CS-GY 6763: Lecture 5
Near neighbor search in high dimensions

NYU Tandon School of Engineering, Prof. Christopher Musco

PROJECT

- Sign-up to present or lead discussion for 1 reading group slot. We need presenters for next Friday!

$$\Pi = k\left[\overset{d}{\Pi}\right]$$

Lemma (Distributional JL Lemma)

Let $\Pi \in \mathbb{R}^{k\times d}$ be a random Gaussian/sign matrix. For any two real-valued vectors $\mathbf{q}, \mathbf{y} \in \mathbb{R}^d$, then with probability $1-\delta$,

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

as long as $k = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$.

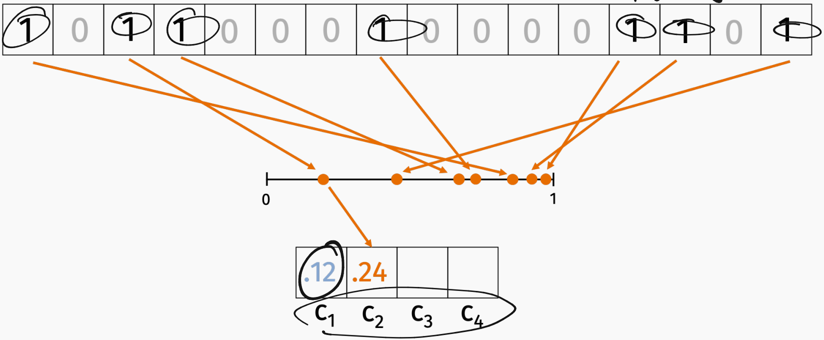
$$\log\left(1/(1/n^2)\right) = \log(n^2)$$
$$= O(\log(n))$$

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

Jaccard similarity

$q, y \in \{0,1\}^d$

$0 \leq \frac{|q \cap y|}{|q \cup y|} \leq 1$



Let $\tilde{J}(C(\mathbf{q}), C(\mathbf{y})) = \frac{1}{k}\sum_{i=1}^{k} \mathbb{1}[C(\mathbf{q})_i = C(\mathbf{y})_i]$.

Lemma ~~(Distributional JL Lemma)~~ Jaccard Similarity lemma

For any two binary vectors $\mathbf{q}, \mathbf{y} \in \mathbb{R}^d$, with probability $1-\delta$,

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

worst

as long as $k = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$.

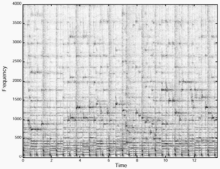
Recall that $J(\mathbf{q},\mathbf{y}) = \frac{|\mathbf{q}\cap\mathbf{y}|}{|\mathbf{q}\cup\mathbf{y}|}$.

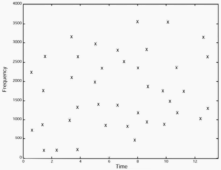
$J(q,y) + \epsilon\sqrt{J(q,y)}$ better

worse

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

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.

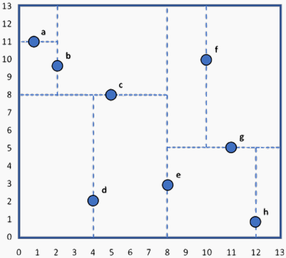
- The Shazam problem.
- Audio + video search.
- Finding duplicate or near duplicate documents.
- Detecting seismic events.

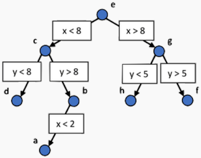
How does similarity sketching help in these applications?

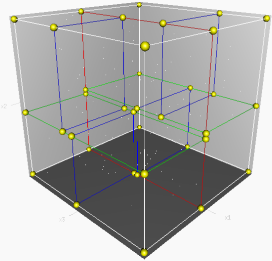
- Improves runtime of "linear scan" from $O(nd)$ to $O(nk)$.
- Improves space complexity from $O(nd)$ to $O(nk)$. This can be super important – e.g. if it means the linear scan only accesses vectors in fast memory.

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

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



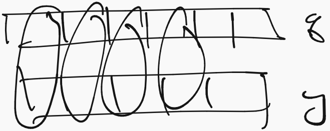




Runtime is roughly $O(d \cdot \min(n, 2^d))$, which is only sublinear for $d = o(\log n)$.

$O(d)$

$O(dn)$



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] Product Quantization

Key Insight of LSH: Trade worse space-complexity for better time-complexity. I.e. typically use more than $O(n)$ space.

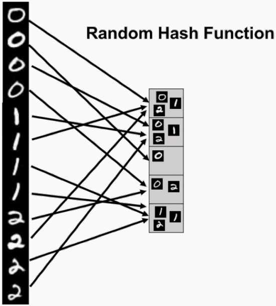
Let $h : \mathbb{R}^d \to \{1, \ldots, m\}$ be a random hash function.

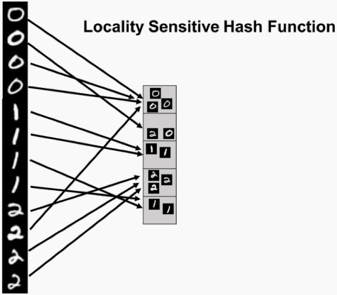
$J(q, y)$

$-\|q - y\|_2^2$

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

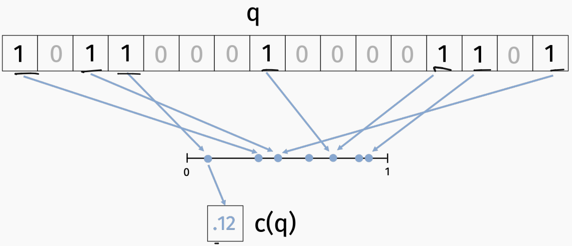
- Higher when $\mathbf{q}$ and $\mathbf{y}$ are more similar, i.e. $s(\mathbf{q}, \mathbf{y})$ is higher.
- Lower when $\mathbf{q}$ and $\mathbf{y}$ are more dissimilar, i.e. $s(\mathbf{q}, \mathbf{y})$ is lower.





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

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



$\Pr[c(q) == c(y)] = J(q, y)$

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

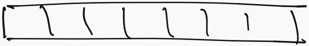
$$\Pr[h(\mathbf{q}) == h(\mathbf{y})] = v + \boxed{(1-v)\cdot\frac{1}{m}}$$

usually super small.

m is chosen large.

NEAR NEIGHBOR SEARCH

Basic approach for near neighbor search in a database.



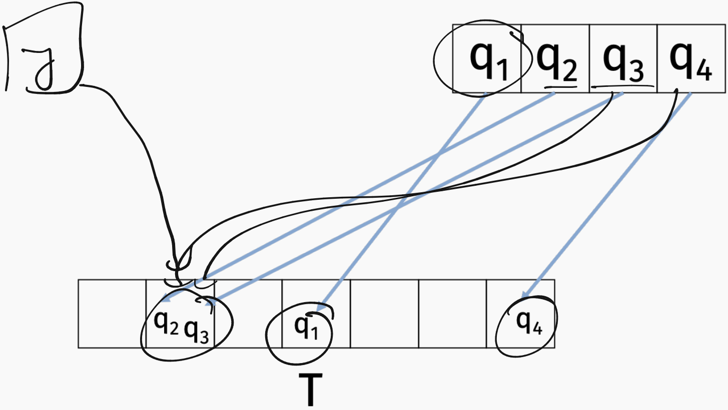
Pre-processing:

- 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 $T(h(\mathbf{q}_i))$. $g(c(q_i))$

Query: $q_1, \ldots, q_n$

- 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.



q1 q2 q3 q4
q2 q3
q1
q4
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 $< .2$.

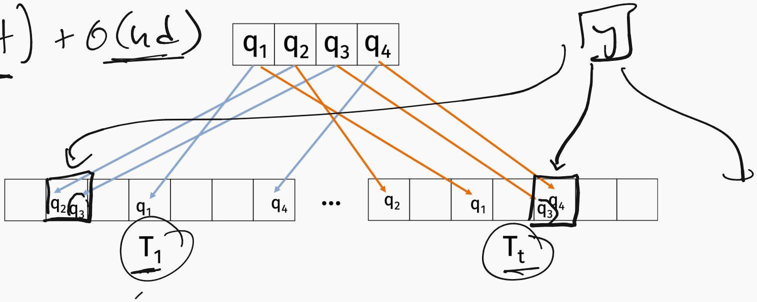
Suppose the nearest database point $\mathbf{q}$ has $J(\mathbf{y}, \mathbf{q}) = .4$.

What's the probability we do not find q?

$$1 - .4 = .6$$

REDUCING FALSE NEGATIVE RATE

$O(n \cdot t) + O(nd)$



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 $T_j(h_j(\mathbf{q}_i))$.

$t = 10$

99%

$J(q, y) = v$

Query:

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

$t = 10$

t

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

What's the probability we find q?

$1 - (1 - v)^t$ $\qquad 1 - (.6)^t \geq 99\%$

89%

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?

$$1 - (1-v)^t = 1 - (.8)^{10} = 89\%$$

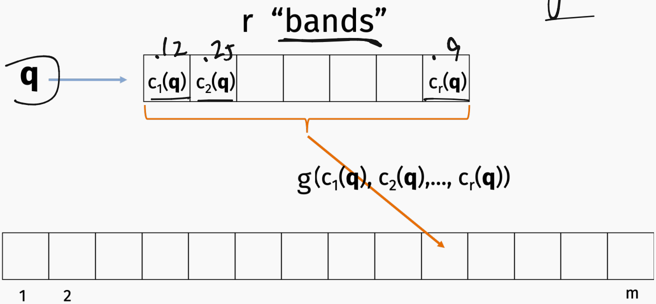
REDUCING FALSE POSITIVES

Change our locality sensitive hash function. $r = 1$

Tunable LSH for Jaccard similarity: $g : [0,1]^r \to 1 \ldots m$

- 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 $h(\mathbf{x}) = g(c_1(\mathbf{x}), \ldots, c_r(\mathbf{x}))$.

$g(y) = g(z)$



Tunable LSH for Jaccard similarity: r

- 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 $\Pr[h(\mathbf{q}) == h(\mathbf{y})] =$ $v^r + \boxed{\frac{1}{m}(1 - v^r)}$

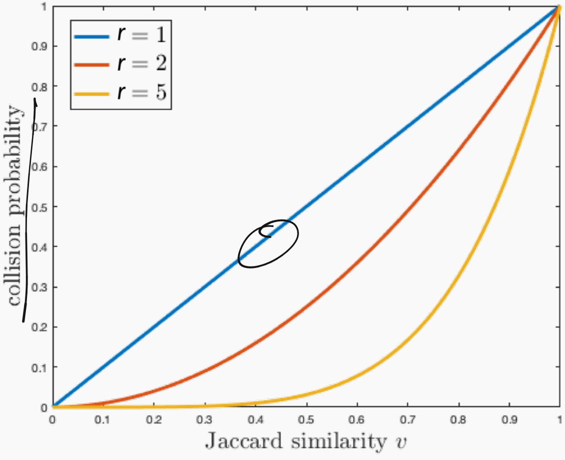
very small

$$\left.\begin{array}{l} c_1(q) = c_1(y) \\ \vdots \\ c_r(q) = c_r(y) \end{array}\right\} h(q) = h(y)$$

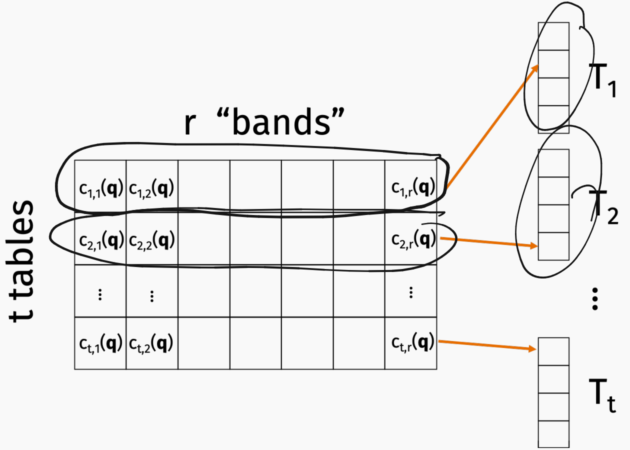
v

v^2

v^5


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
$\downarrow$	$\uparrow$

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

False Negatives	False Positives
$\uparrow$	$\downarrow$

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

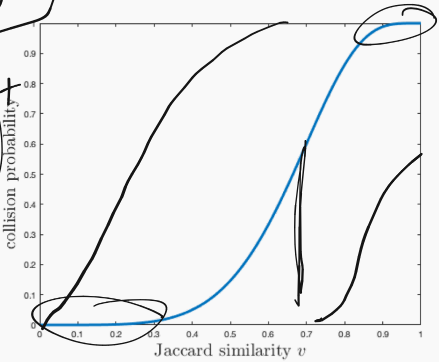
v^r

1 - (Probability land in no buckets with y)

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

1 - (Pr don't land in bucket)^t

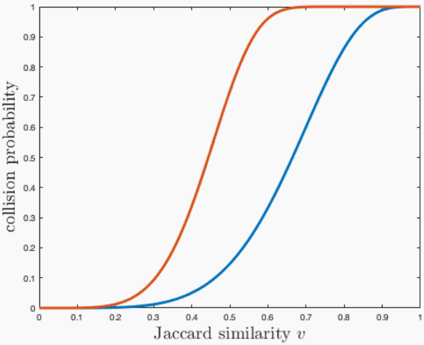
$1 - (1 - v^r)^t$



$r = 5, t = 5$

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

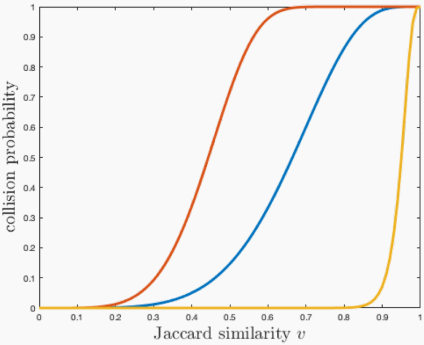
$$\approx 1 - (1 - v^r)^t$$



$$r = 5, t = 40$$

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

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

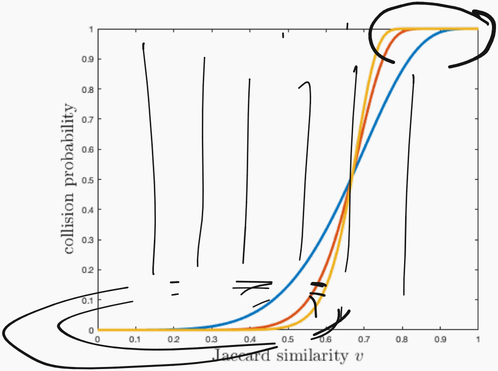


$r = 40, t = 5$

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.

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:

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

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 $\mathbf{y}$.

$$\tilde{O}(n) = O(n \log^{2.3} n)$$

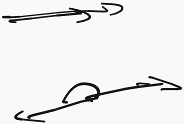
Theorem (Indyk, Motwani, 1998)

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.

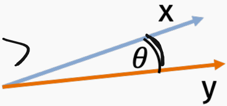
$\|x\|_2 = \|y\|_2 = 1$

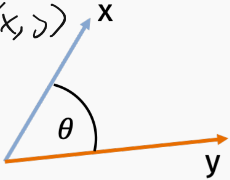


Cosine similarity $\cos(\theta(\mathbf{x}, \mathbf{y})) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}$: $\langle x, y \rangle$

$\|x-y\|_2^2 = \|x\|_2^2 + \|y\|_2^2 - 2\langle x, y \rangle$

$= 2 - 2\langle x, y \rangle$





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

Cosine similarity is natural "inverse" for Euclidean distance.

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

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

$\text{sign}(\langle g, x \rangle)$

g_1
g_2
$\vdots$
g_r

$f(+1\ -1\ -1\ +1)$
$\downarrow$
$\text{sign}(\langle g_i, x \rangle)$

Locality sensitive hash for **cosine similarity**:

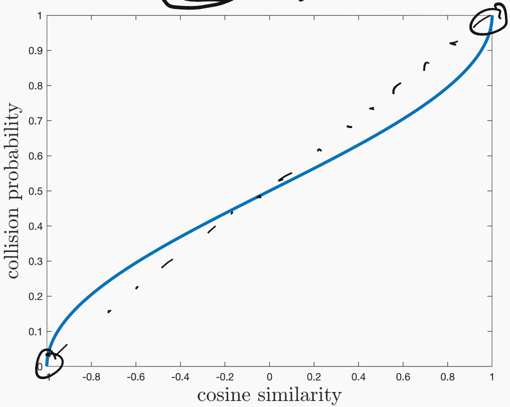
- Let $\mathbf{g} \in \mathbb{R}^d$ be randomly chosen with each entry $\mathcal{N}(0, 1)$.
- Let $f : \{-1, 1\} \to \{1, \ldots, m\}$ be a uniformly random hash function.
- $h : \mathbb{R}^d \to \{1, \ldots, m\}$ is defined $h(\mathbf{x}) = f(\text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle))$.

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

$$\underbrace{\Pr[\text{sign}(\langle g, x \rangle) = \text{sign}(\langle g, y \rangle)]} + \frac{1}{n}$$

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

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



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

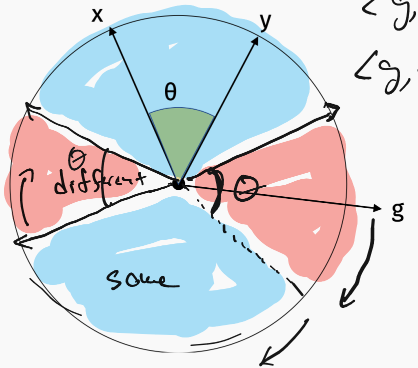
- Let $\mathbf{g}_1, \ldots, \mathbf{g}_r \in \mathbb{R}^d$ be randomly chosen with each entry $\mathcal{N}(0, 1)$.
- Let $f : \{-1, 1\}^r \to \{1, \ldots, m\}$ be a uniformly random hash function.
- $h : \mathbb{R}^d \to \{1, \ldots, m\}$ is defined $h(\mathbf{x}) = f([\text{sign}(\langle \mathbf{g}_1,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

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

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

$1 - \frac{2\theta}{2\pi}$

$1 - \frac{\theta}{\pi}$



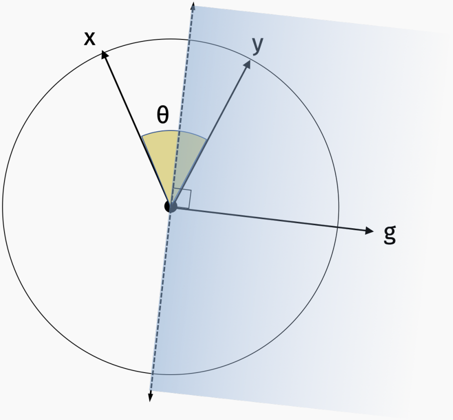
$\langle g, x \rangle = -1$

$\langle g, y \rangle = 1$

$1 - \frac{\theta}{\pi}$

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

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

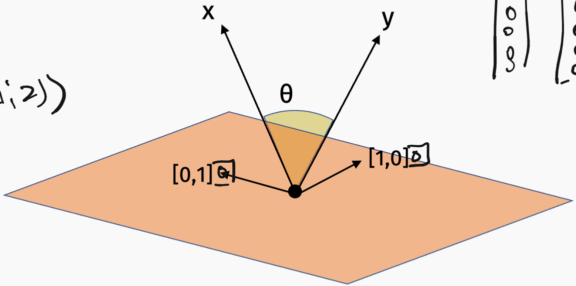


$\Pr[h(\mathbf{x}) == h(\mathbf{y})] \approx$ probability $\mathbf{x}$ and $\mathbf{y}$ are on the same side of hyperplane orthogonal to $\mathbf{g}$.

$\langle x, y \rangle$

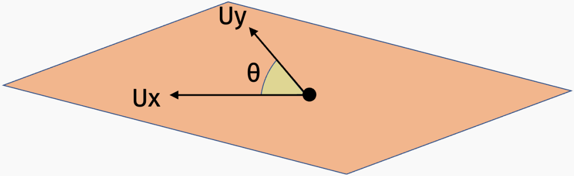
$\langle x(1:2), y(1:2) \rangle$

$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$



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**.

$U^T U = I = U U^T$



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

Claim:

2d analysis

$\langle Ug, Ux\rangle = \langle g, x\rangle$

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

$\langle Ug, Uy\rangle = \langle g, y\rangle$

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

Why?

Rotational Invariance of Gaussians

$g \sim Ug$ identically distributed to Ug for any U.

$= \Pr[\text{sign}(\langle Ug, Ux\rangle) = \text{sign}(\langle Ug, Uy\rangle)]$

$g^T \cancel{U^T U} x = g^T x \qquad g^T U^T U y = g^T y$

BREAK

Have some function $f : \mathbb{R}^d \to \mathbb{R}$. Want to find $\mathbf{x}^*$ such that:

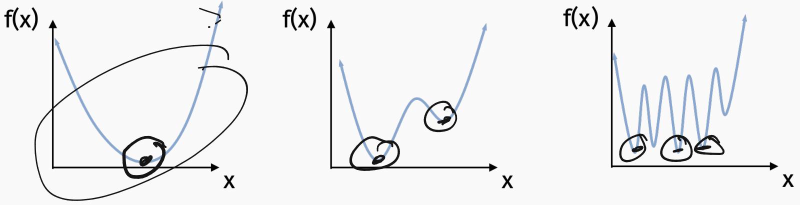
$$f(\mathbf{x}^*) = \min_{\mathbf{x}} f(\mathbf{x}).$$

Or at least $\hat{\mathbf{x}}$ which is close to a minimum. E.g. $f(\hat{\mathbf{x}}) \leq \min_{\mathbf{x}} f(\mathbf{x}) + \epsilon$

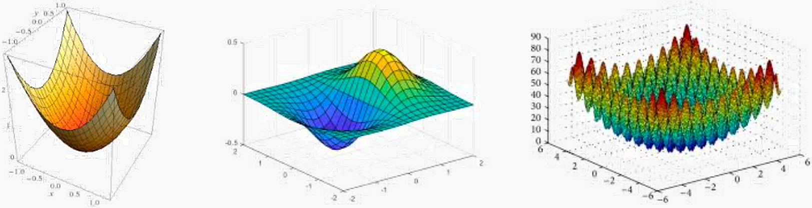
Often we have some additional constraints:

- $\mathbf{x} > 0$.
- $\|\mathbf{x}\|_2 \leq R$, $\|\mathbf{x}\|_1 \leq R$.
- $\mathbf{a}^T\mathbf{x} > c$.

Dimension $d = 1$:



Dimension $d = 2$:



Continuouos optimization is the foundation of modern machine learning.

Supervised learning: Want to learn a model that maps inputs

- numerical data vectors
- images, video
- text documents

to predictions

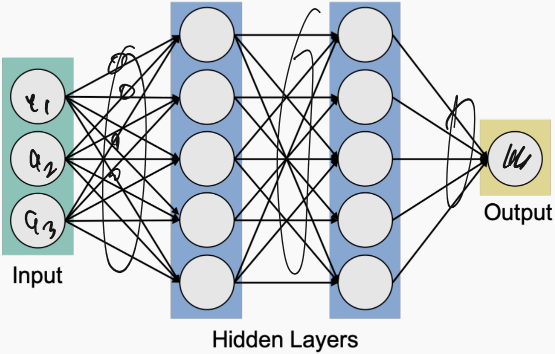
- numerical value (probability stock price increases)
- label (is the image a cat? does the image contain a car?)
- decision (turn car left, rotate robotic arm)

Let $M_{\mathbf{x}}$ be a model with parameters $\mathbf{x} = \{x_1, \ldots, x_k\}$, which takes as input a data vector $\mathbf{a}$ and outputs a prediction.

Example:

$$M_{\mathbf{x}}(\mathbf{a}) = \mathrm{sign}(\mathbf{a}^T\mathbf{x}) \qquad \pm 1$$

Example:



$\mathbf{x} \in \mathbb{R}^{(\#\text{ of connections})}$ is the parameter vector containing all the network weights.

Classic approach in supervised learning: Find a model that works well on data that you already have the answer for (labels, values, classes, etc.).

- Model $M_{\mathbf{x}}$ parameterized by a vector of numbers $\mathbf{x}$.
- Dataset $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(n)}$ with outputs $y^{(1)}, \ldots, y^{(n)}$.

Want to find $\hat{\mathbf{x}}$ so that $M_{\hat{\mathbf{x}}}(\mathbf{a}^{(i)}) \approx y^{(i)}$ for $i \in 1, \ldots, n$.

How do we turn this into a function minimization problem?

Loss function $L(M_{\mathbf{x}}(\mathbf{a}), y)$**:** Some measure of distance between prediction $M_{\mathbf{x}}(\mathbf{a})$ and target output y. Increases if they are further apart.

- Squared (ℓ_2) loss: $|M_{\mathbf{x}}(\mathbf{a}) - y|^2$
- Absolute deviation (ℓ_1) loss: $|M_{\mathbf{x}}(\mathbf{a}) - y|$
- Hinge loss: $1 - y \cdot M_{\mathbf{x}}(\mathbf{a})$
- Cross-entropy loss (log loss).
- Etc.

Finite Sum Structure

x^3 x^4

Empirical risk minimization:

$$f(\mathbf{x}) = \sum_{i=1}^{n} L\left(M_{\mathbf{x}}(\mathbf{a}^{(i)}), y^{(i)}\right)$$

Solve the optimization problem $\min_{\mathbf{x}} f(\mathbf{x})$.

- $M_{\mathbf{x}}(\mathbf{a}) = \mathbf{x}^T\mathbf{a}$. $\mathbf{x}$ contains the regression coefficients.
- $L(z, y) = |z - y|^2$.
- $f(\mathbf{x}) = \sum_{i=1}^{n} |\mathbf{x}^T\mathbf{a}^{(i)} - y^{(i)}|^2$

$$f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$$

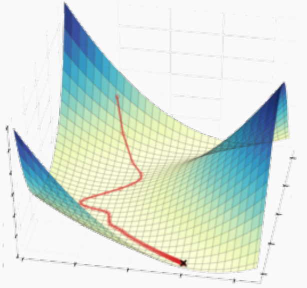
where $\mathbf{A}$ is a matrix with $\mathbf{a}^{(i)}$ as its i^{th} row and $\mathbf{y}$ is a vector with $y^{(i)}$ as its i^{th} entry.

The choice of algorithm to minimize $f(\mathbf{x})$ will depend on:

- The form of $f(\mathbf{x})$ (is it linear, is it quadratic, does it have finite sum structure, etc.)
- If there are any additional constraints imposed on $\mathbf{x}$. E.g. $\|\mathbf{x}\|_2 \leq c$.

What are some example algorithms for continuous optimization?

Gradient descent: A greedy algorithm for minimizing functions of multiple variables that often works amazingly well.



Runtime generally scales <u>linearly</u> with the dimension of $\mathbf{x}$ (although this is a bit of an over-simplification).

- Cutting plane methods (e.g. center-of-gravity, ellipsoid)
- Interior point methods

Fast and more accurate in low-dimensions, slower in very high dimensions. Generally runtime scales <u>polynomially</u> with the dimension of $\mathbf{x}$.

For $i = 1, \ldots, d$, let x_i be the i^{th} entry of $\mathbf{x}$. Let $\mathbf{e}^{(i)}$ be the i^{th} standard basis vector.

Partial derivative:

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{e}^{(i)}) - f(\mathbf{x})}{t}$$

Directional derivative:

$$D_{\mathbf{v}} f(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t}$$

Gradient:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \frac{\partial f}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_d}(\mathbf{x}) \end{bmatrix}$$

Directional derivative:

$$D_{\mathbf{v}} f(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t} = \nabla f(\mathbf{x})^T \mathbf{v}.$$

Given a function f to minimize, assume we have:

- **Function oracle**: Evaluate $f(\mathbf{x})$ for any $\mathbf{x}$.
- **Gradient oracle**: Evaluate $\nabla f(\mathbf{x})$ for any $\mathbf{x}$.

We view the implementation of these oracles as black-boxes, but they can often require a fair bit of computation.

Linear least-squares regression:

- Given $\mathbf{a}^{(1)}, \ldots \mathbf{a}^{(n)} \in \mathbb{R}^d$, $y^{(1)}, \ldots y^{(n)} \in \mathbb{R}$.
- Want to minimize:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \left(\mathbf{x}^T \mathbf{a}^{(i)} - y^{(i)} \right)^2 = \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2.$$

$$\frac{\partial f}{\partial x_j} = \sum_{i=1}^{n} 2 \left(\mathbf{x}^T \mathbf{a}^{(i)} - y^{(i)} \right) \cdot a_j^{(i)} = (2\mathbf{A}\mathbf{x} - \mathbf{y})^T \boldsymbol{\alpha}^{(j)}$$

where $\boldsymbol{\alpha}^{(j)}$ is the j^{th} <u>column</u> of $\mathbf{A}$.

$$\nabla f(\mathbf{x}) = 2\mathbf{A}^T (\mathbf{A}\mathbf{x} - \mathbf{y})$$

What is the time complexity of a gradient oracle for $\nabla f(\mathbf{x})$?

Greedy approach: Given a starting point $\mathbf{x}$, make a small adjustment that decreases $f(\mathbf{x})$. In particular, $\mathbf{x} \leftarrow \mathbf{x} + \eta\mathbf{v}$ and $f(\mathbf{x}) \leftarrow f(\mathbf{x} + \eta\mathbf{v})$.

What property do I want in $\mathbf{v}$?

Leading question: When η is small, what's an approximation for $f(\mathbf{x} + \eta\mathbf{v}) - f(\mathbf{x})$?

$$f(\mathbf{x} + \eta\mathbf{v}) - f(\mathbf{x}) \approx$$

$$D_{\mathbf{v}} f(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t} = \nabla f(\mathbf{x})^T \mathbf{v}.$$

So:

$$f(\mathbf{x} + \eta \mathbf{v}) - f(\mathbf{x}) \approx$$

How should we choose $\mathbf{v}$ so that $f(\mathbf{x} + \eta \mathbf{v}) < f(\mathbf{x})$?

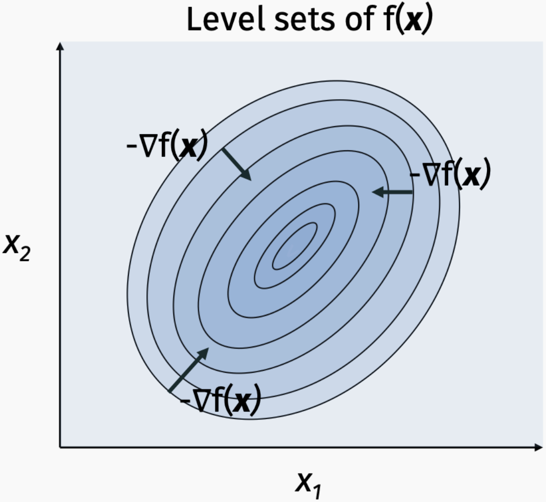
Prototype algorithm:

- Choose starting point $\mathbf{x}^{(0)}$.
- For $i = 0, \ldots, T$:
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f(\mathbf{x}^{(i)})$
- Return $\mathbf{x}^{(T)}$.

η is a step-size parameter, which is often adapted on the go. For now, assume it is fixed ahead of time.

1 dimensional example:

2 dimensional example:



For a convex function $f(\mathbf{x})$: For sufficiently small η and a sufficiently large number of iterations T, gradient descent will converge to a **near global minimum**:

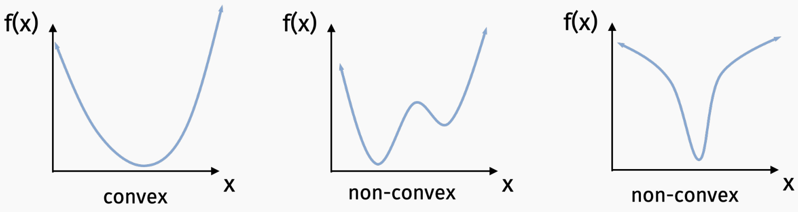
$$f(\mathbf{x}^{(T)}) \leq f(\mathbf{x}^*) + \epsilon.$$

Examples: least squares regression, logistic regression, kernel regression, SVMs.

For a non-convex function $f(\mathbf{x})$: For sufficiently small η and a sufficiently large number of iterations T, gradient descent will converge to a **near stationary point**:

$$\|\nabla f(\mathbf{x}^{(T)})\|_2 \leq \epsilon.$$

Examples: neural networks, matrix completion problems, mixture models.



One issue with non-convex functions is that they can have local minima. Even when they don't, convergence analysis requires different assumptions than convex functions.

We care about how fast gradient descent and related methods converge, not just that they do converge.

- Bounding iteration complexity requires placing some assumptions on $f(\mathbf{x})$.
- Stronger assumptions lead to better bounds on the convergence.

Understanding these assumptions can help us design faster variants of gradient descent (there are many!).