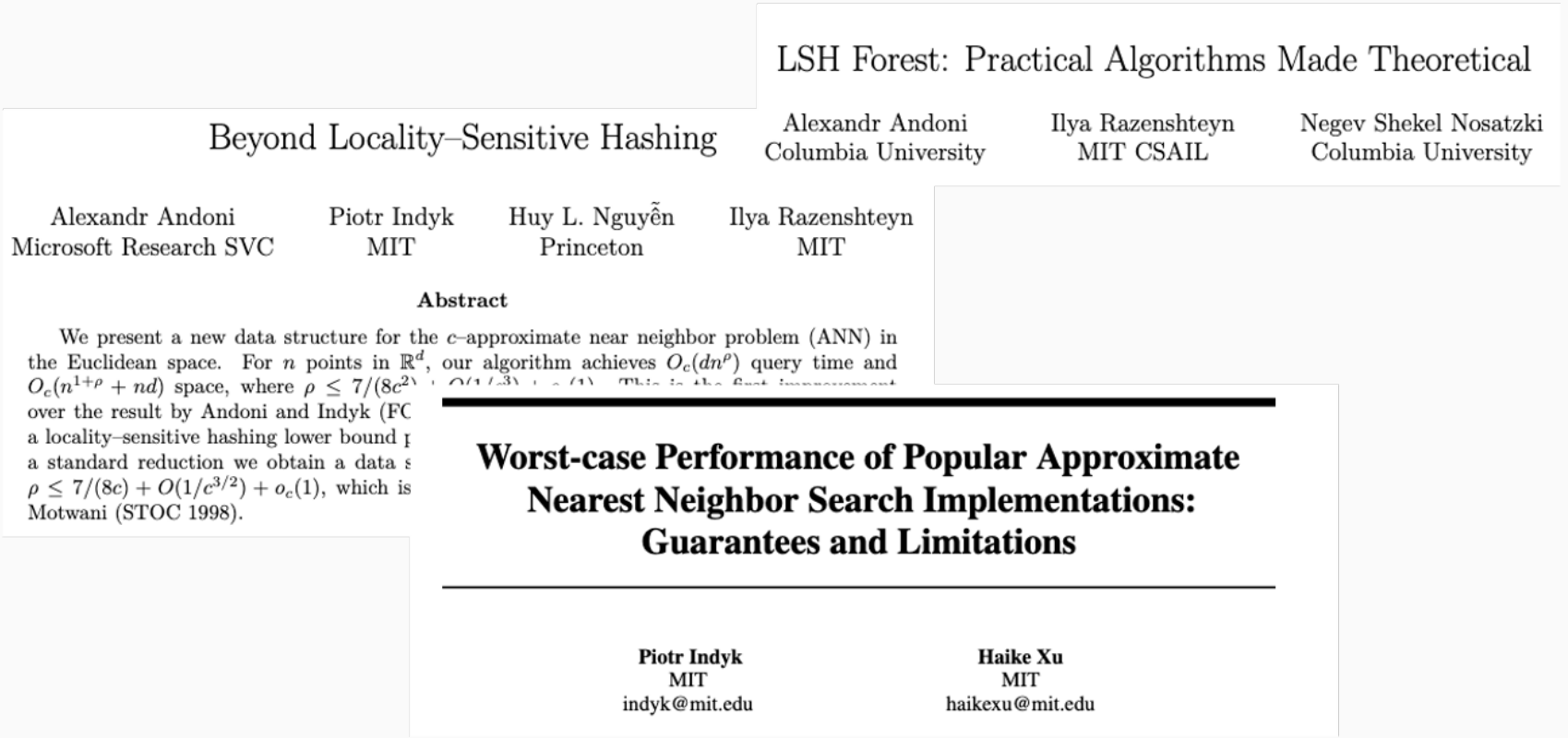
# NEAREST-NEIGHBOR SEARCH IN PRACTICE

**New kid on the block:** Graph-based nearest neighbor search.




Idea behind methods like NSG, HNSW, DiskANN, etc. Inspired by Milgram's famous "small-world" experiments from the 1960's.

Can we better explain the success of data-dependent nearest-neighbor search methods?



LSH Forest: Practical Algorithms Made Theoretical

Alexandr Andoni, Columbia University
Ilya Razenshteyn, MIT CSAIL
Negev Shekel Nosatzki, Columbia University

Beyond Locality–Sensitive Hashing

Alexandr Andoni, Microsoft Research SVC
Piotr Indyk, MIT
Huy L. Nguyễn, Princeton
Ilya Razenshteyn, MIT

**Abstract**

We present a new data structure for the $c$–approximate near neighbor problem (ANN) in the Euclidean space. For $n$ points in $\mathbb{R}^d$, our algorithm achieves $O_c(dn^\rho)$ query time and $O_c(n^{1+\rho} + nd)$ space, where $\rho \leq 7/(8c^2) + O(1/c^3) + o_c(1)$. This is the first improvement over the result by Andoni and Indyk (FC
a locality–sensitive hashing lower bound
a standard reduction we obtain a data
$\rho \leq 7/(8c) + O(1/c^{3/2}) + o_c(1)$, which is
Motwani (STOC 1998).

**Worst-case Performance of Popular Approximate Nearest Neighbor Search Implementations: Guarantees and Limitations**

**Piotr Indyk**
MIT
indyk@mit.edu

**Haike Xu**
MIT
haikexu@mit.edu