COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Fall 2020.

Lecture 8

LOGISTICS

- We uploaded Problem Set 2 last night. It will be due Monday 9/28 at 8pm.
- Start early. Give yourself time to mull over the problems.
- Some reminders from your friendly 514 grading staff:
  - You need to **mark your group-mates as part of the submission in Gradescope**. Just having their name on written on the front page is not enough.
  - Tag the location of each individual subquestion, not just the first page of the full question.
  - If you write in pencil please be sure to write darkly. It can be very hard to read once scanned.
- Quiz 4 is due Monday at 8pm.

SUMMARY

Last Class:

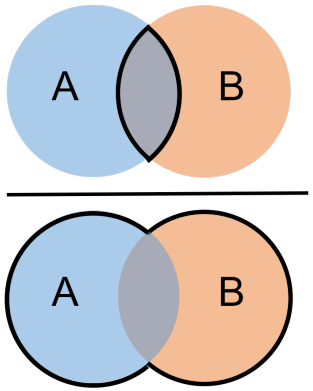
- Boosting the success probability of distinct elements estimation with the median trick.
- Sketched the idea of practical distinct elements algorithms: LogLog and HyperLogLog.
- Started on fast similarity search. MinHashing to estimate the Jaccard similarity between two sets.

This Class:

- MinHash and locality sensitive hashing (LSH).
- Application of LSH to fast similarity search.

JACCARD SIMILARITY

Jaccard Similarity: $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\#\text{ shared elements}}{\#\text{ total elements}}$.



Two Common Use Cases:

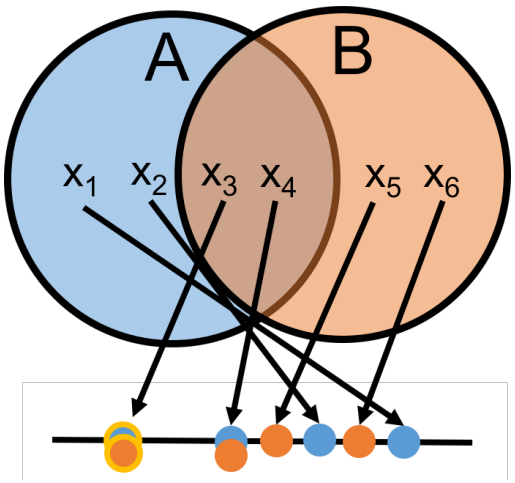
- **Near Neighbor Search:** Have a database of n sets/bit strings and given a set A, want to find if it has high similarity to anything in the database. Naively $\Omega(n)$ time.
- **All-pairs Similarity Search:** Have n different sets/bit strings. Want to find all pairs with high similarity. Naively $\Omega(n^2)$ time.

MINHASHING FOR JACCARD SIMILARITY

$MinHash(A) = \min_{a \in A} \mathbf{h}(a)$ where $\mathbf{h} : U \to [0, 1]$ is random.

For two sets A and B, what is $\Pr(MinHash(A) = MinHash(B))$?

Claim: $MinHash(A) = MinHash(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.



$$\Pr(MinHash(A) = MinHash(B)) = ? \frac{|A \cap B|}{\text{total \# items hashed}}$$

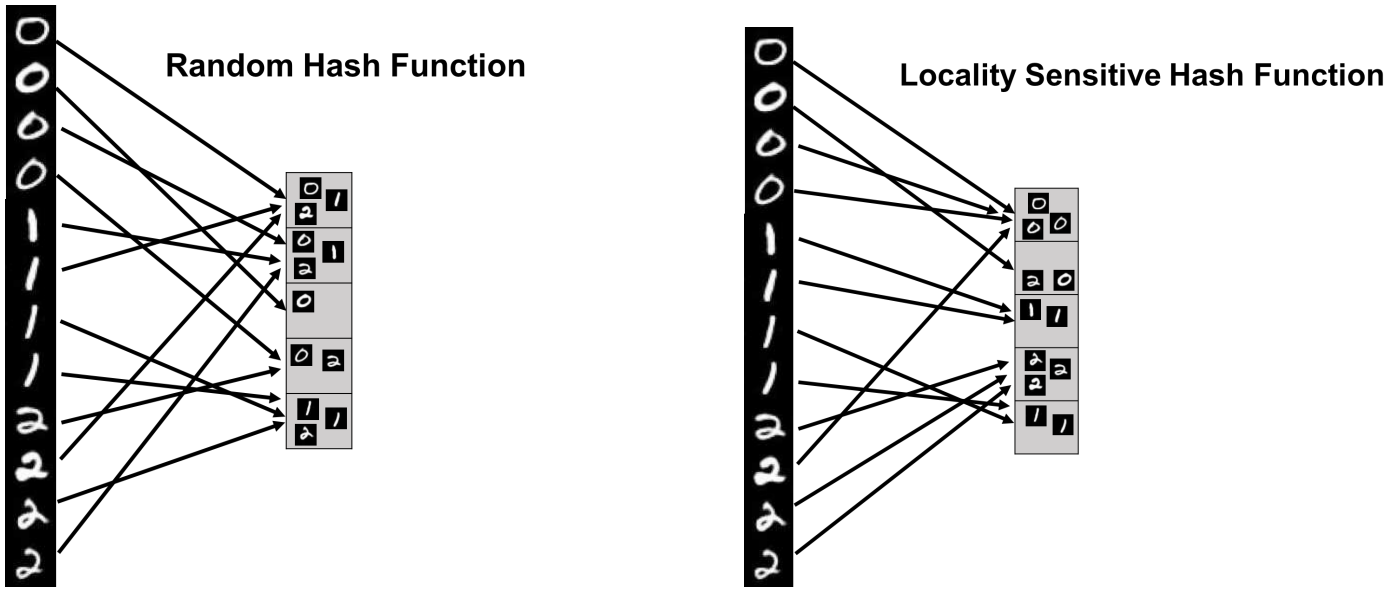
$$= \frac{|A \cap B|}{|A \cup B|} = J(A, B).$$

LOCALITY SENSITIVE HASHING

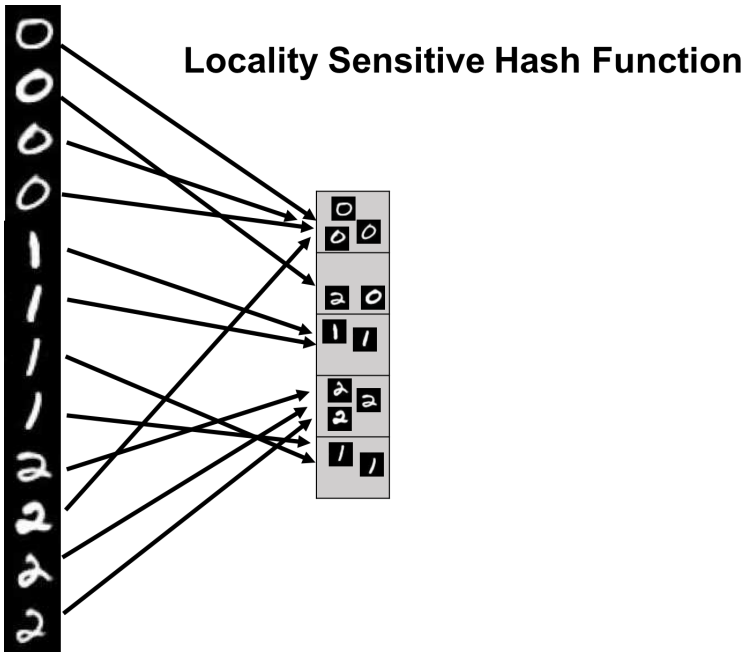
Upshot: MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

$$\Pr(\mathit{MinHash}(A) = \mathit{MinHash}(B)) = J(A, B).$$

- An instance of locality sensitive hashing (LSH).
- A hash function where the collision probability is higher when two inputs are more similar (can design different functions for different similarity metrics.)



How does locality sensitive hashing (LSH) help with similarity search?



- **Near Neighbor Search:** Given item x, compute $\mathbf{h}(x)$. Only search for similar items in the $\mathbf{h}(x)$ bucket of the hash table.
- **All-pairs Similarity Search:** Scan through all buckets of the hash table and look for similar pairs within each bucket.
- We will use $\mathbf{h}(x) = \mathbf{g}(\mathit{MinHash}(x))$ where $\mathbf{g} : [0, 1] \to [n]$ is a random hash function. Why?

Goal: Given a document y, identify all documents x in a database with Jaccard similarity (of their shingle sets) $J(x, y) \geq 1/2$.

Our Approach:

- Create a hash table of size m, choose a random hash function $\mathbf{g} : [0, 1] \to [m]$, and insert every item x into bucket $\mathbf{g}(\mathit{MinHash}(x))$. Search for items similar to y in bucket $\mathbf{g}(\mathit{MinHash}(y))$.
- What is $\Pr[\mathbf{g}(\mathit{MinHash}(x)) = \mathbf{g}(\mathit{MinHash}(y))]$ assuming $J(x, y) = 1/2$ and $\mathbf{g}$ is collision free?
- For every document x in your database with $J(x, y) \geq 1/2$ what is the probability you will find x in bucket $\mathbf{g}(\mathit{MinHash}(y))$?

REDUCING FALSE NEGATIVES

With a simple use of MinHash, we miss a match x with $J(x, y) = 1/2$ with probability 1/2. How can we reduce this false negative rate?

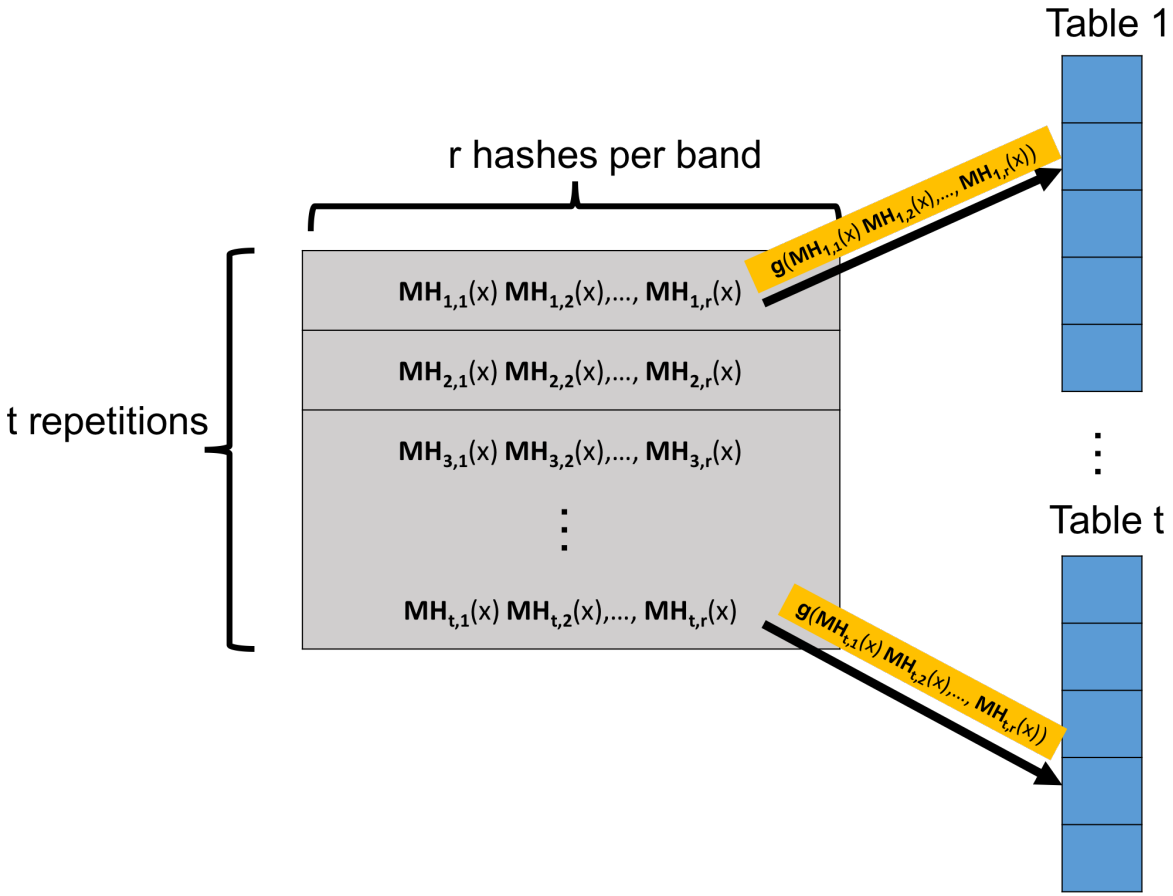
Repetition: Run MinHash t times independently, to produce hash values $MH_1(x), \ldots, MH_t(x)$. Apply random hash function $\mathbf{g}$ to map all these values to locations in t hash tables.

- To search for items similar to y, look at all items in bucket $\mathbf{g}(MH_1(y))$ of the 1^{st} table, bucket $\mathbf{g}(MH_2(y))$ of the 2^{nd} table, etc.
- What is the probability that x with $J(x, y) = 1/2$ is in at least one of these buckets, assuming for simplicity $\mathbf{g}$ has no collisions?
  1– (probability in *no* buckets) $= 1 - \left(\frac{1}{2}\right)^t \approx .99$ for $t = 7$.
- What is the probability that x with $J(x, y) = 1/4$ is in at least one of these buckets, assuming for simplicity $\mathbf{g}$ has no collisions?
  1– (probability in *no* buckets) $= 1 - \left(\frac{3}{4}\right)^t \approx .87$ for $t = 7$.

Potential for a lot of false positives! Slows down search time.

BALANCING HIT RATE AND QUERY TIME

We want to balance a small probability of false negatives (a high hit rate) with a small probability of false positives (a small query time.)



Create *t* hash tables. Each is indexed into not with a single MinHash value, but with *r* values, appended together. A length *r* signature.

BALANCING HIT RATE AND QUERY TIME

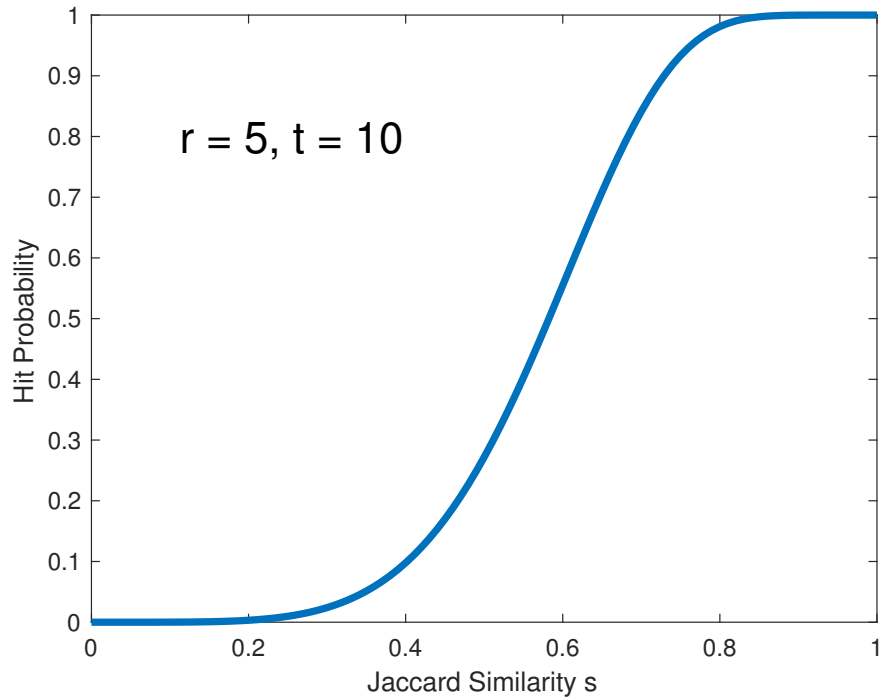
Consider searching for matches in t hash tables, using MinHash signatures of length r. For x and y with Jaccard similarity $J(x, y) = s$:

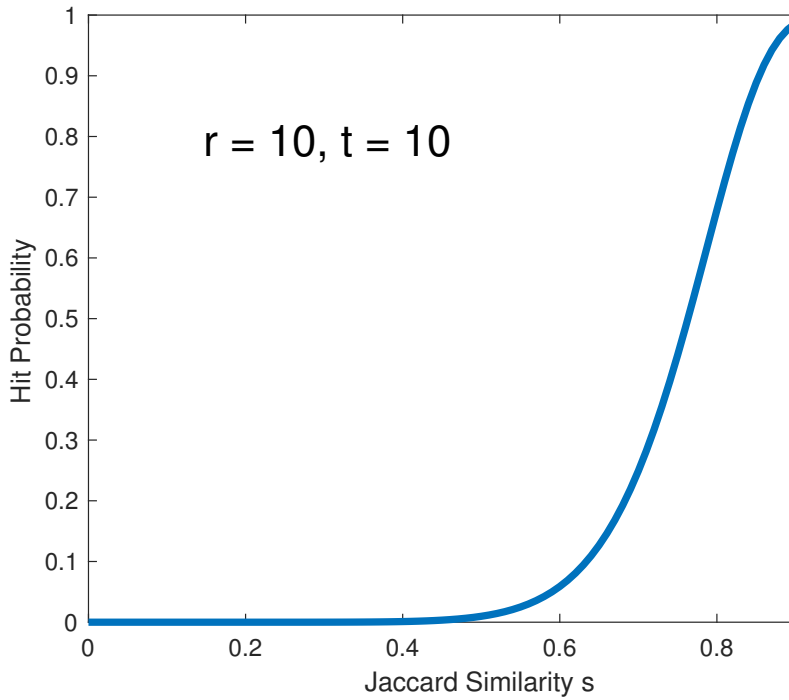
- Probability that a single hash matches.
  $\Pr\left[MH_{i,j}(x) = MH_{i,j}(y)\right] = J(x, y) = s$.
- Probability that x and y having matching signatures in repetition i.
  $\Pr\left[MH_{i,1}(x), \ldots, MH_{i,r}(x) = MH_{i,1}(y), \ldots, MH_{i,r}(y)\right] = s^r$.
- Probability that x and y don't match in repetition i: $1 - s^r$.
- Probability that x and y don't match in *all repetitions*: $(1 - s^r)^t$.
- Probability that x and y match in at least one repetition:

Hit Probability: $1 - (1 - s^r)^t$.

THE S-CURVE

Using t repetitions each with a signature of r MinHash values, the probability that x and y with Jaccard similarity $J(x, y) = s$ match in at least one repetition is: $1 - (1 - s^r)^t$.





r and t are tuned depending on application. 'Threshold' when hit probability is 1/2 is $\approx (1/t)^{1/r}$. E.g., $\approx (1/30)^{1/5} = .51$ in this case.

S-CURVE EXAMPLE

For example: Consider a database with $10,000,000$ audio clips. You are given a clip x and want to find any y in the database with $J(x,y) \geq .9$.

- There are 10 true matches in the database with $J(x,y) \geq .9$.
- There are $10,000$ near matches with $J(x,y) \in [.7,.9]$.

With signature length $r = 25$ and repetitions $t = 50$, hit probability for $J(x,y) = s$ is $1-(1-s^{25})^{50}$.

- Hit probability for $J(x,y) \geq .9$ is $\geq 1-(1-.9^{20})^{40} \approx .98$
- Hit probability for $J(x,y) \in [.7,.9]$ is $\leq 1-(1-.9^{20})^{40} \approx .98$
- Hit probability for $J(x,y) \leq .7$ is $\leq 1-(1-.7^{20})^{40} \approx .007$

Expected Number of Items Scanned: (proportional to query time)

$$\leq 10 + .98 * 10,000 + .007 * 9,989,990 \approx 80,000 \ll 10,000,000.$$

HASHING FOR DUPLICATE DETECTION

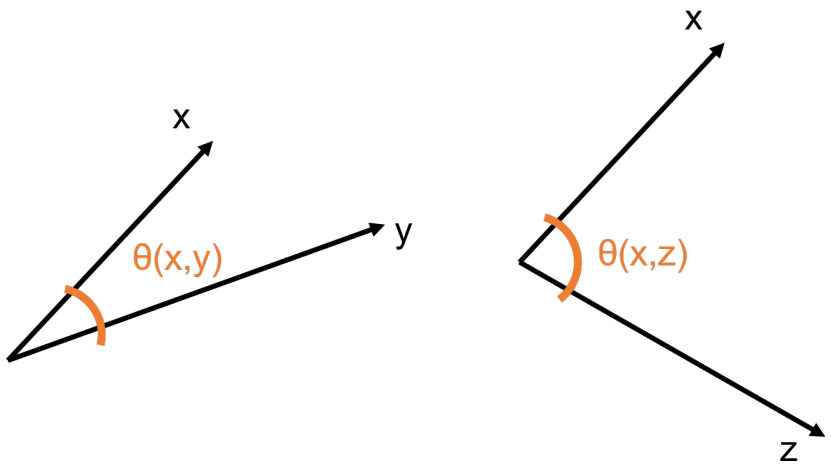
	Hash Table	Bloom Filters	MinHash Similarity Search	Distinct Elements	
Goal	Check if x is a duplicate of any y in database and return y.	Check if x is a duplicate of y in database.	Check if x is a duplicate of any y in database and return y.	Count # of items, excluding duplicates.	**Go**
Space	$O(n)$ items	$O(n)$ bits	$O(n \cdot t)$ items (when t tables used)	$O\left(\frac{\log\log n}{\epsilon^2}\right)$	**Spa**
Query Time	$O(1)$	$O(1)$	Potentially $o(n)$	NA	**Query**
Approximate Duplicates?	✗	✗	✓	✗	**Approx Duplic**

All different variants of detecting duplicates/finding matches in large datasets. An important problem in many contexts!

GENERALIZING LOCALITY SENSITIVE HASHING

Repetition and s-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

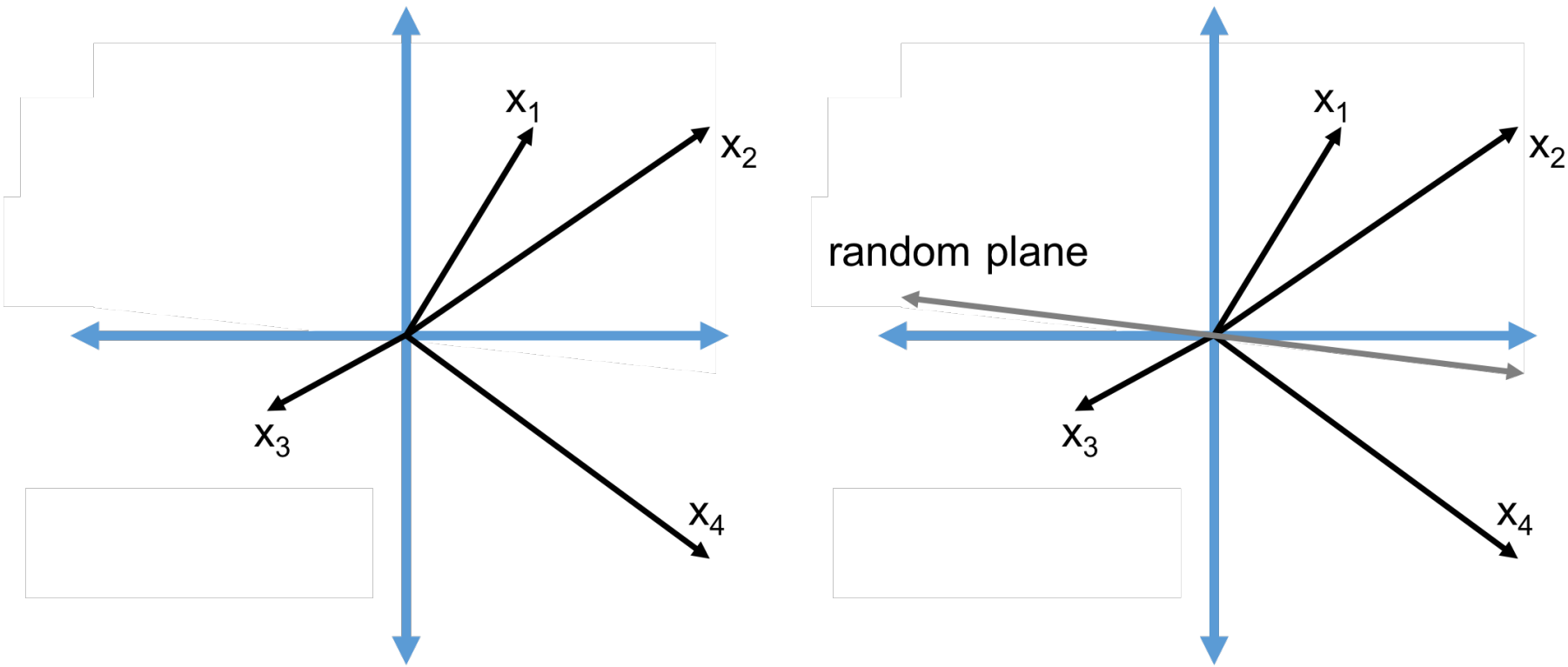
- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.



Cosine Similarity: $\cos(\theta(x,y)) = \frac{\langle x,y \rangle}{\|x\|_2 \cdot \|y\|_2}$.

- $\cos(\theta(x,y)) = 1$ when $\theta(x,y) = 0°$ and $\cos(\theta(x,y)) = 0$ when $\theta(x,y) = 90°$, and $\cos(\theta(x,y)) = -1$ when $\theta(x,y) = 180°$

SimHash Algorithm: LSH for cosine similarity.

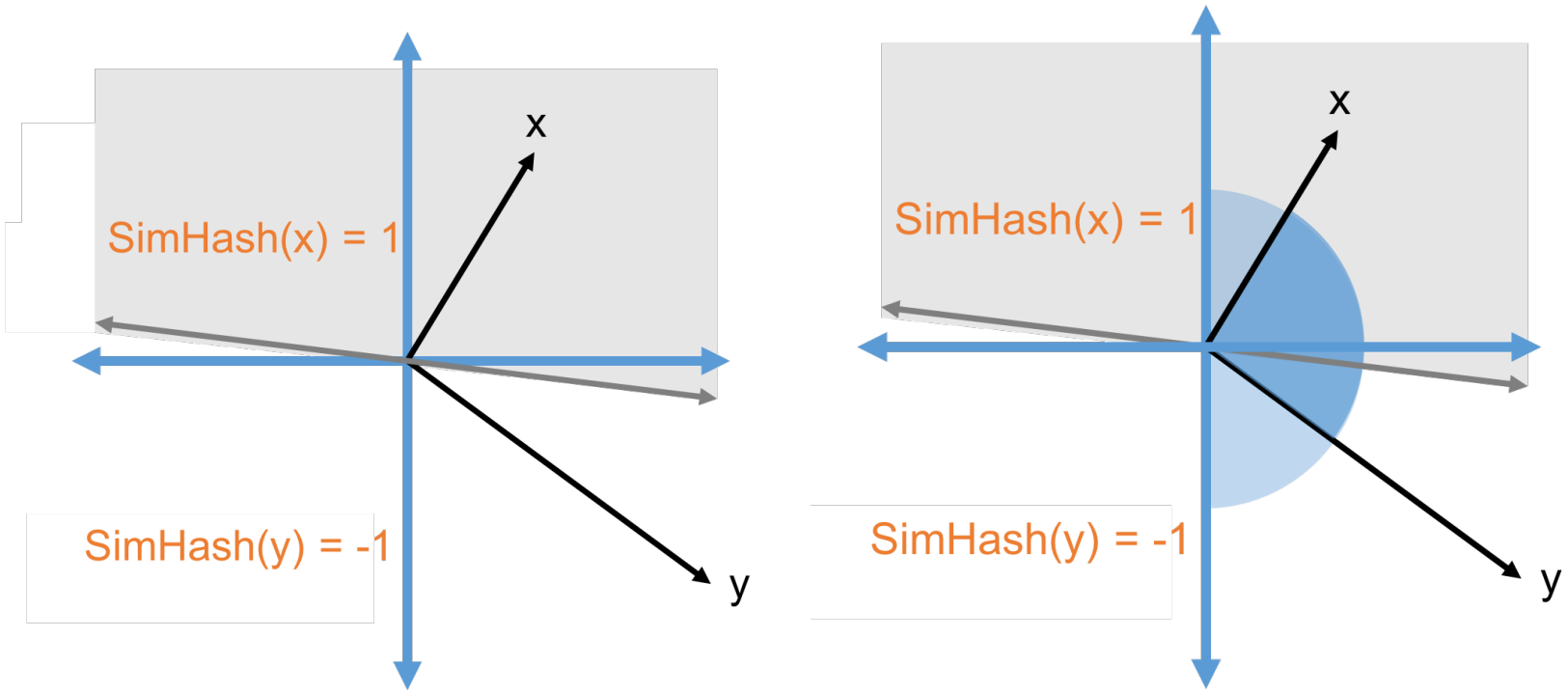


$SimHash(x) = \text{sign}(\langle x, t \rangle)$ for a random vector t.

What is $\Pr[SimHash(x) = SimHash(y)]$?

What is $\Pr[SimHash(x) = SimHash(y)]$?

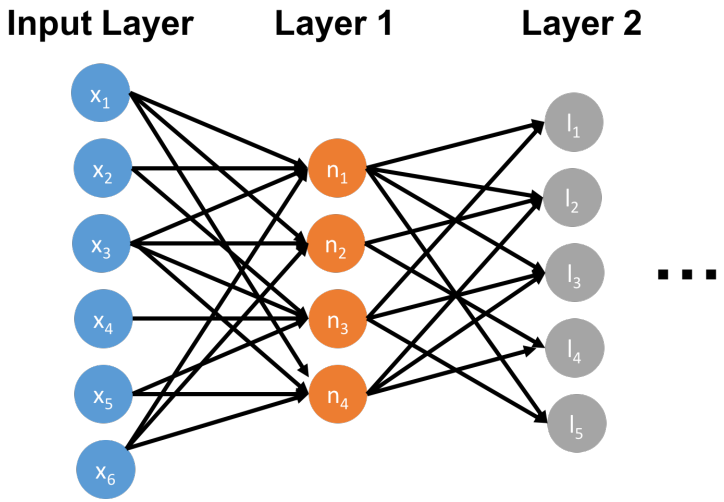
$SimHash(x) \neq SimHash(y)$ when the plane separates x from y.



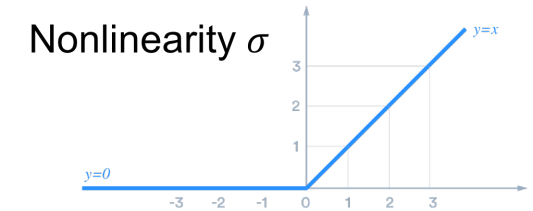
- $\Pr[SimHash(x) \neq SimHash(y)] = \frac{\theta(x,y)}{\pi}$
- $\Pr[SimHash(x) = SimHash(y)] = 1 - \frac{\theta(x,y)}{\pi} \approx \frac{\cos(\theta(x,y))+1}{2}$

Questions on MinHash and Locality Sensitive Hashing?

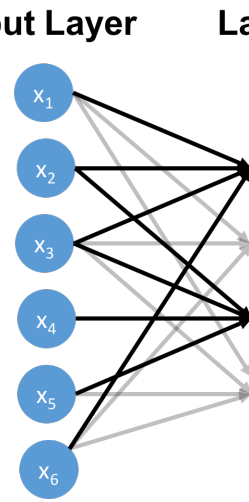
Many applications outside traditional similarity search. E.g., approximate neural net computation (Anshumali Shrivastava).



...

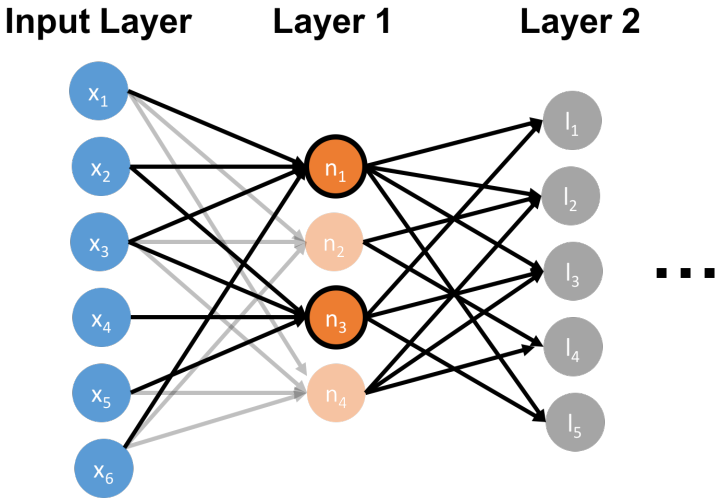


$$n_i = \sigma\left(\sum_{j=1}^{m} w(x_j, n_i) \cdot x_j\right) = \sigma(\langle w_i, x\rangle)$$

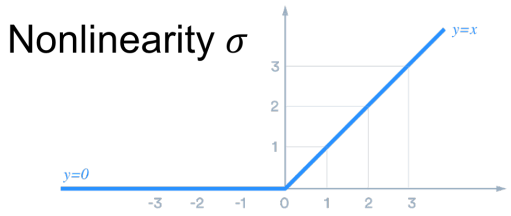


- Evaluating $\mathcal{N}(x)$ requires $|x| \cdot |\text{layer 1}| + |\text{layer 1}| \cdot |\text{layer 2}| + \ldots$ multiplications if fully connected.
- Can be expensive, especially on constrained devices like cellphones, cameras, etc.
- For approximate evaluation, suffices to identify the neurons in each layer with high activation when x is presented

HASHING FOR NEURAL NETWORKS



...



$$n_i = \sigma\left(\sum_{j=1}^{m} w(x_j, n_i) \cdot x_j\right) = \sigma(\langle w_i, x \rangle)$$

- Important neurons have high activation $\sigma(\langle w_i, x \rangle)$.
- Since σ is typically monotonic, this means large $\langle w_i, x \rangle$.
- $\cos(\theta(w_i, x)) = \frac{\langle w_i, x \rangle}{\|w_i\| \|x\|}$. Thus these neurons can be found very quickly using LSH for cosine similarity search.
- Store each weight vector w_i (corresponding to each node) in a set of hash tables and check inputs x for similarity to these stored vectors.