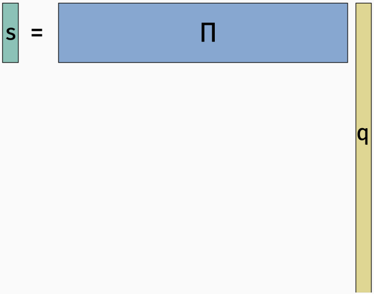
CS-GY 9223 D: Lecture 4
Near neighbor search + locality sensitive hashing

NYU Tandon School of Engineering, Prof. Christopher Musco

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 $\mathbf{\Pi} : \mathbb{R}^d \to \mathbb{R}^k$ where $k = O\left(\frac{\log n}{\epsilon^2}\right)$ such that for all i, j,

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



$\mathbf{\Pi} \in \mathbb{R}^{k\times d}$ be chosen so that each entry equals $\frac{1}{\sqrt{k}}\mathcal{N}(0,1)$.

… or each entry equals $\frac{1}{\sqrt{k}} \pm 1$ with equal probability.

-2.1384	2.9080	-0.3538	0.0229	0.5201	-0.2938	-1.3320	-1.3617	-0.1952
-0.8396	0.8252	-0.8236	-0.2620	-0.0200	-0.8479	-2.3299	0.4550	-0.2176
1.3546	1.3790	-1.5771	-1.7502	-0.0348	-1.1201	-1.4491	-0.8487	-0.3031
-1.0722	-1.0582	0.5080	-0.2857	-0.7982	2.5260	0.3335	-0.3349	0.0230
0.9610	-0.4686	0.2820	-0.8314	1.0187	1.6555	0.3914	0.5528	0.0513
0.1240	-0.2725	0.0335	-0.9792	-0.1332	0.3075	0.4517	1.0391	0.8261
1.4367	1.0984	-1.3337	-1.1564	-0.7145	-1.2571	-0.1303	-1.1176	1.5270
-1.9609	-0.2779	1.1275	-0.5336	1.3514	-0.8655	0.1837	1.2607	0.4669
-0.1977	0.7015	0.3502	-2.0026	-0.2248	-0.1765	-0.4762	0.6601	-0.2097
-1.2078	-2.0518	-0.2991	0.9642	-0.5890	0.7914	0.8620	-0.0679	0.6252

```
>> Pi = randn(m,d);
>> s = (1/sqrt(m))*Pi*q;
```

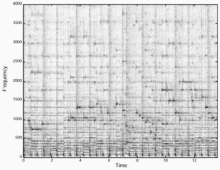
1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	1	-1	-1	1	1	-1
1	1	1	-1	1	-1	-1	-1	1	1	1	1	-1	1	-1	-1	-1
1	1	-1	-1	-1	1	-1	-1	1	1	-1	1	-1	1	-1	1	-1
-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	-1	1
1	-1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	-1	1	1	1
1	1	-1	1	1	-1	1	-1	1	-1	1	-1	1	1	1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	1	1	-1	-1	1	-1	-1	1	1
-1	-1	1	1	1	1	-1	-1	1	-1	1	1	1	-1	1	-1	1
-1	1	-1	1	-1	1	1	-1	-1	1	-1	1	-1	-1	1	-1	1

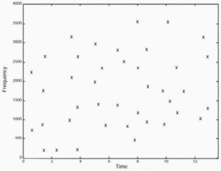
```
>> Pi = 2*randi(2,m,d)-3;
>> s = (1/sqrt(m))*Pi*q;
```

Often called "random projections".

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

Definition (Jaccard Similarity)

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

$0 \leq J(\mathbf{q},\mathbf{y}) \leq 1.$

Similar result to JL: Given a MinHash sketch with $k = O\left(\frac{\log n}{\epsilon^2}\right)$ dimensions, we can estimate the Jaccard similarity between all pairs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with high probability.

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.

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

How does similarity sketching help in these applications?

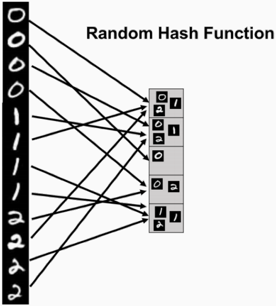
- Improves runtime of "linear scan" from $O(nd)$ to $O(nk)$.
- Improves space complexity from $O(nd)$ to $O(nk)$. This can be super important – e.g. if it means the linear scan only accesses vectors in fast memory.

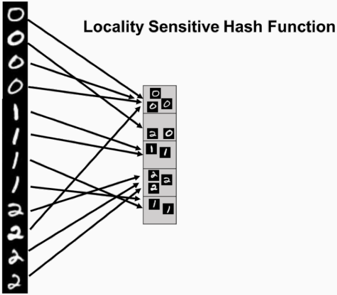
New goal: Sublinear $o(n)$ time to find near neighbors.

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

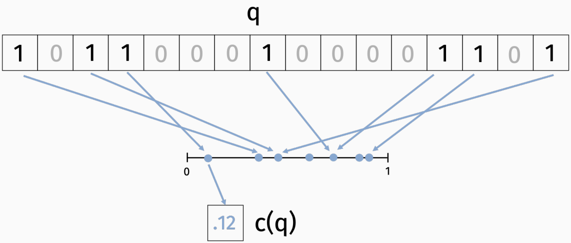
- Higher when $\mathbf{q}$ and $\mathbf{y}$ are more similar, i.e. $s(\mathbf{q}, \mathbf{y})$ is higher.
- Lower when $\mathbf{q}$ and $\mathbf{y}$ are more dissimilar, i.e. $s(\mathbf{q}, \mathbf{y})$ is lower.





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

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



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 fully random hash function.
- Let $h(\mathbf{x}) = g(c(\mathbf{x}))$.

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

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

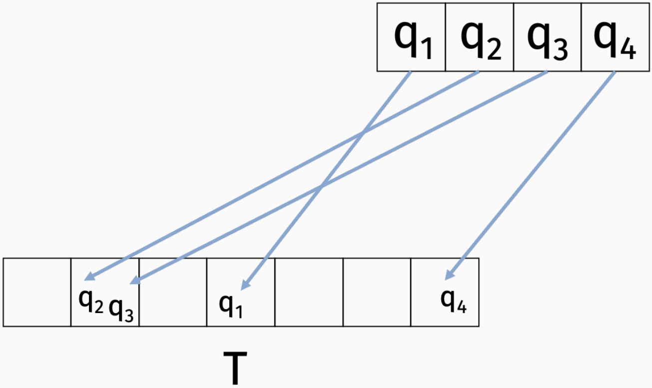
Basic approach for near neighbor search in a database.

Pre-processing:

- Select random LSH function $h : \{0,1\}^d \to 1, \ldots, m$.
- Create table T with $m = O(n)$ slots.
- 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 $O(d \cdot |T(h(\mathbf{y})|)$.


q_1 q_2 q_3 q_4
q_2 q_3
q_1
q_4
T

Two main considerations:

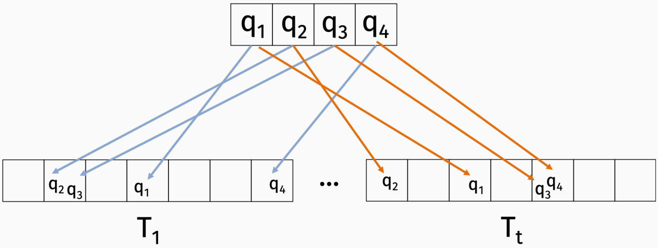
- False Negative Rate: What's the probability we do not find a vector that is close to $\mathbf{y}$?
- False Positive Rate: What's the probability that a vector in $T(h(\mathbf{y}))$ is not close to $\mathbf{y}$?

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 $\mathbf{y}$.

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

What's the probability we do not 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 $T_1(h_1(\mathbf{y})) \cup T_2(h_2(\mathbf{y})) \cup \ldots, T_t(h_t(\mathbf{y}))$.

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?

WHAT HAPPENS TO FALSE POSITIVES?

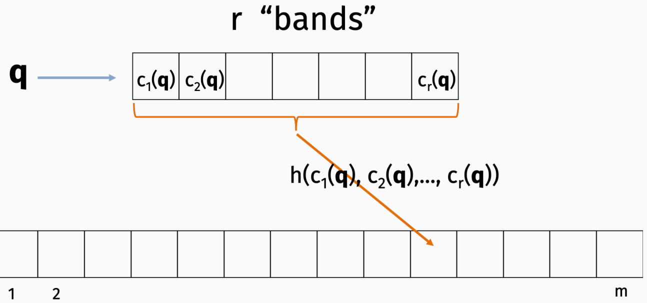
Suppose there is some other database point $\mathbf{z}$ with $J(\mathbf{y}, \mathbf{z}) = .2$. What is the probability we will need to compute $J(\mathbf{z}, \mathbf{y})$ in our hashing scheme with one table?

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 $c_1, \ldots, c_s : \{0,1\}^d \to [0,1]$ be random MinHash.
- Let $g : [0,1]^s \to \{1, \ldots, m\}$ be a fully random hash function.
- Let $h(\mathbf{x}) = g(c_1(\mathbf{x}), \ldots, c_r(\mathbf{x}))$.

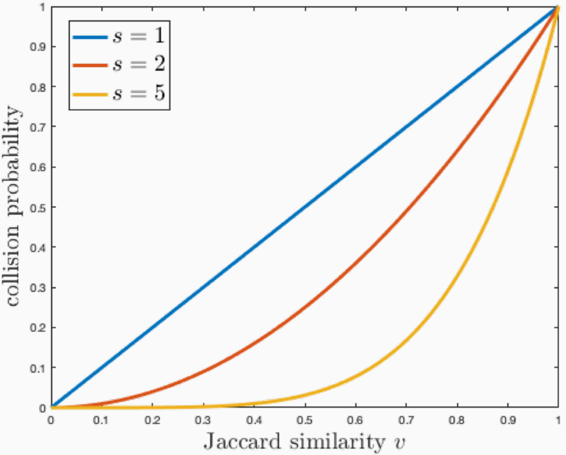


REDUCING FALSE POSITIVES

Tunable LSH for Jaccard similarity:

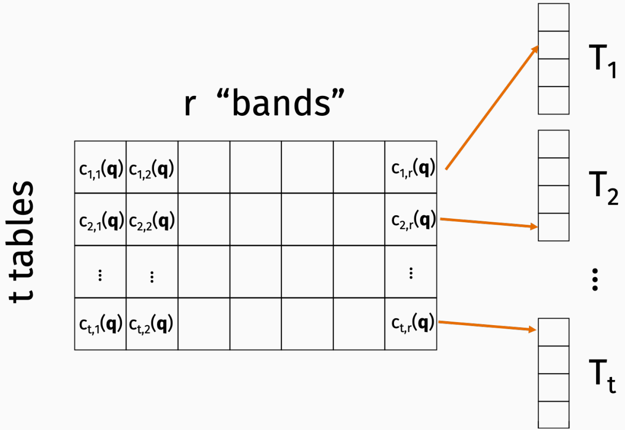
- Choose parameter $r \in \mathbb{Z}^+$.
- Let $c_1, \ldots, c_s : \{0,1\}^d \to [0,1]$ be random MinHash.
- Let $g : [0,1]^s \to \{1, \ldots, m\}$ be a fully random hash function.
- Let $h(\mathbf{x}) = g(c_1(\mathbf{x}), \ldots, c_r(\mathbf{x}))$.

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



$s = 1$
$s = 2$
$s = 5$
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%.

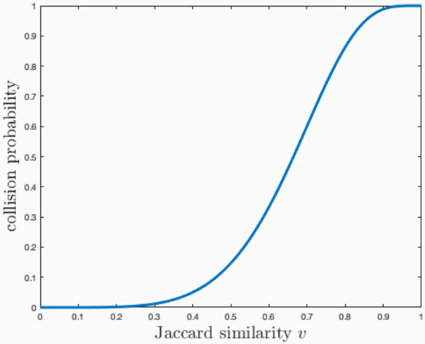
Parameter: $r = 5$.

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

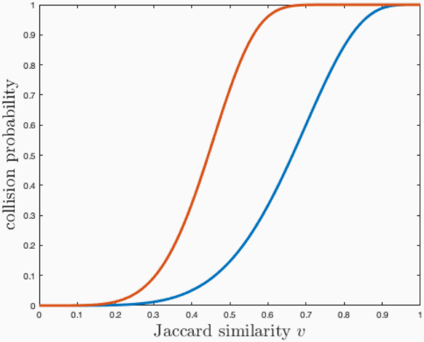
$$\approx 1 - (1 - v^r)^t$$



$r = 5, t = 5$

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

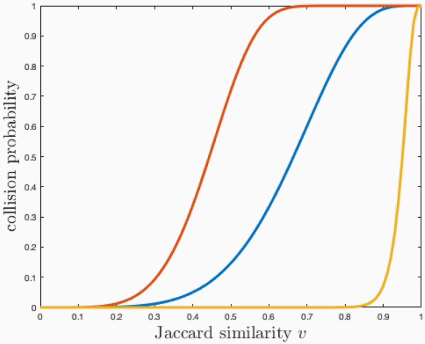
$$\approx 1 - (1 - v^r)^t$$



$$r = 5, t = 40$$

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

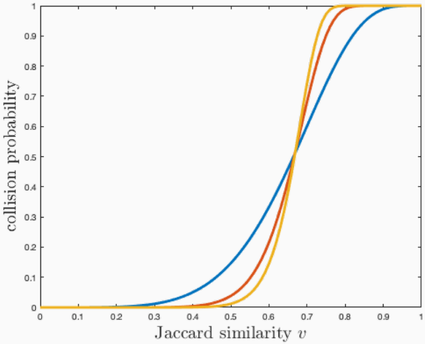
$$\approx 1 - (1 - v^r)^t$$



$$r = 40, t = 5$$

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 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 $s = 25$ 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 $\lesssim 1 - (1 - .7^{25})^{40} = .005$

Expected 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.

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)

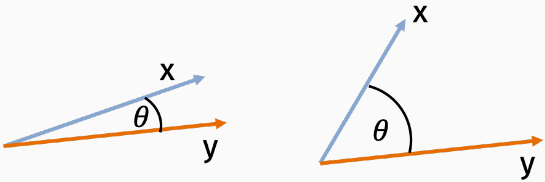
Let q be the closest database vector to $\mathbf{y}$. Return a vector $\tilde{\mathbf{q}}$ with $\|\tilde{\mathbf{q}} - \mathbf{y}\|_0 \leq C \cdot \|\mathbf{q} - \mathbf{y}\|_0$ in:

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

Any ideas for how this is done?

Good locality sensitive hash functions exists for many 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 is natural "inverse" for Euclidean distance.

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

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

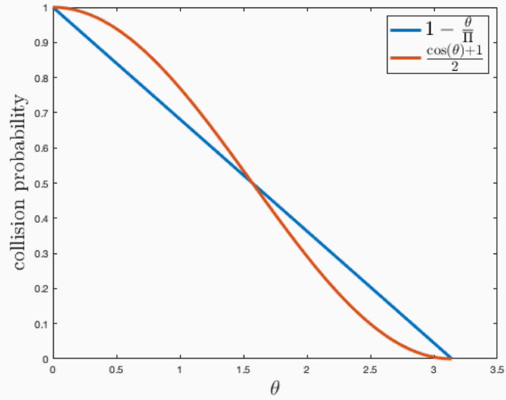
Locality sensitive hash for **cosine similarity**:

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

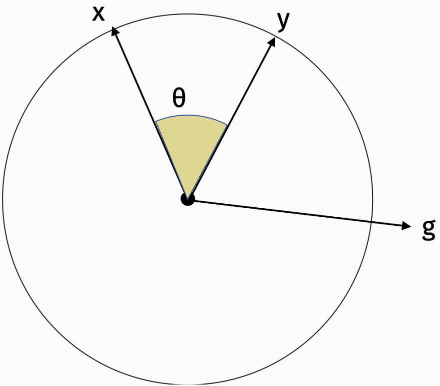
$$\Pr[h(\mathbf{x}) == h(\mathbf{y})] = 1 - \frac{\theta}{\pi} = 1 - \frac{\cos^{-1}(v)}{\pi}$$



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

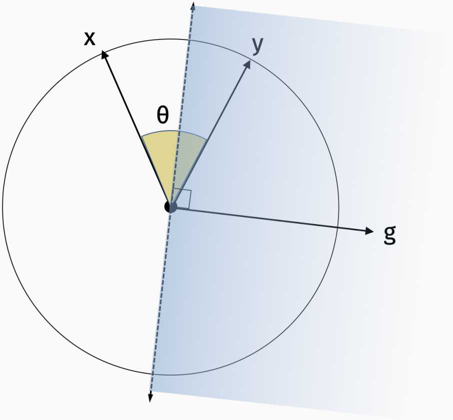
$$\Pr[h(\mathbf{x}) == h(\mathbf{y})] = \left(1 - \frac{\theta}{\Pi}\right)^r$$



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

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

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



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