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

NYU, Prof. Ainesh Bakshi

Last Class

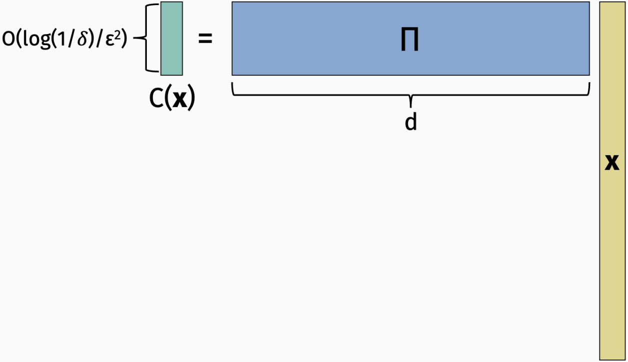
Dimensionality reduction: Given vectors $\mathbf{x}, \mathbf{y}$, compute small space compressions $C(\mathbf{x})$ and $C(\mathbf{y})$ that can be used to estimate the distance or similarity between $\mathbf{x}$ and $\mathbf{y}$.

Euclidean Dimensionality Reduction

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* $(1-\delta)$*:*

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

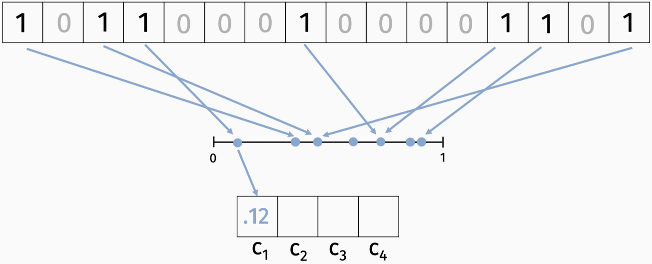


Dimensionality Reduction for Jaccard Similarity

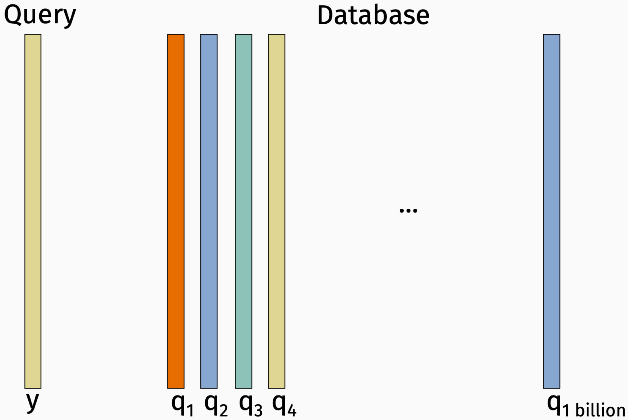
Lemma (MinHash)

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:

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



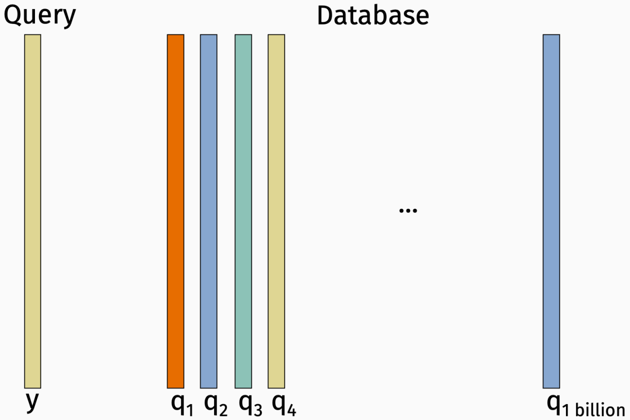
Key Application: Modern Vector Search



Goal: Find $\arg\max_i \mathrm{dist}(\mathbf{q}_i, \mathbf{q})$

Cost of naive algorithm is

Key Application: Modern Vector Search

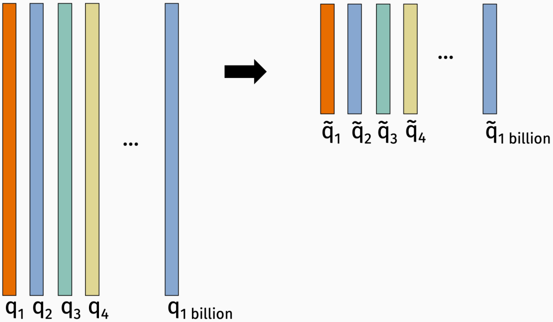


Goal: Find $\arg\max_i \mathrm{dist}(\mathbf{q}_i, \mathbf{q})$

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

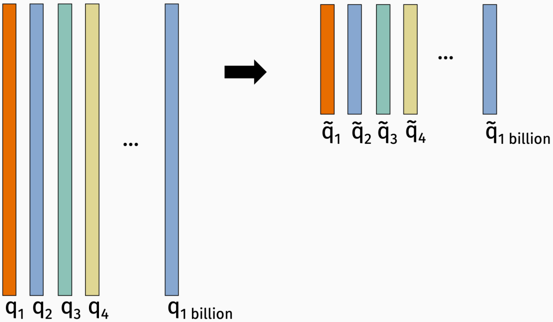
Key Application: Modern Vector Search

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



Key Application: Modern Vector 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

Vector Indexing / Near Neighbor Search

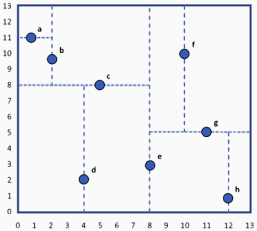
Dimensionality reduction methods are typically paired with vector indexing methods.

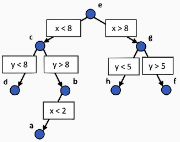
Goal of Dimensionality Reduction: Reduce **dependence on** d in $O(nd)$ search cost. Reduce space complexity.

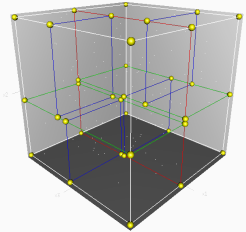
Goal of Vector Indexing: Reduce **dependence on** n in $O(nd)$ search cost. Often at the cost of added space complexity.

Beyond a Linear Scan

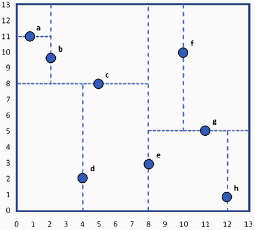
This problem can already be solved in low-dimensions using space partitioning approaches (namely, kd-trees).

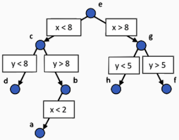


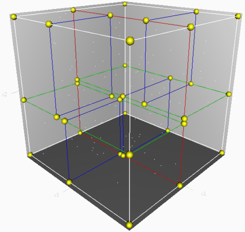




This problem can already be solved in low-dimensions using space partitioning approaches (namely, kd-trees).



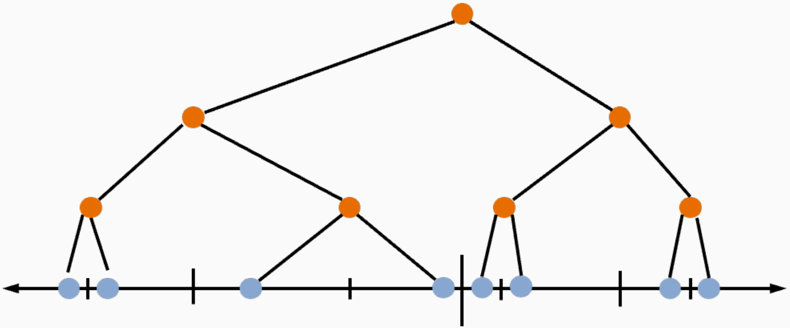




Search time is roughly $O(d \cdot \log n \cdot 2^d))$, which is only sublinear for $d = o(\log n)$.

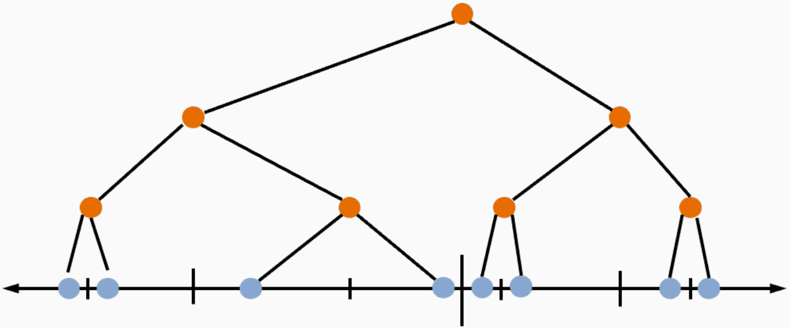
Issue with KD-Trees

How do we build the tree?



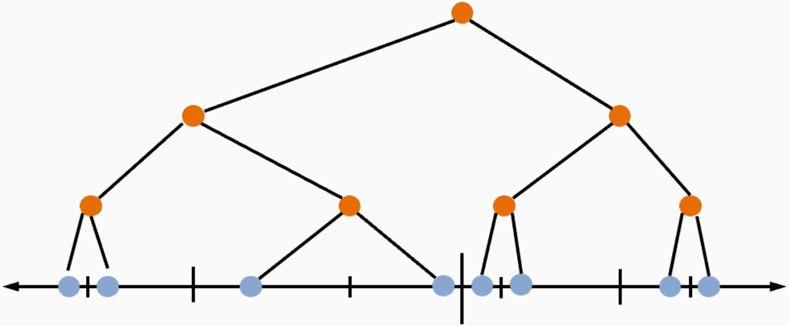
Issue with KD-Trees

How do we build the tree?



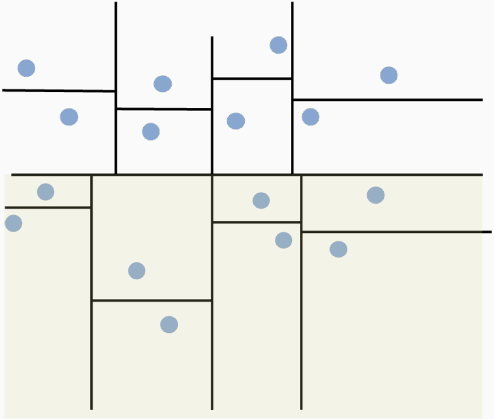
Given a query q, how do we find the nearest neighbor?

Issue with KD-Trees



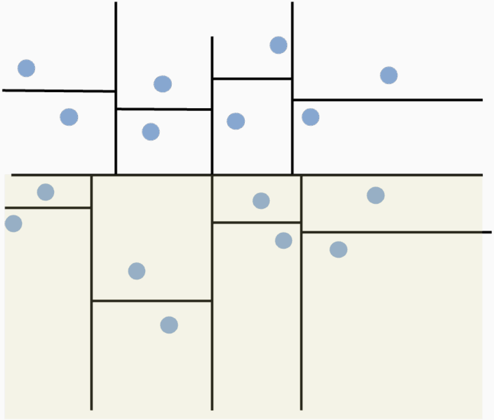
Are we guaranteed to get the nearest neighbor in the worst case?

Issue with KD-Trees



How many boxes do I need to check now?

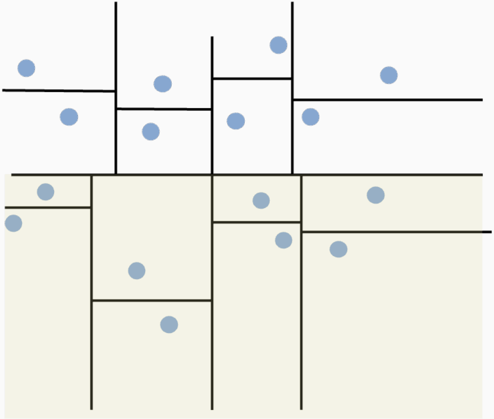
Issue with KD-Trees



How many boxes do I need to check now?

2^d in d dimensions, so query time is exponential in d.

Issue with KD-Trees



Theorem (informal): Exact nearest neighbor search in d dimensions requires exponential (in d) space and time.

High Dimensional Near Neighbor Search

Only been attacked much more recently:

- **Locality-sensitive hashing [Indyk, Motwani, 1998]**

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 query time.

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 query time.
2. Preprocess and create data structure that uses $\Omega(n)$ space.

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 query time.
2. Preprocess and create data structure that uses $\Omega(n)$ space.
3. Allow for approximation.

Intuitively Why Do Preprocessing and Space Help?

Why does extra space help?

Intuitively Why Do Preprocessing and Space Help?

Why does extra space help?

Question: Dataset is n points in the cube $[-1,1]^d$. Given ϵ and a query $\mathbf{q} \in [-1,1]^d$, you want to find $\tilde{\mathbf{y}}$ with

$$\|\mathbf{q} - \tilde{\mathbf{y}}\|_2 \leq \min_i \|\mathbf{q} - \mathbf{v}_i\|_2 + \epsilon.$$

Intuitively Why Do Preprocessing and Space Help?

Why does extra space help?

Question: Dataset is n points in the cube $[-1,1]^d$. Given ϵ and a query $\mathbf{q} \in [-1,1]^d$, you want to find $\tilde{\mathbf{y}}$ with

$$\|\mathbf{q} - \tilde{\mathbf{y}}\|_2 \leq \min_i \|\mathbf{q} - \mathbf{v}_i\|_2 + \epsilon.$$

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

Locality Sensitive Hashing

Key Idea: Map close by points in R^d to the same bucket. Map far away points to different buckets.

Locality Sensitive Hashing

Key Idea: Map close by points in R^d to the same bucket. Map far away points to different buckets.

Why does such a function even exist? It's the complete opposite of uniformly random hash functions.

Locality Sensitive Hash Functions

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

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.

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:

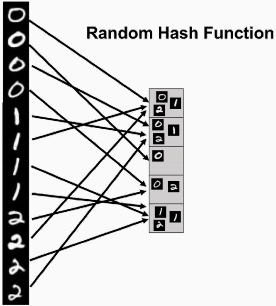
- Higher when $\mathbf{q}$ and $\mathbf{y}$ are more similar, i.e. $s(\mathbf{q}, \mathbf{y})$ is higher.
- Lower when $\mathbf{q}$ and $\mathbf{y}$ are more dissimilar, i.e. $s(\mathbf{q}, \mathbf{y})$ is lower.

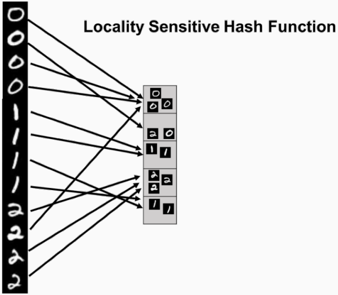
Locality Sensitive Hash Functions

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 $c : \{0, 1\}^d \rightarrow [0, 1]$ be a single instantiation of MinHash.

Locality Sensitive Hash Functions

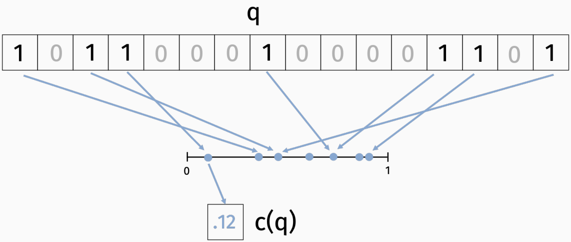
LSH for $s(\mathbf{q}, \mathbf{y})$ equal to Jaccard similarity:

- Let $c : \{0,1\}^d \rightarrow [0,1]$ be a single instantiation of MinHash.
- Let $g : [0,1] \rightarrow \{1, \ldots, m\}$ be a uniform random hash function.

Locality Sensitive Hash Functions

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

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



Why should this be locality sensitive?

Locality Sensitive Hash Functions

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

- Let $c : \{0,1\}^d \rightarrow [0,1]$ be a single instantiation of MinHash.

Locality Sensitive Hash Functions

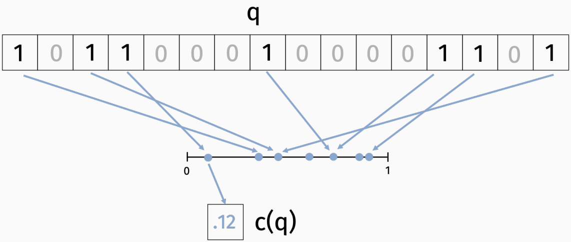
LSH for $s(\mathbf{q}, \mathbf{y})$ equal to Jaccard similarity:

- 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.

Locality Sensitive Hash Functions

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

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



$$Pr[c(\mathbf{q}) == c(\mathbf{y})] = J(\mathbf{q}, \mathbf{y}) \text{ and } c(\mathbf{q}) = c(\mathbf{y}) \text{ implies } g(c(\mathbf{q})) = g(c(\mathbf{y})).$$

Locality Sensitive Hash Functions

LSH for Jaccard similarity:

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

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

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

Locality Sensitive Hash Functions

LSH for Jaccard similarity:

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

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

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

$$= J(\mathbf{q}, \mathbf{y}) + \underbrace{(1 - J(\mathbf{q}, \mathbf{y}))\frac{1}{m}}_{\text{negligible}}$$

Near Neighbor Search

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

Pre-processing:

- Select random LSH function $h : \{0, 1\}^d \rightarrow 1, \ldots, m$.

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

Near Neighbor Search

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]

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

Near Neighbor Search

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

Pre-processing:

- Select random LSH function $h : \{0,1\}^d \rightarrow 1, \ldots, m$.
- 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 $\mathbf{y} \in \{0,1\}^d$.

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

Near Neighbor Search

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 $\mathbf{y} \in \{0,1\}^d$.

- Hash $\mathbf{y}$ to get $h(\mathbf{y})$.

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

Near Neighbor Search

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 $\mathbf{y} \in \{0,1\}^d$.

- Hash $\mathbf{y}$ to get $h(\mathbf{y})$.
- Go to slot $T(h(\mathbf{y}))$. Linear scan through all vectors $\mathbf{q} \in T(h(\mathbf{y}))$ and return any that are close to $\mathbf{y}$.

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

Near Neighbor Search

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

Pre-processing:

- Select random LSH function $h : \{0,1\}^d \rightarrow 1, \ldots, m$.
- 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 $\mathbf{y} \in \{0,1\}^d$.

- Hash $\mathbf{y}$ to get $h(\mathbf{y})$.
- Go to slot $T(h(\mathbf{y}))$. 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 $\mathbf{y}$?

Near Neighbor Search

Two main considerations:

- **False Negative Rate**: What's the probability we do not find a vector that is close to $\mathbf{y}$?
  - A higher false negative rate means we miss near neighbors.

Near Neighbor Search

Two main considerations:

- **False Negative Rate**: What's the probability we do not find a vector that is close to $\mathbf{y}$?
  - A higher false negative rate means we miss near neighbors.
- **False Positive Rate**: What's the probability that a vector in $T(h(\mathbf{y}))$ is not close to $\mathbf{y}$?

Near Neighbor Search

Two main considerations:

- **False Negative Rate**: What's the probability we do not find a vector that is close to $\mathbf{y}$?
  - A higher false negative rate means we miss near neighbors.
- **False Positive Rate**: What's the probability that a vector in $T(h(\mathbf{y}))$ is not close to $\mathbf{y}$?
  - 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 $\mathbf{y}$.

Near Neighbor Search

Two main considerations:

- **False Negative Rate**: What's the probability we do not find a vector that is close to $\mathbf{y}$?
  - A higher false negative rate means we miss near neighbors.
- **False Positive Rate**: What's the probability that a vector in $T(h(\mathbf{y}))$ is not close to $\mathbf{y}$?
  - 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.

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

What's the probability we do not find q?

$$\Pr[\text{We find } \mathbf{q}] = \Pr[h(\mathbf{q}) == h(\mathbf{y})] \approx 0.4$$

Reducing False Negative Rate

Let's use Jaccard similarity as a running example.

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

What's the probability we do not find q?

$$\Pr[\text{We find } \mathbf{q}] = \Pr[h(\mathbf{q}) == h(\mathbf{y})] \approx 0.4$$

$$\Pr[\text{We do not find } \mathbf{q}] \approx 1 - 0.4 = 0.6$$

Reducing False Negative Rate

Let's use Jaccard similarity as a running example.

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

What's the probability we do not find q?

$$\Pr[\text{We find } \mathbf{q}] = \Pr[h(\mathbf{q}) == h(\mathbf{y})] \approx 0.4$$

$$\Pr[\text{We do not find } \mathbf{q}] \approx 1 - 0.4 = 0.6$$

Reducing False Negative Rate

Let's use Jaccard similarity as a running example.

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

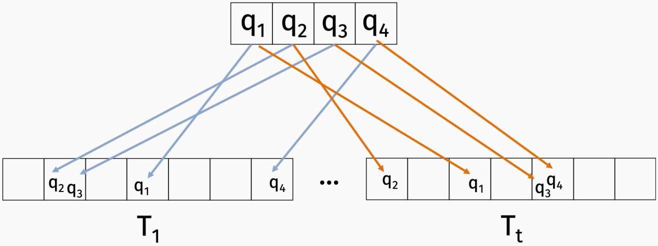
What's the probability we do not find q?

$$\Pr[\text{We find } \mathbf{q}] = \Pr[h(\mathbf{q}) == h(\mathbf{y})] \approx 0.4$$

$$\Pr[\text{We do not find } \mathbf{q}] \approx 1 - 0.4 = 0.6$$

This is quite bad, how do we dampen the false negative rate?

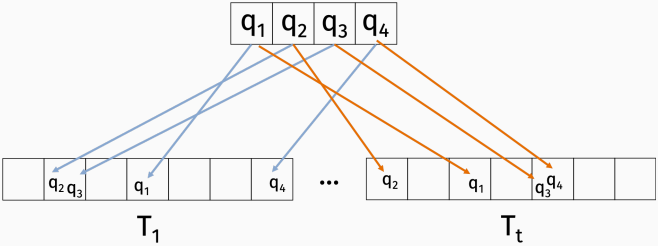
Reducing False Negative Rate



Pre-processing:

- Select t independent LSH's $h_1, \ldots, h_t : \{0,1\}^d \to 1, \ldots, m$.

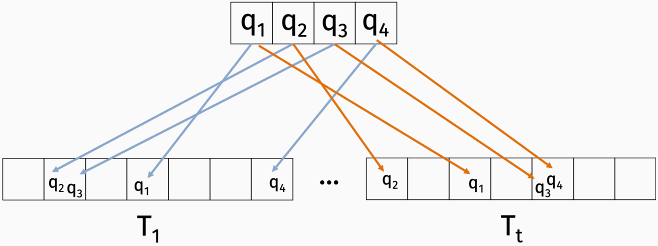
Reducing False Negative Rate



Pre-processing:

- Select t independent LSH's $h_1, \ldots, h_t : \{0,1\}^d \rightarrow 1, \ldots, m$.
- Create tables $T_1, \ldots, T_t$, each with m slots.

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 $\mathbf{y} \in \{0,1\}^d$.

- Computer $h_1(\mathbf{y}), \ldots, h_t(\mathbf{y})$.

Reducing False Negative Rate

Query: Want to find near neighbors of input $\mathbf{y} \in \{0,1\}^d$.

- Computer $h_1(\mathbf{y}), \ldots, h_t(\mathbf{y})$.
- Linear scan through all vectors in $T_1(h_1(\mathbf{y})) \cup T_2(h_2(\mathbf{y})) \cup \ldots, T_t(h_t(\mathbf{y}))$.

Reducing False Negative Rate

Query: Want to find near neighbors of input $\mathbf{y} \in \{0,1\}^d$.

- Computer $h_1(\mathbf{y}), \ldots, h_t(\mathbf{y})$.
- 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?

Reducing False Negative Rate

Query: Want to find near neighbors of input $\mathbf{y} \in \{0,1\}^d$.

- Computer $h_1(\mathbf{y}), \ldots, h_t(\mathbf{y})$.
- 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?

$$\Pr[\text{We find } \mathbf{q}] = 1 - \Pr[\text{We do not find } \mathbf{q} \text{ in any table }]$$

Reducing False Negative Rate

Query: Want to find near neighbors of input $\mathbf{y} \in \{0,1\}^d$.

- Computer $h_1(\mathbf{y}), \ldots, h_t(\mathbf{y})$.
- 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?

$$\begin{aligned}\Pr[\text{We find } \mathbf{q}] &= 1 - \Pr[\text{We do not find } \mathbf{q} \text{ in any table }] \\ &= 1 - (1-0.4)^t = 1 - 0.6^t\end{aligned}$$

Reducing False Negative Rate

Query: Want to find near neighbors of input $\mathbf{y} \in \{0,1\}^d$.

- Computer $h_1(\mathbf{y}), \ldots, h_t(\mathbf{y})$.
- 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?

$$\begin{aligned}
\Pr[\text{We find } \mathbf{q}] &= 1 - \Pr[\text{We do not find } \mathbf{q} \text{ in any table }] \\
&= 1 - (1 - 0.4)^t = 1 - 0.6^t \\
&\approx 99\% \text{ for } t = 10
\end{aligned}$$

What Happens to False Positives?

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

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?

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?

$$\Pr[\text{We find } \mathbf{z}] = 1 - (1 - 0.2)^t = 1 - 0.8^t$$

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?

$$\Pr[\text{We find } \mathbf{z}] = 1 - (1 - 0.2)^t = 1 - 0.8^t$$
$$\approx 89\% \text{ for } t = 10$$

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?

$$\Pr[\text{We find } \mathbf{z}] = 1 - (1 - 0.2)^t = 1 - 0.8^t$$
$$\approx 89\% \text{ for } t = 10$$

What Happens to False Positives?

Suppose there is some other database point **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?

$$\Pr[\text{We find } \mathbf{z}] = 1 - (1 - 0.2)^t = 1 - 0.8^t$$
$$\approx 89\% \text{ for } t = 10$$

This is really bad! All the other points could have similarity .2

We have to search over $\Omega(n)$ points anyway.

Reducing False Positives

Secret Sauce: **Change our locality sensitive hash function.**

Reducing False Positives

Secret Sauce: **Change our locality sensitive hash function**.

Tunable LSH for Jaccard similarity:

- Choose parameter $r \in \mathbb{Z}^+$.

Reducing False Positives

Secret Sauce: Change our locality sensitive hash function.

Tunable LSH for Jaccard similarity:

- Choose parameter $r \in \mathbb{Z}^+$.
- Let $c_1, \ldots, c_r : \{0,1\}^d \rightarrow [0,1]$ be independent random MinHash's.

Reducing False Positives

Secret Sauce: Change our locality sensitive hash function.

Tunable LSH for Jaccard similarity:

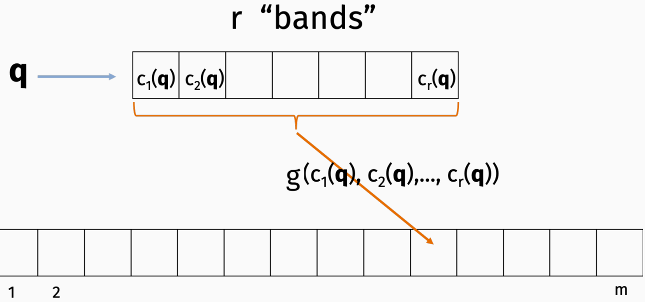
- Choose parameter $r \in \mathbb{Z}^+$.
- Let $c_1, \ldots, c_r : \{0,1\}^d \rightarrow [0,1]$ be independent random MinHash's.
- Let $g : [0,1]^r \rightarrow \{1, \ldots, m\}$ be a uniform random hash function.

Reducing False Positives

Secret Sauce: Change our locality sensitive hash function.

Tunable LSH for Jaccard similarity:

- Choose parameter $r \in \mathbb{Z}^+$.
- Let $c_1, \ldots, c_r : \{0,1\}^d \rightarrow [0,1]$ be independent random MinHash's.
- Let $g : [0,1]^r \rightarrow \{1, \ldots, m\}$ 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 $r \in \mathbb{Z}^+$. (number of bands)
- Let $c_1, \ldots, c_r : \{0,1\}^d \rightarrow [0,1]$ be random MinHash.
- Let $g : [0,1]^r \rightarrow \{1, \ldots, m\}$ 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 $\Pr[h(\mathbf{q}) == h(\mathbf{y})] =??$

Reducing False Positives

Tunable LSH for Jaccard similarity:

- Choose parameter $r \in \mathbb{Z}^+$. (number of bands)
- 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 $\Pr[h(\mathbf{q}) == h(\mathbf{y})] = ??$

$$\Pr[h(\mathbf{q}) == h(\mathbf{y})] = \Pr[g(c_1(\mathbf{q}), \ldots, c_r(\mathbf{q})) = g(c_1(\mathbf{y}), \ldots, c_r(\mathbf{y}))] + \text{correction}$$

Reducing False Positives

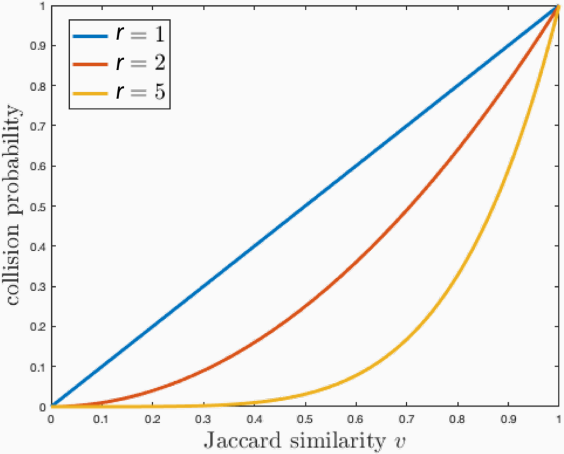
Tunable LSH for Jaccard similarity:

- Choose parameter $r \in \mathbb{Z}^+$. (number of bands)
- 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 $\Pr[h(\mathbf{q}) == h(\mathbf{y})] =??$

$$\Pr[h(\mathbf{q}) == h(\mathbf{y})] = \Pr[g(c_1(\mathbf{q}), \ldots, c_r(\mathbf{q})) = g(c_1(\mathbf{y}), \ldots, c_r(\mathbf{y}))] + \text{correction}$$
$$= v \cdot v \cdots v + \text{correction} = v^r + (1 - v^r)\frac{1}{m}$$

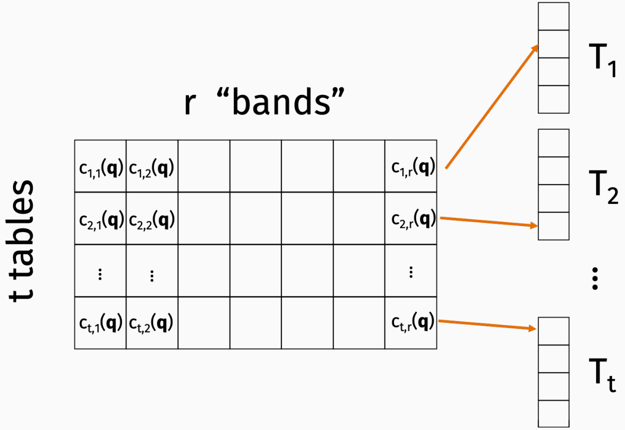
Tunable LSH



$r = 1$
$r = 2$
$r = 5$
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

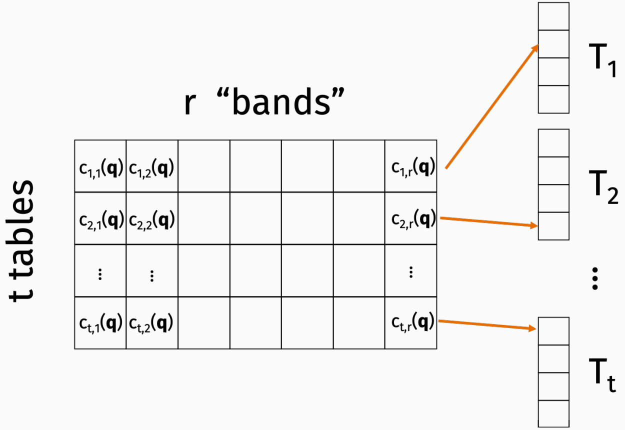
Tunable LSH

Full LSH scheme has two parameters to tune:



Tunable LSH

Full LSH scheme has two parameters to tune:



Still unclear why this is useful

Tunable LSH

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

False Negatives

False Positives

Tunable LSH

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

False Negatives	False Positives
Decrease	Increase

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

False Negatives	False Positives

Tunable LSH

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

False Negatives	False Positives
Decrease	Increase

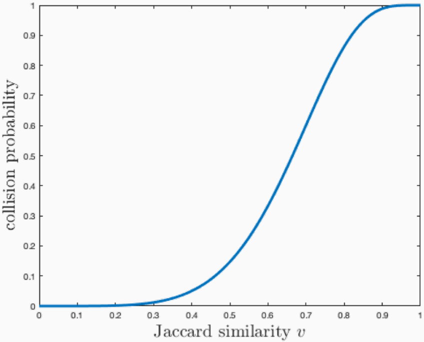
Effect of **increasing number of bands** r on:

False Negatives	False Positives
Increase	Decrease

s-curve tuning

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

$$\text{Collison Probability} \approx 1 - (1 - v^r)^t$$

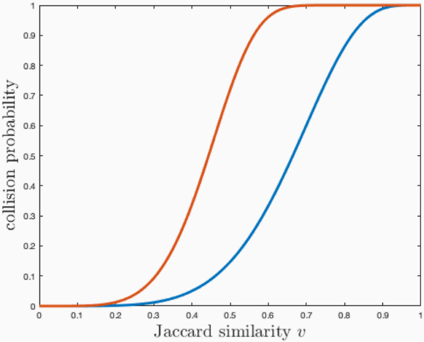


$r = 5, t = 5$

s-curve tuning

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

$$\text{Collison Probability} \approx 1 - (1 - v^r)^t$$

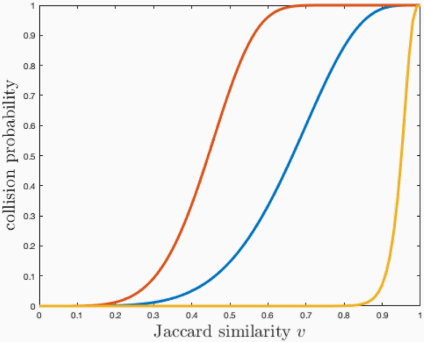


$r = 5, t = 40$

s-curve tuning

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

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

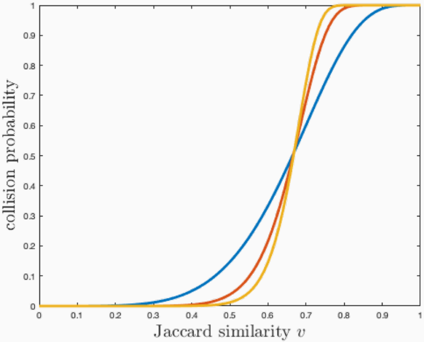


$r = 40, t = 5$

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, approaching a step function.

Fixed Threshold

Use Case 1: Fixed threshold.

- Shazam wants to find match to audio clip $\mathbf{y}$ in a database of 10 million clips.

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 $J(\mathbf{y}, \mathbf{q}) > .9$.

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 $J(\mathbf{y}, \mathbf{q}) \in [.7, .9]$.

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 $\gtrsim 1 - (1 - .9^{25})^{40} = .95$

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 $\lesssim 1 - (1 - .9^{25})^{40} = .95$

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.

LSH Based Nearest-Neighbor in Theory

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: $O\left(n^{1/C}\right)$.*

LSH Based Nearest-Neighbor in Theory

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 $\mathbf{y}$.

LSH Based Nearest-Neighbor in Theory

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 $\mathbf{y}$.

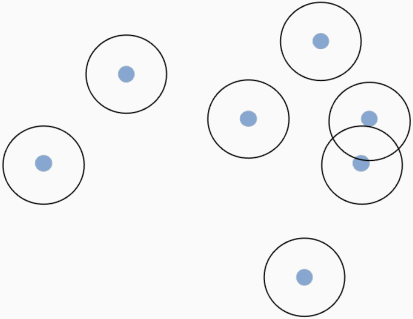
If there is no point at distance R, algorithm does not need to return anything.

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.

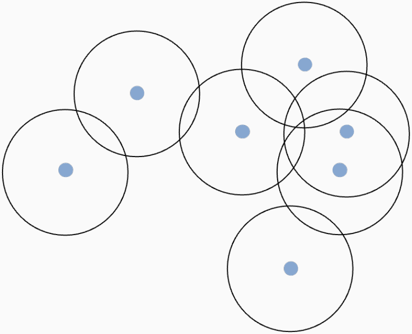


LSH Based Nearest-Neighbor in Theory

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

$$R \qquad 2R \qquad 4R \qquad 8R \qquad \dots$$

Search from most accurate level to least accurate.

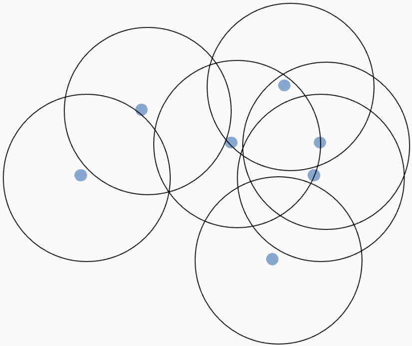


LSH Based Nearest-Neighbor in Theory

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.



Approximate Nearest Neighbor Search

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:

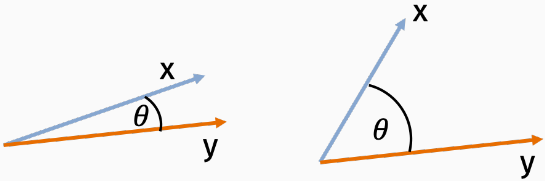
- *Time:* $\tilde{O}\left(n^{1/C}\right)$.
- *Space:* $\tilde{O}\left(n^{1+1/C} + nd\right)$.

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

Other LSH Functions

Good locality sensitive hash functions exists for other similarity measures.

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



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

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).

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).

$$\cos\left(\theta(\mathbf{x}, \mathbf{y})\right) = \langle \mathbf{x}, \mathbf{y} \rangle$$

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).

$$\begin{aligned}\cos(\theta(\mathbf{x}, \mathbf{y})) &= \langle \mathbf{x}, \mathbf{y} \rangle \\ &= 1 - \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|_2^2\end{aligned}$$

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).

$$\begin{aligned}\cos(\theta(\mathbf{x}, \mathbf{y})) &= \langle \mathbf{x}, \mathbf{y} \rangle \\ &= 1 - \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|_2^2\end{aligned}$$

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).

$$\begin{aligned} \cos\left(\theta(\mathbf{x},\mathbf{y})\right) &= \langle \mathbf{x},\mathbf{y}\rangle \\ &= 1 - \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 \end{aligned}$$

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

SimHash

Locality sensitive hash for **cosine similarity**:

- Let $\mathbf{g} \in \mathbb{R}^d$ be randomly chosen with each entry $\mathcal{N}(0,1)$.

SimHash

Locality sensitive hash for **cosine similarity**:

- Let $\mathbf{g} \in \mathbb{R}^d$ be randomly chosen with each entry $\mathcal{N}(0, 1)$.
- Let $f : \{-1, 1\} \rightarrow \{1, \ldots, m\}$ be a uniformly random hash function.

SimHash

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 defined $h(\mathbf{x}) = f(\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle))$.

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

SimHash

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 defined $h(\mathbf{x}) = f(\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle))$.

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

$$\textbf{Claim: } \Pr[h(\mathbf{x}) == h(\mathbf{y})] = 1 - \frac{\theta}{\pi} + \frac{\theta}{\pi m}$$

SimHash

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 defined $h(\mathbf{x}) = f\left(\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle)\right)$.

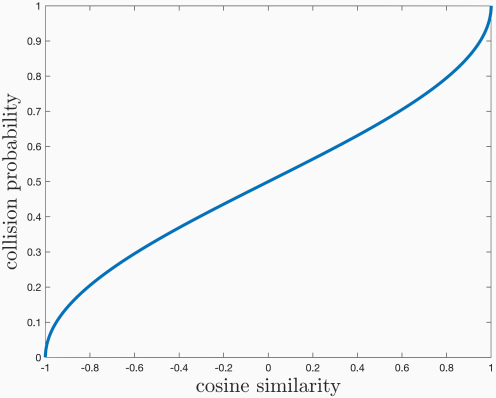
If $\cos(\theta(\mathbf{x}, \mathbf{y})) = v$, what is $\Pr[h(\mathbf{x}) == h(\mathbf{y})]$?

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

SimHash Analysis in 2D

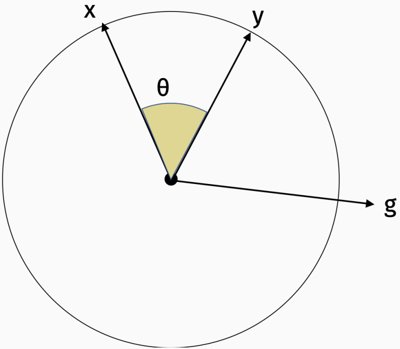
Theorem (to be proven): If $\cos(\theta(\mathbf{x}, \mathbf{y})) = v$, then

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



SimHash Analysis in 2D

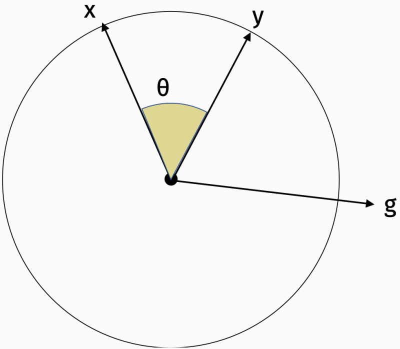
To prove: $\Pr[h(\mathbf{x}) == h(\mathbf{y})] \approx 1 - \frac{\theta}{\pi}$, where $h(\mathbf{x}) = f(\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle))$ and f is uniformly random hash function.



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

SimHash Analysis in 2D

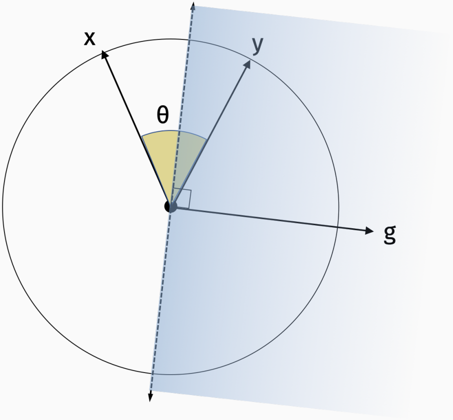
To prove: $\Pr[h(\mathbf{x}) == h(\mathbf{y})] \approx 1 - \frac{\theta}{\pi}$, where $h(\mathbf{x}) = f(\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle))$ and f is uniformly random hash function.



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

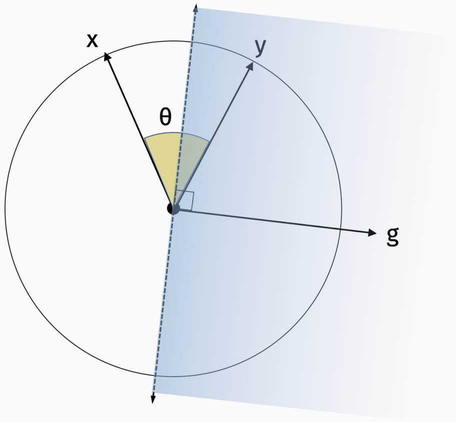
$$\text{where } z = \Pr[\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle) == \text{sign}(\langle \mathbf{g}, \mathbf{y} \rangle)]$$

SimHash Analysis 2D



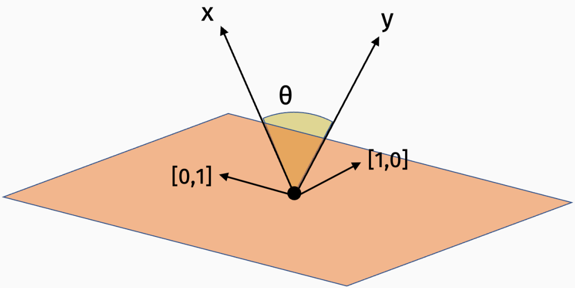
- $\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle) \neq \text{sign}(\langle \mathbf{g}, \mathbf{y} \rangle)$ when $\mathbf{g}$ lands between $\mathbf{x}$ and $\mathbf{y}$.
- By rotational invariance of Gaussian
  $\Pr[\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle) \neq \text{sign}(\langle \mathbf{g}, \mathbf{y} \rangle)] = \frac{2\theta}{2\pi} = \frac{\theta}{\pi}$

SimHash Analysis 2D



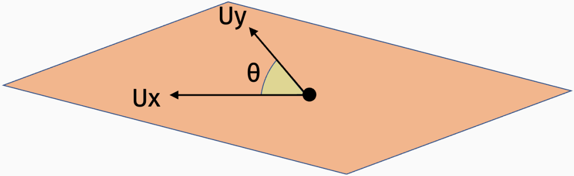
$$\Pr[\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle) == \text{sign}(\langle \mathbf{g}, \mathbf{y} \rangle)] = 1 - \frac{\theta}{\pi}$$

SimHash Analysis Higher Dimensions



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

SimHash Analysis Higher Dimensions



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

Note: We have now reduced to the 2D case (we can also revert this rotation at the end)

SimHash Analysis Higher Dimensions

Analysis: Let U be the rotation matrix that maps **x** and **y** to the XY plane

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

SimHash Analysis Higher Dimensions

Analysis: Let U be the rotation matrix that maps $\mathbf{x}$ and $\mathbf{y}$ to the XY plane

$$\begin{aligned}&\Pr[\text{sign}(\langle \mathbf{g}, \mathbf{x}\rangle) == \text{sign}(\langle \mathbf{g}, \mathbf{y}\rangle)]\\ &= \Pr[\text{sign}(\langle \mathbf{U}^T\mathbf{g}, \mathbf{x}\rangle) == \text{sign}(\langle \mathbf{U}^T\mathbf{g}, \mathbf{y}\rangle)]\end{aligned}$$

SimHash Analysis Higher Dimensions

Analysis: Let U be the rotation matrix that maps **x** and **y** to the XY plane

$$\begin{aligned}
&\Pr[\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle) == \text{sign}(\langle \mathbf{g}, \mathbf{y} \rangle)] \\
&= \Pr[\text{sign}(\langle \mathbf{U}^T \mathbf{g}, \mathbf{x} \rangle) == \text{sign}(\langle \mathbf{U}^T \mathbf{g}, \mathbf{y} \rangle)] \\
&= \Pr[\text{sign}(\langle \mathbf{g}, \mathbf{Ux} \rangle) == \text{sign}(\langle \mathbf{g}, \mathbf{Uy} \rangle)]
\end{aligned}$$

SimHash Analysis Higher Dimensions

Analysis: Let U be the rotation matrix that maps $\mathbf{x}$ and $\mathbf{y}$ to the XY plane

$$\begin{aligned}
&\Pr[\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle) == \text{sign}(\langle \mathbf{g}, \mathbf{y} \rangle)] \\
&= \Pr[\text{sign}(\langle \mathbf{U}^T \mathbf{g}, \mathbf{x} \rangle) == \text{sign}(\langle \mathbf{U}^T \mathbf{g}, \mathbf{y} \rangle)] \\
&= \Pr[\text{sign}(\langle \mathbf{g}, \mathbf{U}\mathbf{x} \rangle) == \text{sign}(\langle \mathbf{g}, \mathbf{U}\mathbf{y} \rangle)] \\
&= \Pr[\text{sign}(\langle \mathbf{g}[1,2], (\mathbf{U}\mathbf{x})[1,2] \rangle) == \text{sign}(\langle \mathbf{g}[1,2], (\mathbf{U}\mathbf{y}[1,2] \rangle)]
\end{aligned}$$

SimHash Analysis Higher Dimensions

Analysis: Let U be the rotation matrix that maps $\mathbf{x}$ and $\mathbf{y}$ to the XY plane

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

Banded SimHash

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

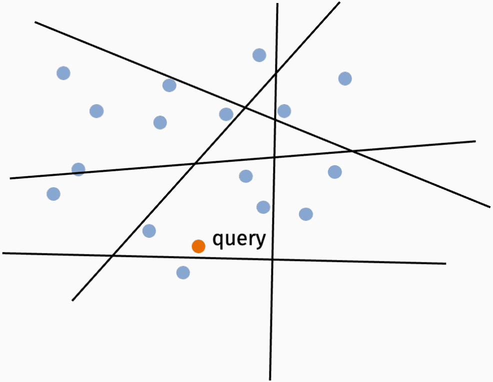
- Let $\mathbf{g}_1, \ldots, \mathbf{g}_r \in \mathbb{R}^d$ be randomly chosen with each entry $\mathcal{N}(0, 1)$.
- Let $f : \{-1, 1\}^r \to \{1, \ldots, m\}$ be a uniformly random hash function.
- $h : \mathbb{R}^d \to \{1, \ldots, m\}$ is defined $h(\mathbf{x}) = f\left([\text{sign}(\langle \mathbf{g}_1,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

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

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.

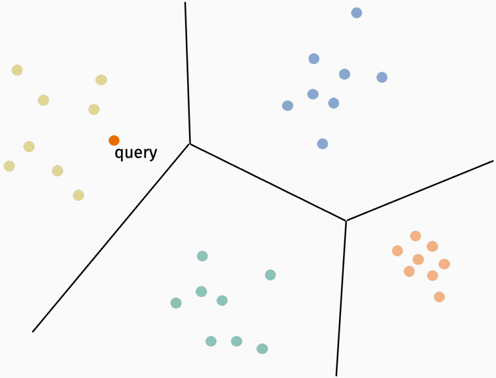
Starting point: Think of LSH as a randomized space-partitioning method.



Nearest-Neighbor Search in Practice

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

Common approach: Split data using k-means clustering.



Open Theory Challenge

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 $\rho \leq 7/(8c^2$ …
over the result by Andoni and Indyk (FC …
a locality–sensitive hashing lower bound …
a standard reduction we obtain a data s…
$\rho \leq 7/(8c) + O(1/c^{3/2}) + o_c(1)$, 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