CS-GY 6763: Lecture 4
Near neighbor search in high dimensions + locality sensitive hashing

NYU Tandon School of Engineering, Prof. Christopher Musco

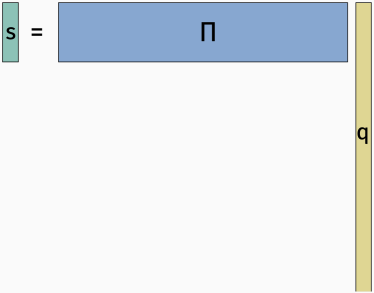
Given two length d vectors $\mathbf{y}$ and $\mathbf{q}$, construct compact representations (sketches) $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{q}}$ such that $\text{dist}(\mathbf{y}, \mathbf{q})$ can be estimated accurately from $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{q}}$.

Each of $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{q}}$ should require $k \ll d$ space.

Lemma (Johnson-Lindenstrauss, 1984)

For any two data points $\mathbf{y}, \mathbf{q} \in \mathbb{R}^d$ *there exists a* <u>*linear map*</u> $\mathbf{\Pi} : \mathbb{R}^d \rightarrow \mathbb{R}^k$ *where* $k = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ *such that with probability* $1 - \delta$,

$$(1-\epsilon)\|\mathbf{q}-\mathbf{y}\|_2 \leq \|\mathbf{\Pi}\mathbf{q}-\mathbf{\Pi}\mathbf{y}\|_2 \leq (1+\epsilon)\|\mathbf{q}-\mathbf{y}\|_2.$$



Lemma (Johnson-Lindenstrauss, 1984)

For any set of n data points $\mathbf{q}_1, \ldots, \mathbf{q}_n \in \mathbb{R}^d$ *there exists a* *linear map* $\Pi : \mathbb{R}^d \to \mathbb{R}^k$ *where* $k = O\left(\frac{\log(n/\delta)}{\epsilon^2}\right)$ *such that with probability* $(1-\delta)$, *for all i, j,*

$$(1-\epsilon)\|\mathbf{q}_i - \mathbf{q}_j\|_2 \leq \|\mathbf{\Pi}\mathbf{q}_i - \mathbf{\Pi}\mathbf{q}_j\|_2 \leq (1+\epsilon)\|\mathbf{q}_i - \mathbf{q}_j\|_2.$$

Extends to approximating all pairwise distances in a set of n vectors via a union bound.

Another distance measure (actually a similarity measure) between binary vectors in $\{0,1\}^d$:

Definition (Jaccard Similarity)

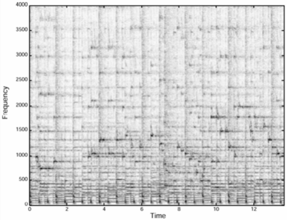
$$J(\mathbf{q},\mathbf{y}) = \frac{|\mathbf{q}\cap\mathbf{y}|}{|\mathbf{q}\cup\mathbf{y}|} = \frac{\text{\# of non-zero entries in common}}{\text{total \# of non-zero entries}}$$

Natural similarity measure for binary vectors. $0 \leq J(\mathbf{q},\mathbf{y}) \leq 1$.

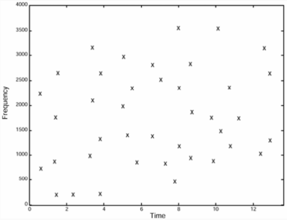
Can be applied to any data which has a natural binary representation (more than you might think).

$$\mathbf{y} = \begin{bmatrix}1 & 0 & 1 & 1 & 0 & 0\end{bmatrix}$$
$$\mathbf{q} = \begin{bmatrix}1 & 1 & 0 & 1 & 0 & 0\end{bmatrix}$$

How does **Shazam** match a song clip against a library of 8 million songs (32 TB of data) in a fraction of a second?



Spectrogram extracted from audio clip.

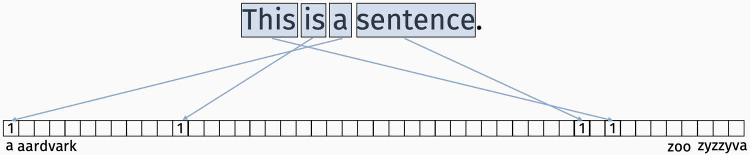


Processed spectrogram: used to construct audio "fingerprint" $\mathbf{q} \in \{0,1\}^d$.

Each clip is represented by a high dimensional binary vector $\mathbf{q}$.

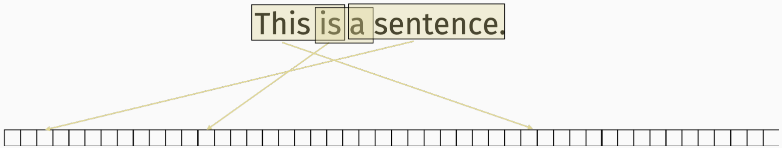
1	0	1	1	0	0	0	1	0	0	0	0	1	1	0	1

"Bag-of-words" model:



How many words do a pair of documents have in common?

"Bag-of-words" model:



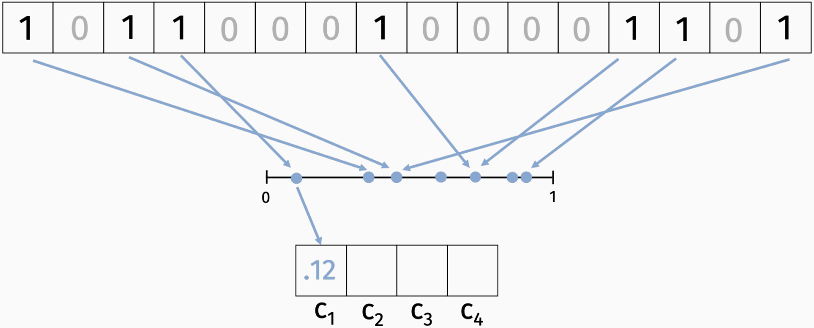
How many bigrams do a pair of documents have in common?

APPLICATIONS: DOCUMENT SIMILARITY

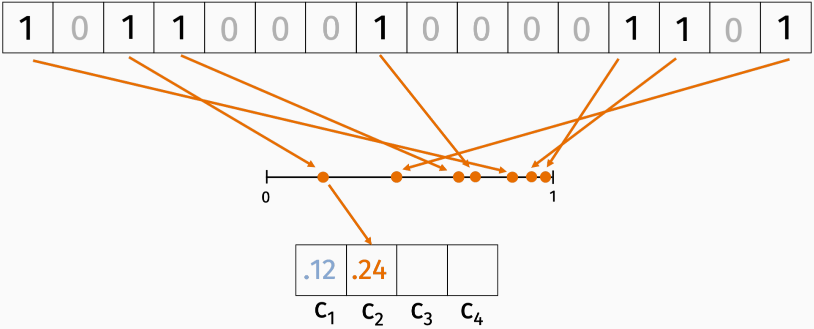
- Finding duplicate or new duplicate documents or webpages.
- Change detection for high-speed web caches.
- Finding near-duplicate emails or customer reviews which could indicate spam.

MinHash (Broder, '97):

- Choose k random hash functions $h_1, \ldots, h_k : \{1, \ldots, n\} \rightarrow [0, 1]$.
- For $i \in 1, \ldots, k$,
  - Let $c_i = \min_{j, \mathbf{q}_j = 1} h_i(j)$.
- $C(\mathbf{q}) = [c_1, \ldots, c_k]$.

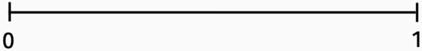


- Choose k random hash functions $h_1, \ldots, h_k : \{1, \ldots, n\} \to [0, 1]$.
- For $i \in 1, \ldots, k$,
  - Let $c_i = \min_{j, \mathbf{q}_j = 1} h_i(j)$.
- $C(\mathbf{q}) = [c_1, \ldots, c_k]$.



Claim: $\Pr[c_i(\mathbf{q}) = c_i(\mathbf{y})] = J(\mathbf{q}, \mathbf{y})$.

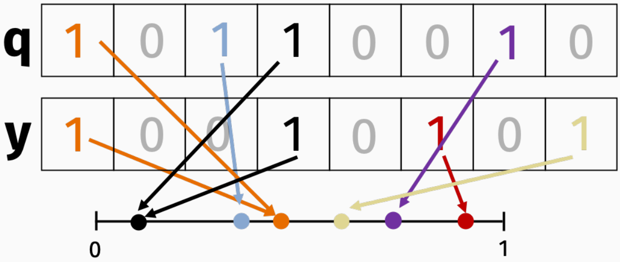
q	1	0	1	1	0	0	1	0
y	1	0	0	1	0	1	0	1



Proof:

1. For $c_i(\mathbf{q}) = c_i(\mathbf{y})$, we need that $\arg\min_{i \in \mathbf{q}} h(i) = \arg\min_{i \in \mathbf{y}} h(i)$.

Claim: $\Pr[c_i(\mathbf{q}) = c_i(\mathbf{y})] = J(\mathbf{q}, \mathbf{y})$.



2. Every non-zero index in $\mathbf{q} \cup \mathbf{y}$ is equally likely to produce the lowest hash value. $c_i(\mathbf{q}) = c_i(\mathbf{y})$ only if this index is 1 in <u>both</u> $\mathbf{q}$ and $\mathbf{y}$. There are $\mathbf{q} \cap \mathbf{y}$ such indices. So:

$$\Pr[c_i(\mathbf{q}) = c_i(\mathbf{y})] = \frac{\mathbf{q} \cap \mathbf{y}}{\mathbf{q} \cup \mathbf{y}} = J(\mathbf{q}, \mathbf{y})$$

Let $J = J(\mathbf{q}, \mathbf{y})$ denote the Jaccard similarity between **q** and **y**.

Return: $\tilde{J} = \frac{1}{k}\sum_{i=1}^{k} \mathbb{1}[c_i(\mathbf{q}) = c_i(\mathbf{y})]$.

Unbiased estimate for Jaccard similarity:

$$\mathbb{E}\tilde{J} =$$

C(q)

.12	.24	.76	.35

C(y)

.12	.98	.76	.11

The more repetitions, the lower the variance.

Let $J = J(\mathbf{q}, \mathbf{y})$ denote the true Jaccard similarity.

Estimator: $\tilde{J} = \frac{1}{k}\sum_{i=1}^{k} \mathbb{1}[c_i(\mathbf{q}) = c_i(\mathbf{y})]$.

$$\text{Var}[\tilde{J}] =$$

Plug into Chebyshev inequality. How large does k need to be so that with probability $> 1 - \delta$:

$$|J - \tilde{J}| \leq \epsilon?$$

Chebyshev inequality: As long as $k = O\left(\frac{1}{\epsilon^2 \delta}\right)$, then with prob. $1 - \delta$,

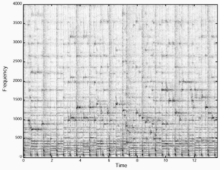
$$J(\mathbf{q}, \mathbf{y}) - \epsilon \leq \tilde{J}(C(\mathbf{q}), C(\mathbf{y})) \leq J(\mathbf{q}, \mathbf{y}) + \epsilon.$$

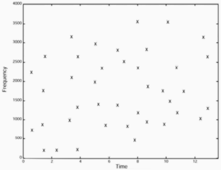
And $\tilde{J}$ only takes $O(k)$ time to compute! Independent of original fingerprint dimension d.

Can be improved to $\log(1/\delta)$ dependence. Can anyone tell me how?

SIMILARITY SKETCHING

input data





high dimensional vector representation

1	0	1	1	0	0	0	1	0	0	0	0	1	1	0	1

.45	.68	.10	.92

sketched representation

BREAK

Common goal: Find all vectors in database $\mathbf{q}_1, \ldots, \mathbf{q}_n \in \mathbb{R}^d$ that are close to some input query vector $\mathbf{y} \in \mathbb{R}^d$. I.e. find all of $\mathbf{y}$'s "nearest neighbors" in the database.

- The Shazam problem.
- Audio + video search.
- Finding duplicate or near duplicate documents.
- Detecting seismic events.

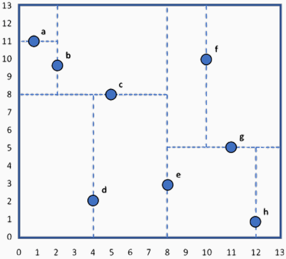
How does similarity sketching help in these applications?

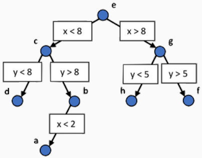
- Improves runtime of "linear scan" from $O(nd)$ to $O(nk)$.
- Improves space complexity from $O(nd)$ to $O(nk)$. This can be super important – e.g. if it means the linear scan only accesses vectors in fast memory.

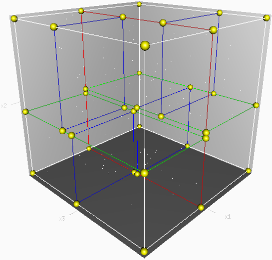
New goal: Sublinear $o(n)$ time to find near neighbors.

BEYOND A LINEAR SCAN

This problem can already be solved for a small number of dimensions using space partitioning approaches (e.g. kd-tree).







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

Only been attacked much more recently:

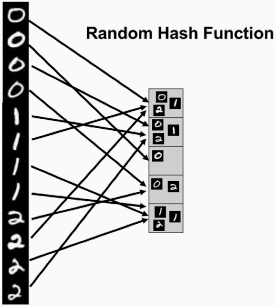
- Locality-sensitive hashing [Indyk, Motwani, 1998]
- Spectral hashing [Weiss, Torralba, and Fergus, 2008]
- Vector quantization [Jégou, Douze, Schmid, 2009]

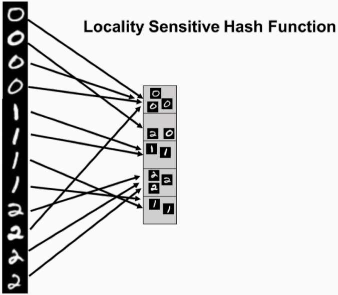
Key Insight: Trade worse space-complexity for better time-complexity.

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:

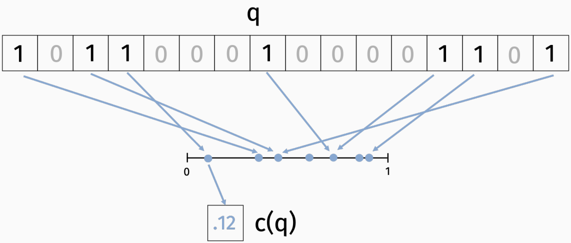
- Higher when $\mathbf{q}$ and $\mathbf{y}$ are more similar, i.e. $s(\mathbf{q}, \mathbf{y})$ is higher.
- Lower when $\mathbf{q}$ and $\mathbf{y}$ are more dissimilar, i.e. $s(\mathbf{q}, \mathbf{y})$ is lower.





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 $h(\mathbf{q}) = g(c(\mathbf{q}))$.



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

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

Basic approach for near neighbor search in a database.

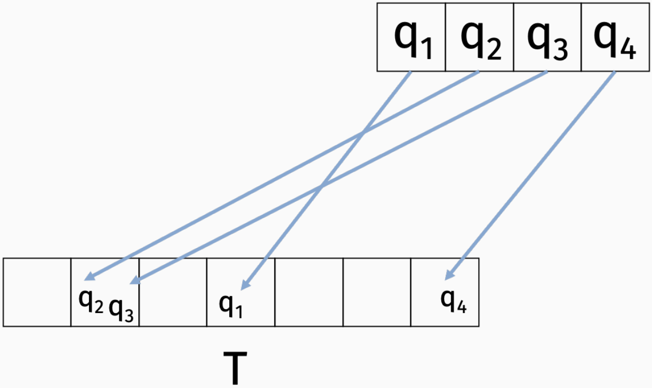
Pre-processing:

- 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.


q_1 q_2 q_3 q_4
q_2 q_3
q_1
q_4
T

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 $\mathbf{y}$?

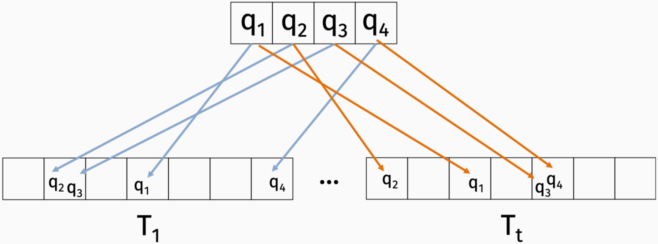
A higher false negative rate means we miss near neighbors.

A higher false positive rate means increased runtime – we need to compute $J(\mathbf{q}, \mathbf{y})$ 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 $< .2$.

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

What's the probability we find q?



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.
- For $i = 1, \ldots, n$, $j = 1, \ldots, t$,
  - Insert $\mathbf{q}_i$ into $T_j(h_j(\mathbf{q}_i))$.

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?

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

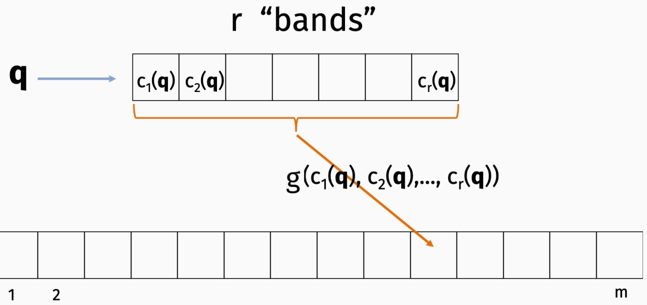
What is the probability we will need to compute $J(\mathbf{z}, \mathbf{y})$ in our hashing scheme with one table? I.e. the probability that $\mathbf{y}$ hashes into at least one bucket containing $\mathbf{z}$.

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

Change our locality sensitive hash function.

Tunable LSH for Jaccard similarity:

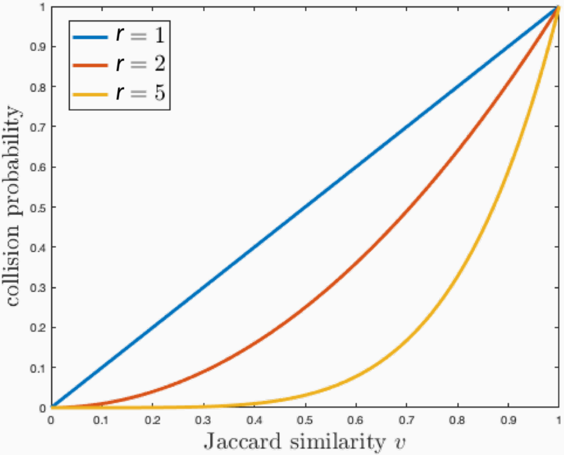
- Choose parameter $r \in \mathbb{Z}^+$.
- Let $c_1, \ldots, c_r : \{0,1\}^d \to [0,1]$ be random MinHash.
- Let $g : [0,1]^r \to \{1, \ldots, m\}$ be a uniform random hash function.
- Let $h(\mathbf{x}) = g(c_1(\mathbf{x}), \ldots, c_r(\mathbf{x}))$.



Tunable LSH for Jaccard similarity:

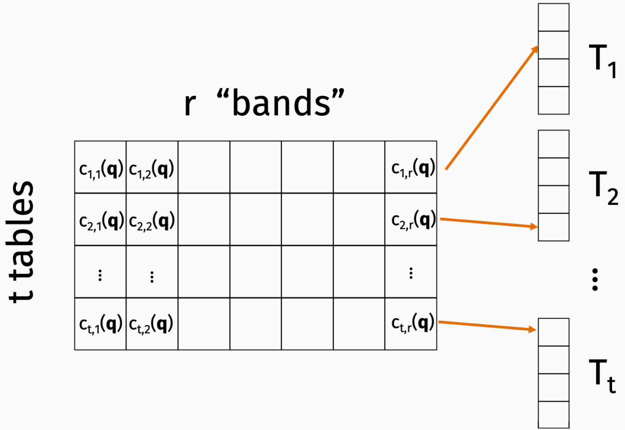
- Choose parameter $r \in \mathbb{Z}^+$.
- Let $c_1, \ldots, c_r : \{0,1\}^d \rightarrow [0,1]$ be random MinHash.
- Let $g : [0,1]^r \rightarrow \{1, \ldots, m\}$ be a uniform random hash function.
- Let $h(\mathbf{x}) = g(c_1(\mathbf{x}), \ldots, c_r(\mathbf{x}))$.

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



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

Choose tables t large enough so false negative rate to 1%.

Parameter: $r = 1$.

Chance we find $\mathbf{q}$ with $J(\mathbf{y}, \mathbf{q}) = .8$:

Chance we need to check $\mathbf{z}$ with $J(\mathbf{y}, \mathbf{z}) = .4$:

Choose tables t large enough so false negative rate to 1%.

Parameter: $r = 2$.

Chance we find $\mathbf{q}$ with $J(\mathbf{y}, \mathbf{q}) = .8$:

Chance we need to check $\mathbf{z}$ with $J(\mathbf{y}, \mathbf{z}) = .4$:

Choose tables t large enough so false negative rate to 1%.

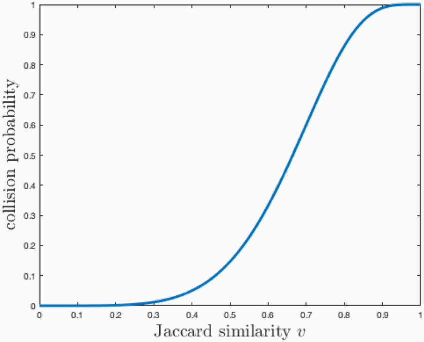
Parameter: $r = 5$.

Chance we find $\mathbf{q}$ with $J(\mathbf{y}, \mathbf{q}) = .8$:

Chance we need to check $\mathbf{z}$ with $J(\mathbf{y}, \mathbf{z}) = .4$:

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

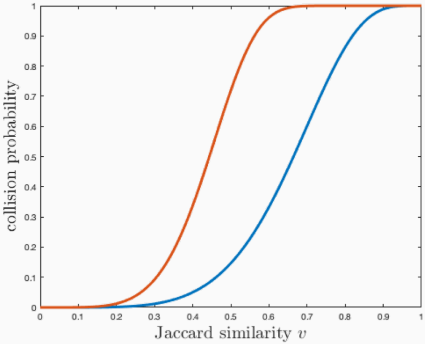
$$\approx 1 - (1 - v^r)^t$$



$$r = 5, t = 5$$

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

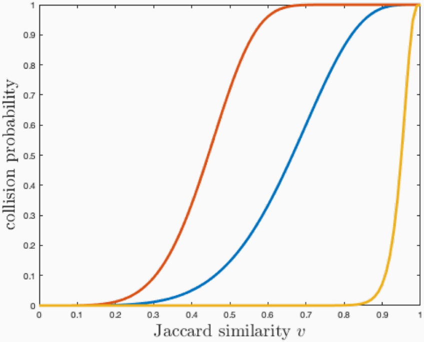
$$\approx 1 - (1 - v^r)^t$$



$$r = 5, t = 40$$

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

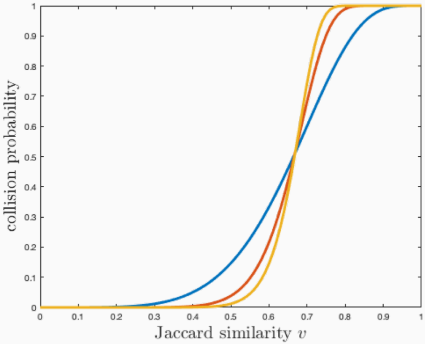
$$\approx 1 - (1 - v^r)^t$$



$r = 40, t = 5$

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.

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 <u>true matches</u> with $J(\mathbf{y}, \mathbf{q}) > .9$.
- There are 10,000 <u>near matches</u> 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:

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

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

Directly trade space for fast search.

Near Neighbor Search Problem

Concrete worst case result:

Theorem (Indyk, Motwani, 1998)

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

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

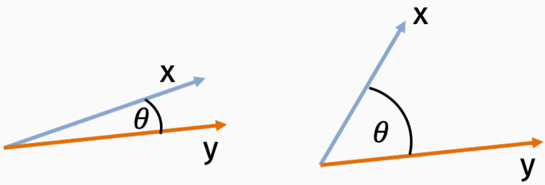
Theorem (Indyk, Motwani, 1998)

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

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 is natural "inverse" for Euclidean distance.

Euclidean distance $\|\mathbf{x} - \mathbf{y}\|_2^2$:

- Suppose for simplicity that $\|\mathbf{x}\|_2^2 = \|\mathbf{y}\|_2^2 = 1$.

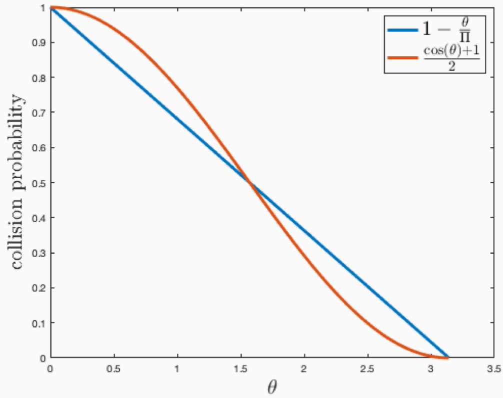
Locality sensitive hash for **cosine similarity**:

- 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.
- $h : \mathbb{R}^d \rightarrow \{1, \ldots, m\}$ is definied $h(\mathbf{x}) = f(\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle))$.

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

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

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



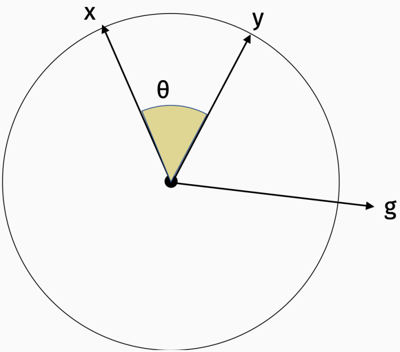
SimHash can be tuned, just like our MinHash based LSH function for Jaccard similarity:

- 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 $h(\mathbf{x}) = f([\text{sign}(\langle \mathbf{g}_1, \mathbf{x} \rangle), \ldots, \text{sign}(\langle \mathbf{g}_r, \mathbf{x} \rangle)])$.

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

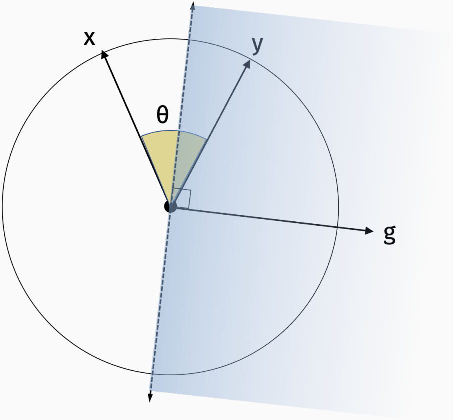
To prove:

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



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

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



$\Pr[h(\mathbf{x}) == h(\mathbf{y})] \approx$ probability $\mathbf{x}$ and $\mathbf{y}$ are on the same side of hyperplane orthogonal to $\mathbf{g}$.