

DS 503: Advanced Data Analytics

Week 3: Locality Sensitive Hashing

Finding similar sets

- Find near-neighbors in high-dimensional space
- **Pages with similar words**
 - For duplicate detection, classification by topic
- **Customers who purchased similar products**
 - Products with similar customer sets
- **Images with similar features**
 - Image completion
- **Recommendations and search**
- **Entity Resolution (3.8.1 in MMDS)**
- **Matching Fingerprints (3.8.4 in MMDS)**
- **Matching Newspaper articles (3.8.6 in MMDS)**

$$A^* =$$



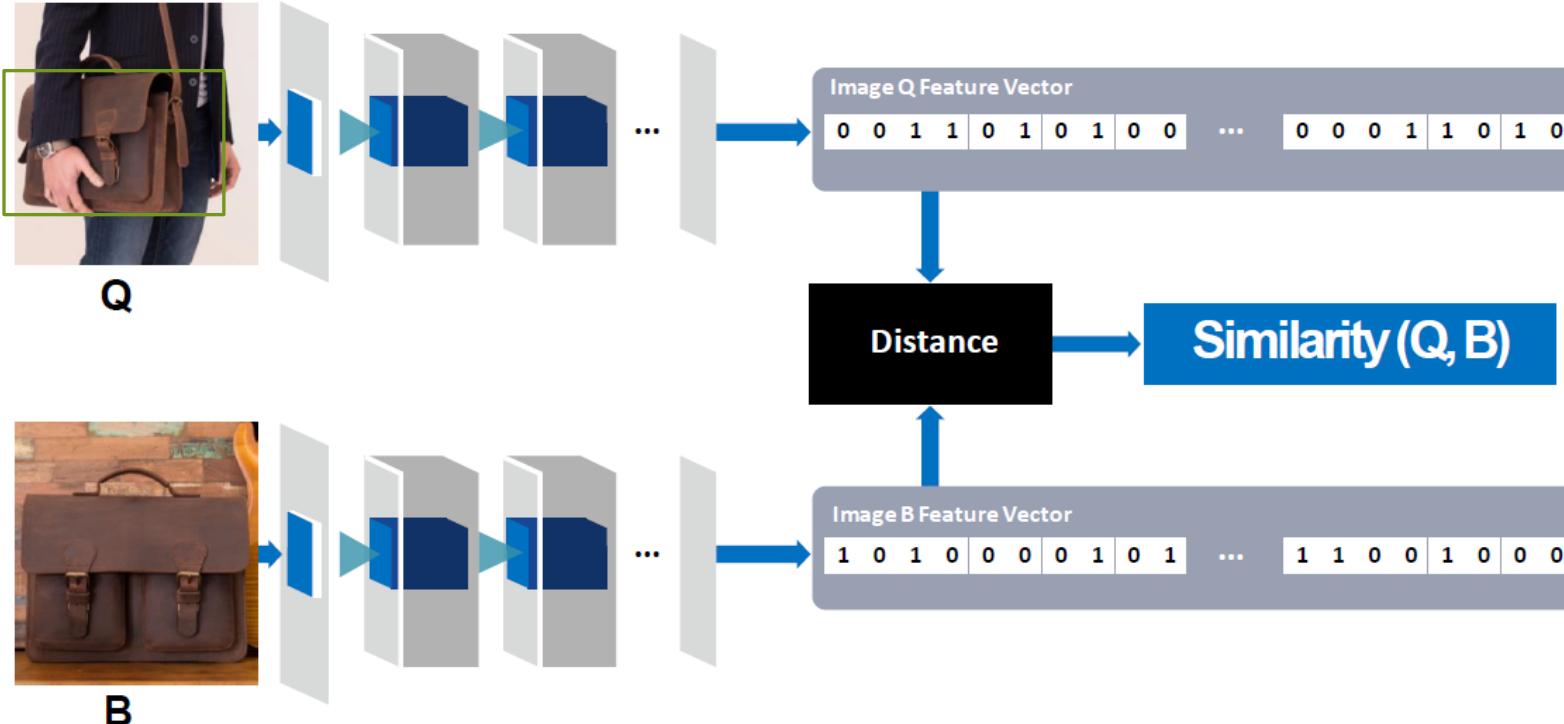
Image Completion
Problem

Can also be solved
by sparse solutions
with low rank constraint

$$\min \|A\|_0$$

s.t. A agrees with A^* in the known entries.

Finding similar objects/items



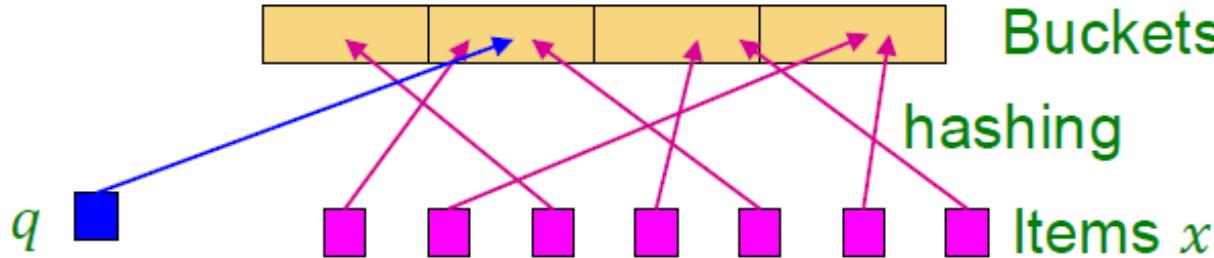
○ Pinterest Visual Search

- Given a query image patch (Q), find similar images among the collection
- Determine a feature vector and find nearest neighbors

Problem Statement

- Given, high dimensional points $x_1, x_2 \dots x_n$
 - Example, images are long vectors of pixels
- And some distance function $d(x_1, x_2)$
 - Which quantifies distance between x_1 and x_2
- Goal 1: Given a query point q , find all x_j that are within a distance threshold $d(q, x_j) \leq s$
 - Can it be done in $O(1)$?
- Goal 2: Find all pairs of data points x_i, x_j that are within a distance threshold $d(x_i, x_j) \leq s$
 - Can it be done in $O(n)$?

LSH Algorithm (High Level Idea)



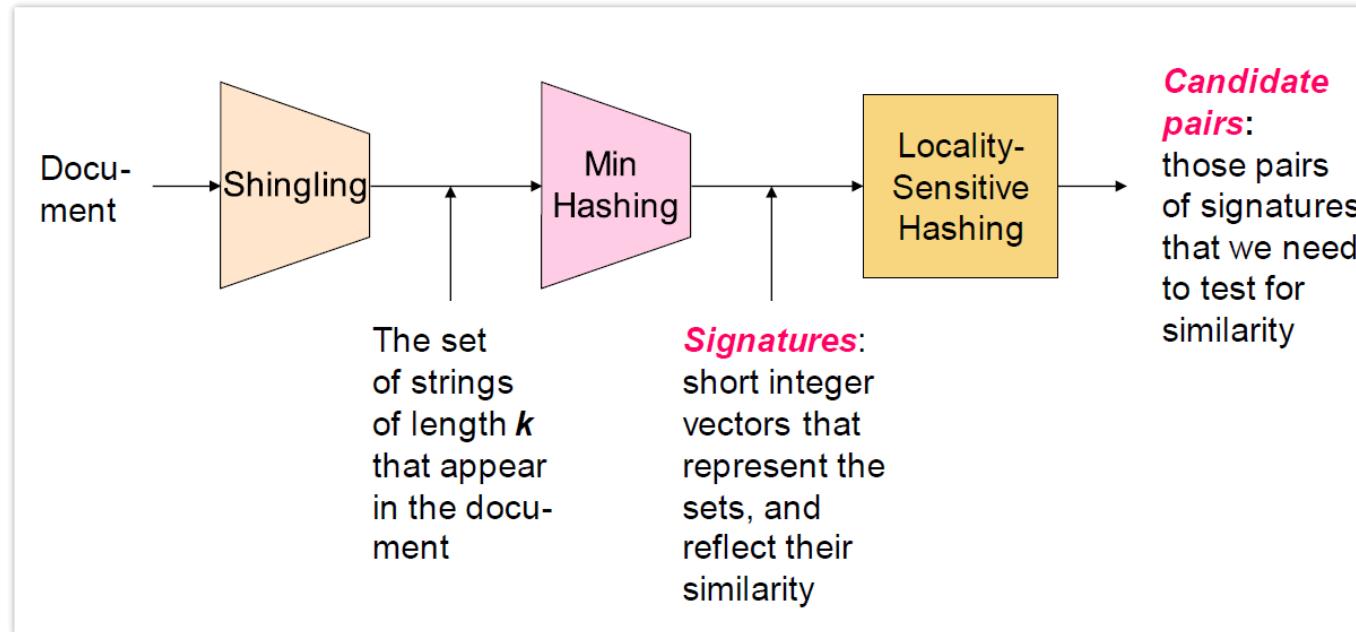
- LSH is a family of related techniques
- In general, one uses several hash functions to throw items into buckets
- Examine only those items that share a bucket for at least one hash function
- **Advantage:** With good design, only a small fraction of points are examined
- **Downside:** Some similar items may get missed (false negatives)

Motivating Example

- Near Duplicate Document in a Large Set
 - Suppose N=1 million
 - Pair-wise similarity = $5 * 10^{11}$ comparisons
 - At 10^5 seconds a day, 10^6 comparisons a sec
 - It would take 5 days
 - For N=10 million
 - It would take a year!
- Similarly, if we have a dataset of 1 B documents, finding the document which is most similar to query document q

$$\text{Pairs} = \frac{n(n-1)}{2}$$

Document Similarity (3-step process)



- Shingling produces a set representation of the document (Boolean vector)
- Min-hashing converts large sets to short signatures while preserving similarity
- LSH focuses on the pair of signatures that are most likely from similar documents

Step 1: Shingling

- Converts a document into a set
- A k-shingle (or k-gram) is a sequence of k tokens that appear in the document
 - Tokens: characters, words, or something else
- To compress shingles, we can hash them (4 bytes)
- Represent the document by the set of hash values of its shingles

- **Example:** k=3; Document $D_1 = abdgabdg$ a
- Set of 3-shingles: {abd, bdg, dga, gab}
- Hash the shingles: $h(D_1) = \{1,3,9,11\}$
- K= 8, 9, 10 is often used in practice

Benefits

- Documents that are intuitively similar will have many shingles in common
- Changing a word only affects k-shingles within distance k-1 from the word

Similarity Metric

- Document D_i is represented by a set of its k-shingles
 $C_i = S(D_i)$

- A natural similarity measure is the Jaccard similarity:

- $Sim(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$

- Jaccard distance:

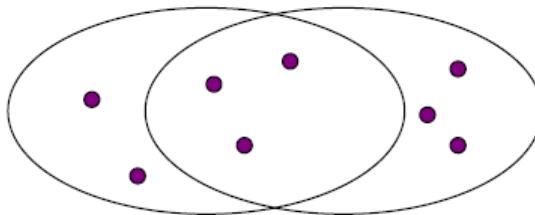
- $d(C_1, C_2) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$

- Sets: Jaccard Similarity of A = {a,b,c,d,e} and B={c,b,f,h} is = 2/7

- Bags: $\frac{\sum(\min \text{ frequency of common items})}{|C_1| + |C_2|}$

- Jaccard Similarity of A = {a,a,a,b} and B={a,a,b,b,c} is = 3/9=1/3

- **Bonus Question (QQ5.1, MMDS 3.1.3):** Suppose we have a universal set U of n elements, and we choose two subsets S and T at random (no repetitions), each with m of the n elements. What is the expected value of the Jaccard similarity of S and T?



3 in intersection.
8 in union.
Jaccard similarity
= 3/8

Representing Documents

Boolean Matrix Representation

- Rows: elements (shingles)
- Columns : sets (Documents)
 - 1 in row e and column s if and only if e is a member of S
 - Column similarity = Jaccard similarity of sets
- Each document is a column

Shingles	Documents	1	1	1	0
	1	1	0	1	
	0	1	0	1	
	0	0	0	1	
	1	0	0	1	
	1	1	1	0	
	1	0	1	0	

We don't really construct the matrix; just imagine it exists

Min-hashing: Convert large sets to small signatures

- Important: While converting sets to small signatures, we need to preserve similarity
- Key idea: “hash” each column C to a small signature $h(C)$ such that:
 - $\text{Sim}(C_1, C_2) = \text{“similarity”}$ of signatures $h(C_1)$ and $h(C_2)$
- Goal: Find a hash function $h(\cdot)$ such that:
 - If $\text{sim}(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
 - If $\text{sim}(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$
- Idea: Hash docs into buckets. Expect that “most” pairs of near duplicate docs hash into the same bucket!
 - Clearly, the hash function depends on the similarity metric
 - For Jaccard similarity, the hash function is called **Min-Hashing**

Min-hashing overview

- Imagine the rows of the boolean matrix permuted under **random permutation π**
- Define a “**minhash**” function for this permutation,
- $h_\pi(C)$ = the index of the **first** (in the permuted order π) row in which column **C** has value **1**:
$$h_\pi(C) = \min_\pi \pi(C)$$
- Apply, to all columns, several randomly chosen permutations π (e.g. 100) to create a **signature** for each column
- **Result is a signature matrix:** Columns = sets, Rows = minhash values for each permutation π

Min-hashing example

Input matrix
(Shingles x Documents)

1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	0	1
5	0	1	0	1
6	1	0	1	0
7	1	0	1	0

Permutation π

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

$$h_{\pi}(C) = \min_{\pi} \pi(C)$$

Signature matrix M

2	1	2	1
---	---	---	---

new

OR

1	5	1	5
---	---	---	---

old

Min-hashing example

Input matrix
(Shingles x Documents)

1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	0	1
5	0	1	0	1
6	1	0	1	0
7	1	0	1	0

Permutation π

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

$$h_{\pi}(C) = \min_{\pi} \pi(C)$$

Signature matrix M

2	1	2	1
2	1	4	1

Min-hashing example

Input matrix
(Shingles x Documents)

1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	0	1
5	0	1	0	1
6	1	0	1	0
7	1	0	1	0

Permutation π

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

$$h_{\pi}(C) = \min_{\pi} \pi(C)$$

Signature matrix M

2	1	2	1
2	1	4	1
1	2	1	2

Min-Hash Property

- Choose a random permutation π
- Claim: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- Why?
 - Let X be a doc (set of shingles), $y \in X$ is a shingle
 - Then: $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
 - It is equally likely that any $y \in X$ is mapped to the **min** element
 - Let y be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
 - Then either: $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, or
 $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$
 - So the prob. that **both** are true is the prob. $y \in C_1 \cap C_2$
 - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = \text{sim}(C_1, C_2)$

G	C_2
0	0
0	0
1	1
0	0
0	1
1	0

Similarity of Signatures

- We know: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions
- The *similarity of two signatures* is the fraction of the hash functions in which they agree
- Note: Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures
- To get good estimates, we must use a lot of Min-Hash functions

Min-hashing Example

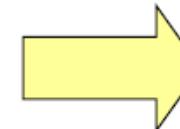
Permutation π Input matrix (Shingles x Documents)

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

	C_1	C_2	C_3	C_4
1	0	1	0	
1	0	0	1	
0	1	0	1	
0	1	0	1	
0	1	0	1	
1	0	1	0	
1	0	1	0	

Signature matrix M

	M_1	M_2	M_3	M_4
2	1	2	1	
2	1	4	1	
1	2	1	2	



Similarities:

Col/Col
Sig/Sig

1-3	2-4	1-2	3-4
0.75	0.75	0	0
0.67	1.00	0	0

Implementation Trick

- Permuting rows even once is prohibitive

- Row hashing!

- Pick $K = 100$ hash functions h_i ,
 - Ordering under h_i gives a random row permutation!

- One-pass implementation

{ Looks at each element of input matrix }

- For each column C and hash-func. h_i , keep a “slot” $M(i, c)$ for the min-hash value

- Initialize all $M(i, c) = \infty$

- Scan rows looking for 1s

- Suppose row j has 1 in column C

- Then for each h_i :

- If $h_i(j) < M(i, c)$, then $M(i, c) \leftarrow h_i(j)$

How to pick a random hash function $h(x)$?

Universal hashing:

$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:

a, b ... random integers

p ... prime number ($p > N$)

Example Code

```
for each row  $r$  do begin  
    for each hash function  $h_i$  do  
        compute  $h_i(r)$ ;  
    for each column  $c$   
        if  $c$  has 1 in row  $r$   
            for each hash function  $h_i$  do  
                if  $h_i(r) < M(i, c)$  then  
                     $M(i, c) := h_i(r);$   
end;
```

Important: so you hash r only once per hash function, not once per 1 in row r .

How to pick a random hash function $h(x)$?

Universal hashing:

$$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$$

where:

$a, b \dots$ random integers

$p \dots$ prime number ($p > N$)

Example run

permutation				
$x=1$	$h(x) \ g(x)$	Row	C_1	C_2
1	1 3	1	1	0
2	2 0	2	0	1
3	3 2	3	1	1
4	4 4	4	1	0
5	0 1	5	0	1

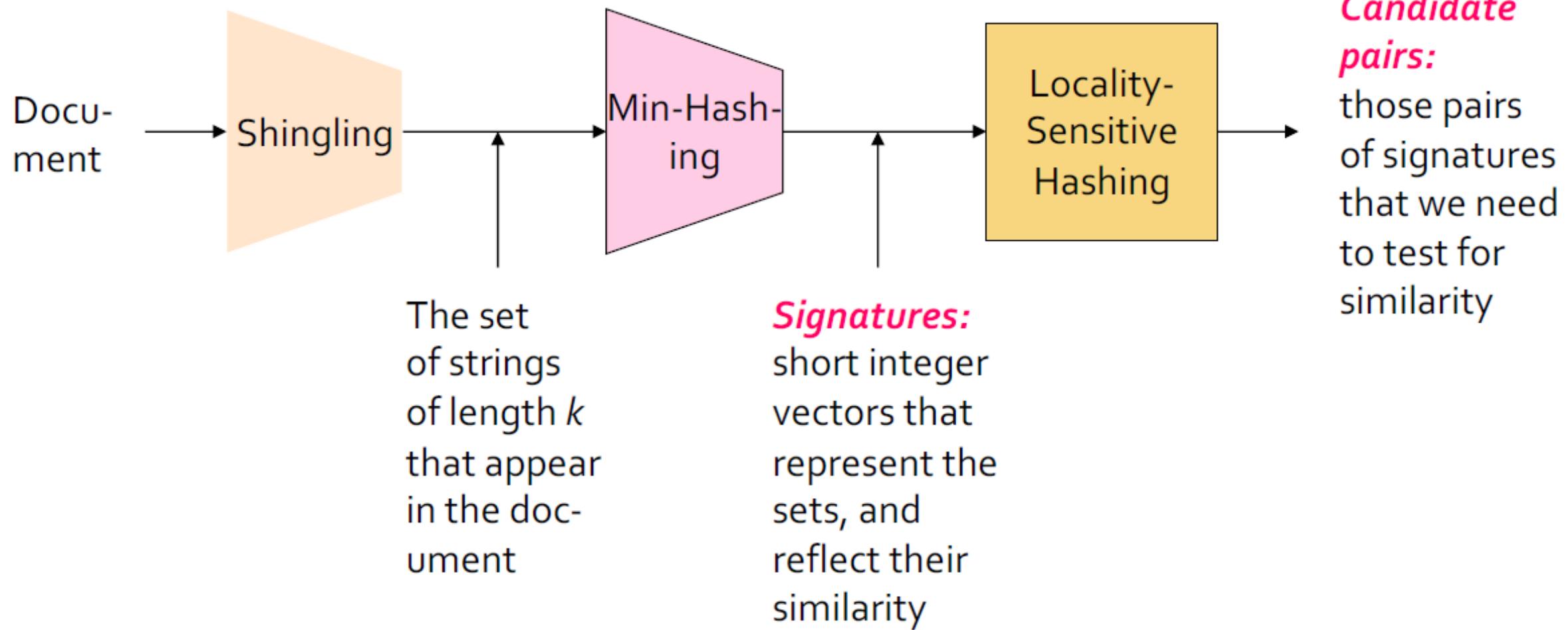
$$\begin{aligned} h(x) &= x \bmod 5 \\ g(x) &= (2x+1) \bmod 5 \end{aligned}$$

	$M(i, C_1)$	$M(i, C_2)$
$h(1) = 1$	1	∞
$g(1) = 3$	3	∞
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

1	0
2	0

Signature matrix M

Locality Sensitive Hashing



Key Idea of LSH

- **Goal:** Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s=0.8$)
- **LSH – General idea:** Use a function $f(x,y)$ that tells whether x and y is a ***candidate pair***: a pair of elements whose similarity must be evaluated
- **For Min-Hash matrices:**
 - Hash columns of **signature matrix M** to many buckets
 - Each pair of documents that hashes into the same bucket is a ***candidate pair***

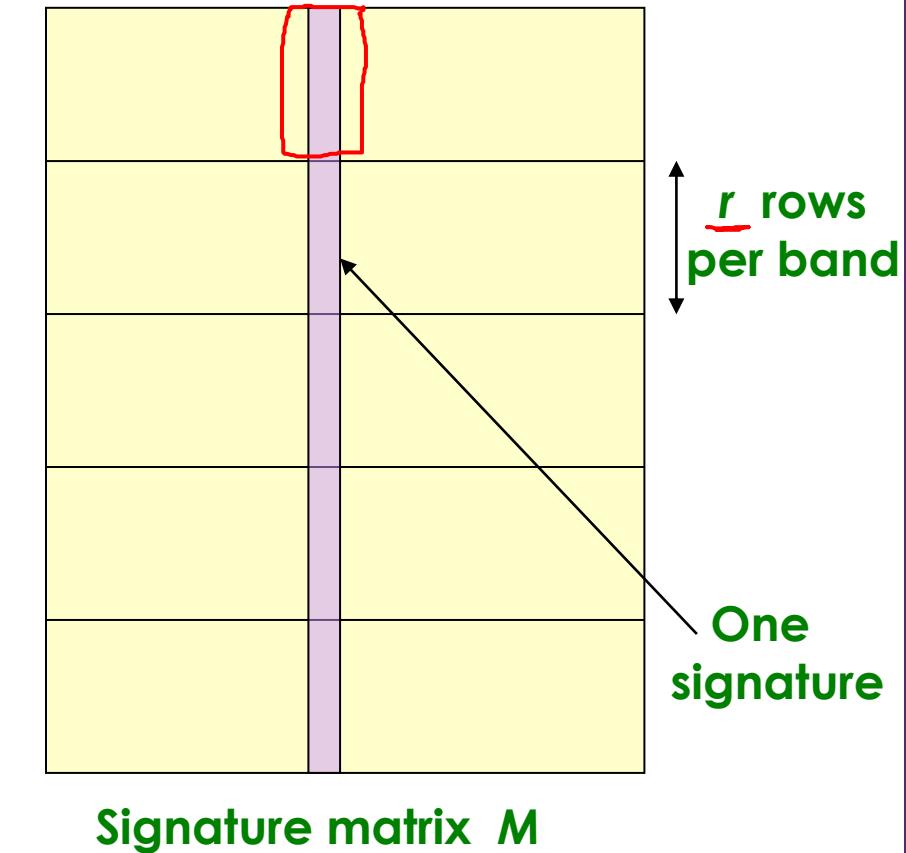
2	1	4	1
1	2	1	2
2	1	2	1

Picking Good Candidates

- Pick a similarity threshold s ($0 < s < 1$)
- Columns x and y of M are a **candidate pair** if their signatures agree on at least fraction s of their rows:
 $M(i, x) = M(i, y)$ for at least frac. s values of i
 - We expect documents x and y to have the same (Jaccard) similarity as their signatures
- **Big idea: Hash columns of signature matrix M several times**
- Arrange that (only) similar columns are likely to **hash to the same bucket**, with high probability
- **Candidate pairs are those that hash to the same bucket**

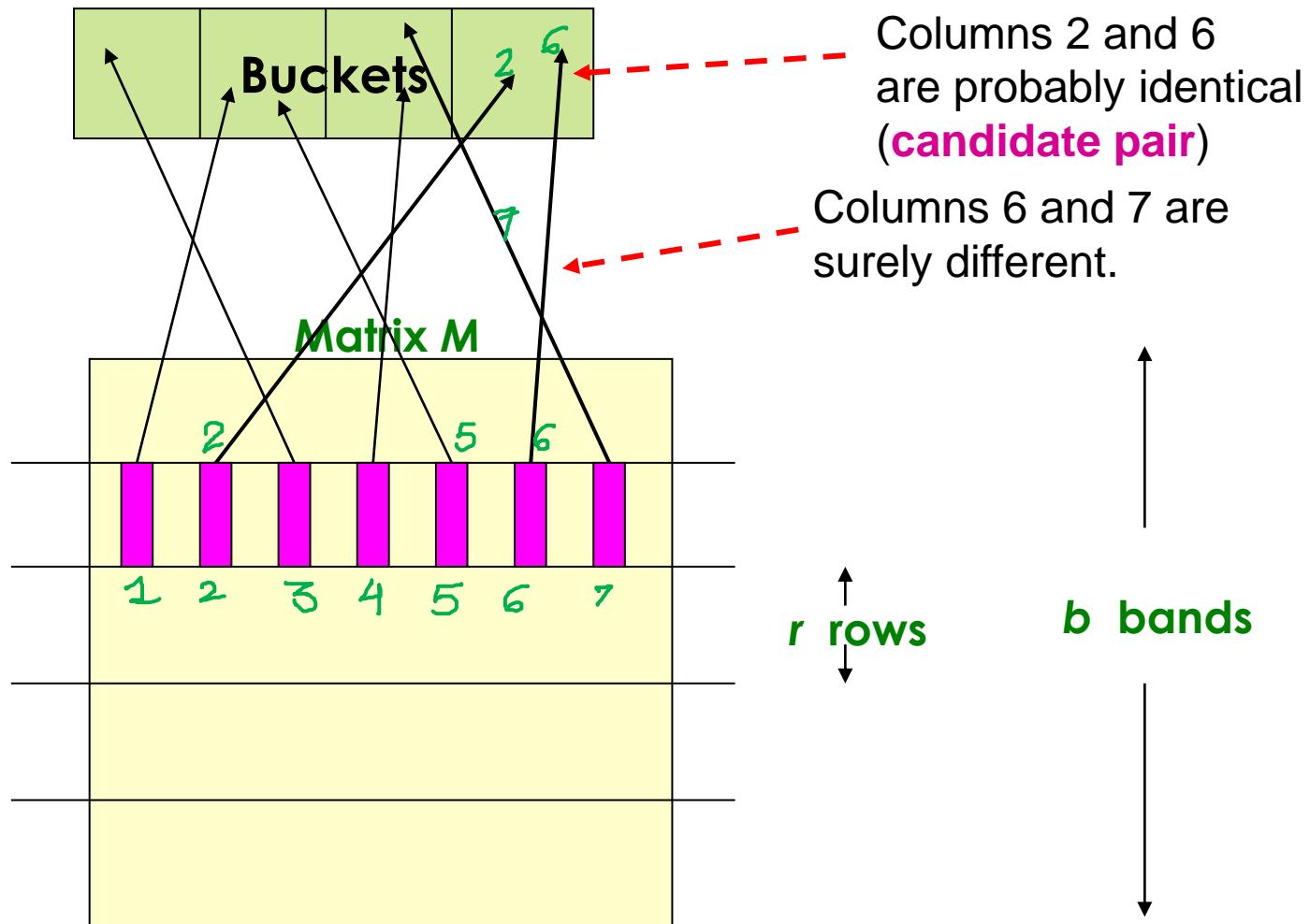
Partition Signature Matrix Into Bands

- Divide matrix M into b bands of r rows
- For each band, hash its portion of each column to a hash table with k buckets b bands
 - Make k as large as possible
- **Candidate** column pairs are those that hash to the same bucket for ≥ 1 band
- Tune b and r to catch most similar pairs, but few non-similar pairs



Hashing Bands into Buckets

- There are **enough buckets** that columns are unlikely to hash to the same bucket unless they are **identical** in a particular band
- Hereafter, we assume that “**same bucket**” means “**identical in that band**”
- Assumption needed only to simplify analysis, not for correctness of algorithm



Analysis for a simple example

Assume the following case:

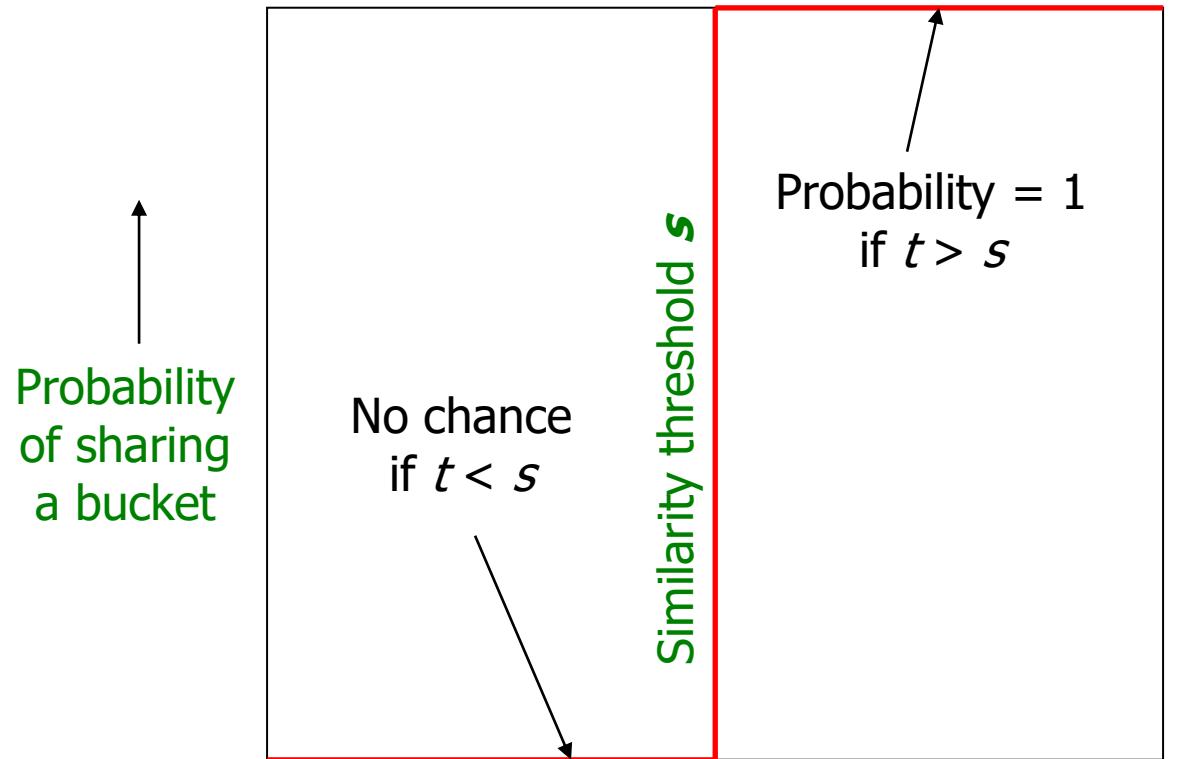
- Suppose 100,000 columns of M (100k docs)
 - Signatures of 100 integers (rows)
 - Therefore, signatures take 40Mb
 - Choose $b = 20$ bands of $r = 5$ integers/band
 - **Goal:** Find pairs of documents that are at least $s = 0.8$ similar
 - Find pairs of $\geq s=0.8$ similarity, set $b=20, r=5$
 - **Assume:** $\text{sim}(C_1, C_2) = 0.8$
 - Since $\text{sim}(C_1, C_2) \geq s$, we want C_1, C_2 to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)
 - **Probability C_1, C_2 identical in one particular band:** $(0.8)^5 = 0.328$
 - Probability C_1, C_2 are **not** similar in all of the 20 bands: $(1-0.328)^{20} = 0.00035$
 - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
 - **We would find 99.965% pairs of truly similar documents**
- $\{ P(\text{true is similar}) = 0.8 \}$

Eliminating False Positives

- Find pairs of $\geq s=0.8$ similarity, set $b=20$, $r=5$
- Assume: $\text{sim}(C_1, C_2) = 0.3$
 - Since $\text{sim}(C_1, C_2) < s$ we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$
- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)^{20} = 0.0474$
 - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming **candidate pairs**
 - They are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s

LSH Trade-off

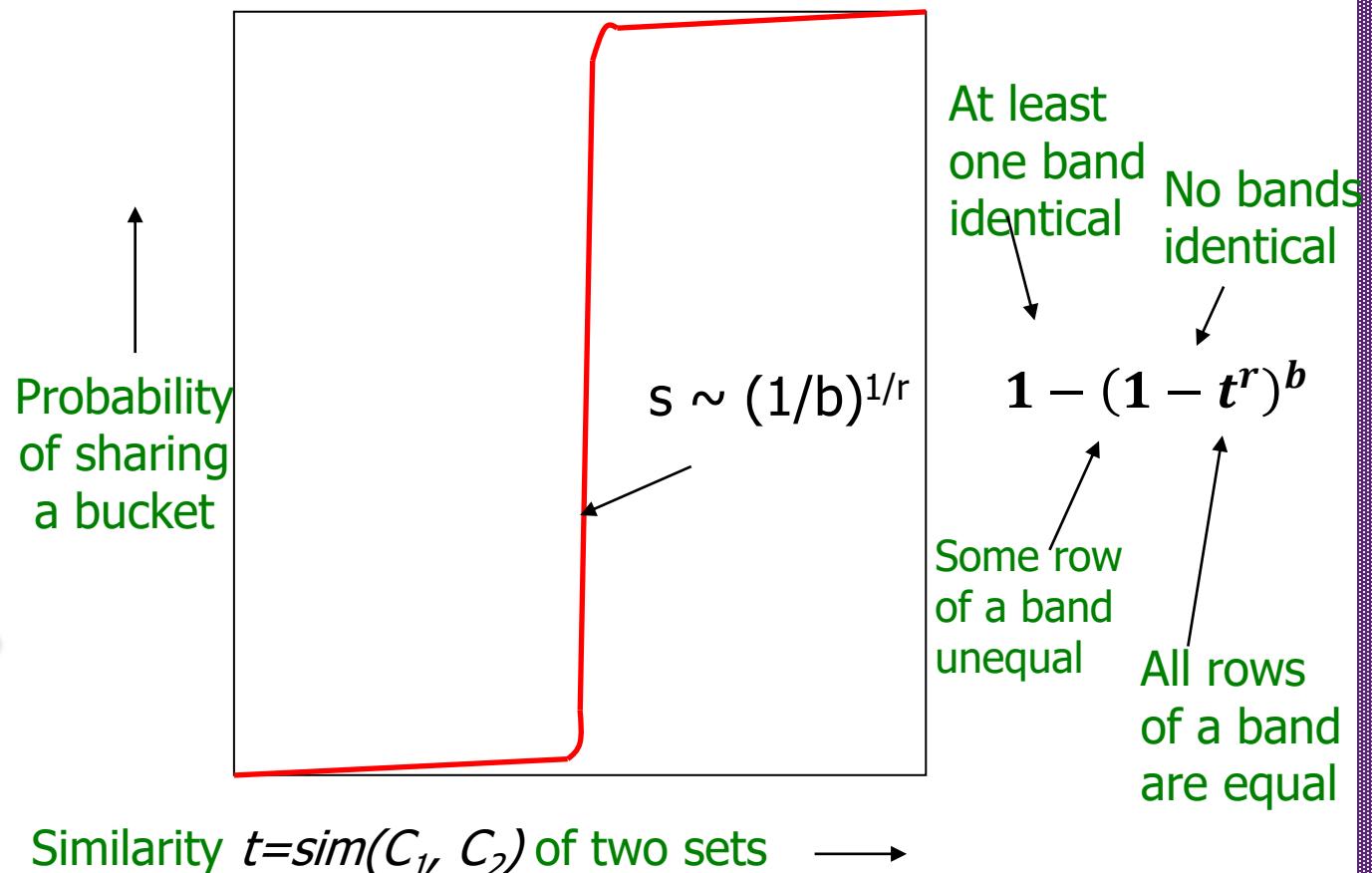
- **Pick:**
 - The number of Min-Hashes (rows of M)
 - The number of bands b , and
 - The number of rows r per band to balance false positives/negatives ($M=b \cdot r$)
- **Example:** If we had only 10 bands of 10 rows, the number of false positives would go down, but the number of false negatives would go up (it's harder to become a candidate pair in the bucket now)



What we would like to get?

LSH Design Parameters (b bands and r rows/band)

- Columns C_1 and C_2 have similarity t
- Pick any band (r rows)
 - Prob. that all rows in band equal = t^r
 - Prob. that some row in band unequal = $1 - t^r$
- Prob. that no band identical = $(1 - t^r)^b$
- Prob. that at least 1 band identical = $1 - (1 - t^r)^b$
-



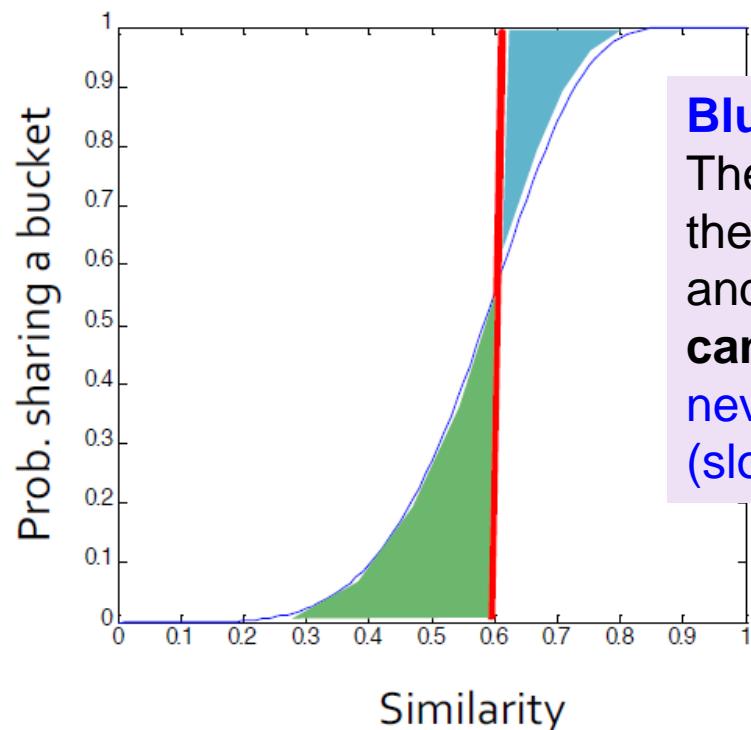
Example Calculation ($r=5$, $b=10$)

- Similarity threshold s
- Prob. that at least 1 band is identical:

s	$1-(1-s^r)^b$
.2	0.003
.3	0.024
.4	0.098
.5	0.272
.6	0.555
.7	0.841
.8	0.981

Picking r and b to get the best S-curve

50 hash-functions ($r=5$, $b=10$)



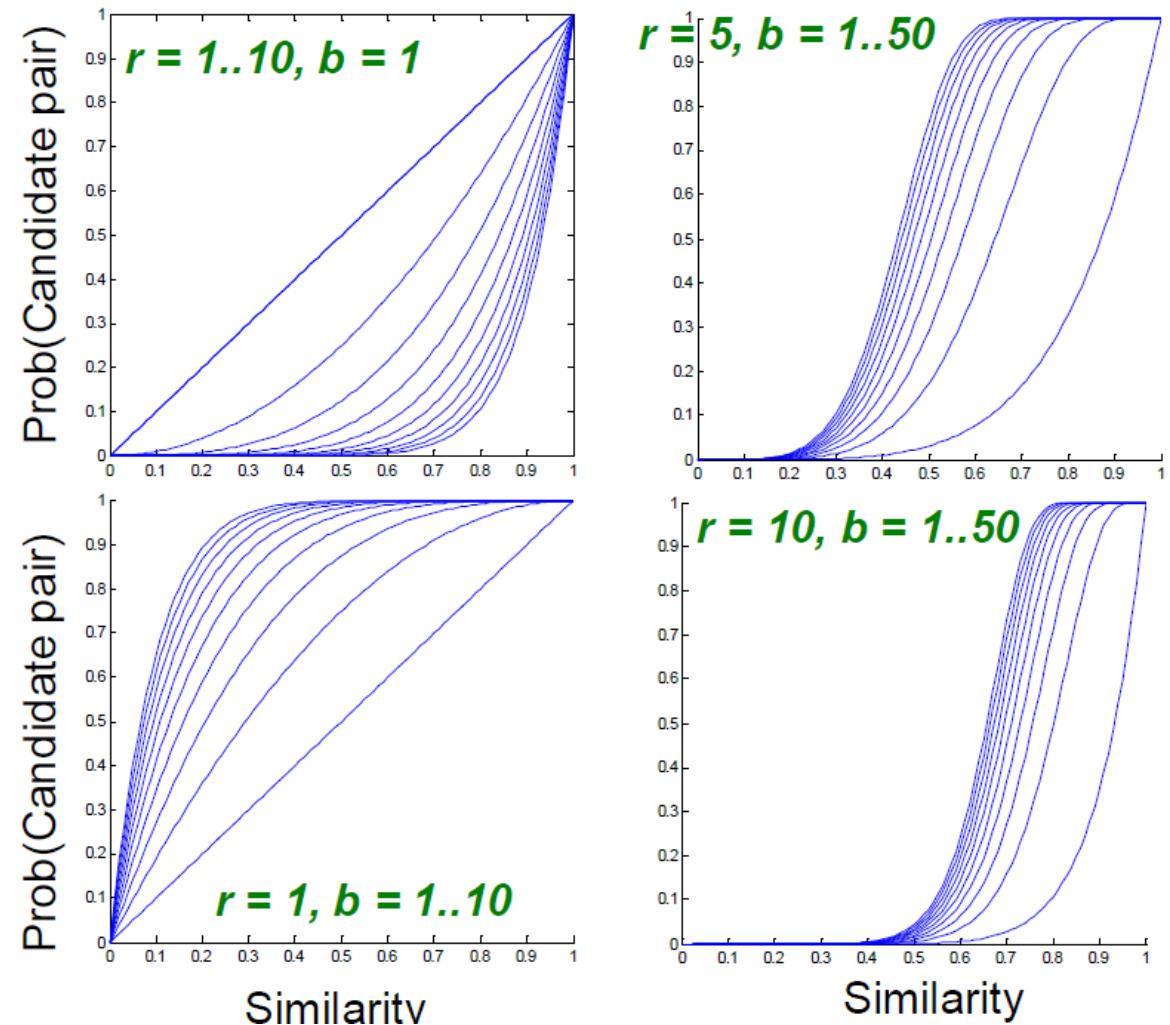
Blue area X: False Negative rate
These are pairs with $sim > s$ but the **X** fraction won't share a band and then will **never become candidates**. This means we will never consider these pairs for (slow/exact) similarity calculation!

Green area Y: False Positive rate

These are pairs with $sim < s$ but we will consider them as candidates. This is not too bad, we will consider them for (slow/exact) similarity computation and discard them.

S curves as a function of b and r

- Given a fixed threshold t in the query
- We want choose r and b such that the $P(\text{Candidate pair})$ has a “step” right around t .
- Visualization of the effect of threshold, band size, and # of rows in LSH
- <https://www.desmos.com/calculator/lzzvfjiujn>



Conclusion

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures
 - Check in main memory that **candidate pairs** really do have **similar signatures**
 - **Optional:** In another pass through data, check that the remaining candidate pairs really represent similar documents
- 3 Step Process:
- **Shingling:** Convert documents to sets
 - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
 - We used **similarity preserving hashing** to generate signatures with property $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
 - We used hashing to get around generating random permutations
- **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
 - We used hashing to find **candidate pairs** of similarity $\geq s$

Generalization of distance functions

- We have used LSH to find similar documents
- More generally, we found similar columns in large sparse matrices with high Jaccard similarity
- Can we use LSH for other distance measures?
- e.g., Euclidean distances, Cosine distance
- Let's generalize what we've learned!

Distance Measures

- $d(\cdot)$ is a **distance measure** if it is a function from pairs of points \mathbf{x}, \mathbf{y} to real numbers s.t.:
 - $d(x,y) \geq 0$
 - $d(x,y) = 0$ iff $x = y$
 - $d(x,y) = d(y,x)$
 - $d(x,y) \leq d(x,z) + d(z,y)$ (triangle inequality)
- Jaccard distance for sets = $1 - \text{Jaccard similarity}$
- Cosine distance for vectors = angle between the vectors
- Euclidean distances:
 - *L2 norm*: $d(\mathbf{x}, \mathbf{y})$ = square root of the sum of the squares of the differences between \mathbf{x} and \mathbf{y} in each dimension
 - The most common notion of “distance”
 - *L1 norm*: sum of absolute value of the differences in each dimension
 - *Manhattan distance* = distance if you travel along axes only

Locally Sensitive (LS) Families of Hash Function

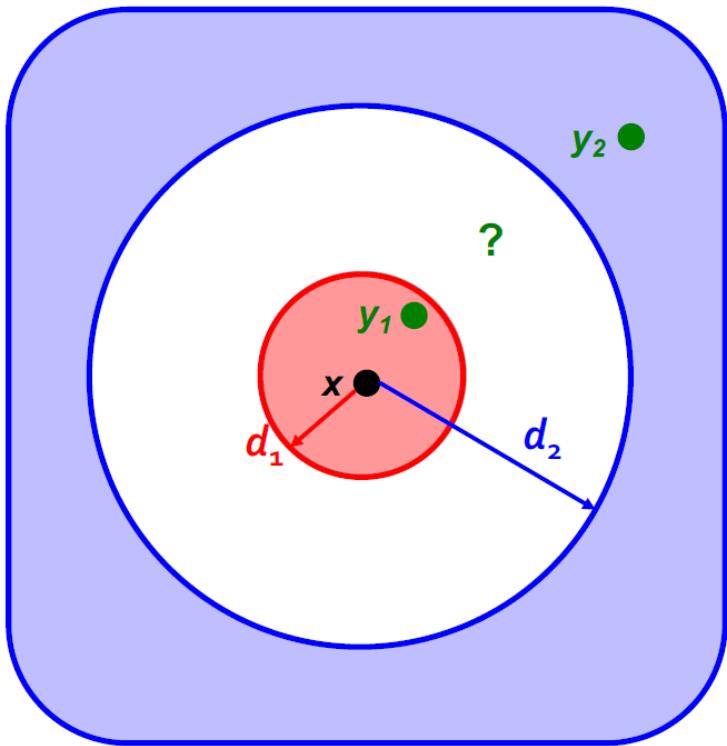
- For Min-Hashing signatures, we got a Min-Hash function for each permutation of rows
- A “hash function” is any function that allows us to say whether two elements are “equal”
- Shorthand: $h(x) = h(y)$ means “*h says x and y are equal w.h.p.*”
- A *family* of hash functions is any set of hash functions from which we can *efficiently pick one at random*
- Example: The set of Min-Hash functions generated from permutations of rows

Suppose we have a space S of points with a distance measure $d(x,y)$
A family H of hash functions is said to be **(d_1, d_2, p_1, p_2) -sensitive** if for any x and y in S :

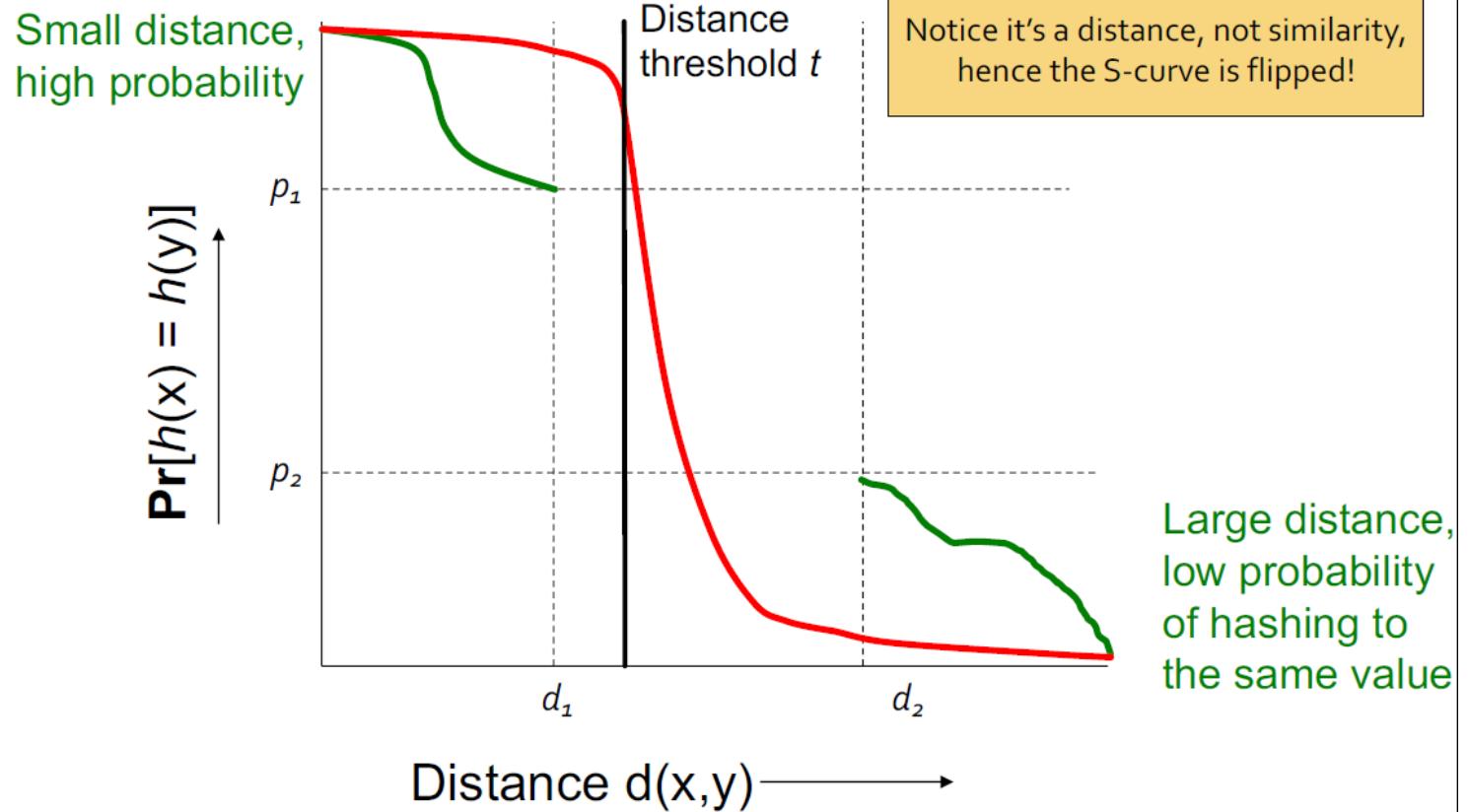
1. If $d(x, y) < d_1$, then the probability over all $h \in H$, that $h(x) = h(y)$ is at least p_1
2. If $d(x, y) > d_2$, then the probability over all $h \in H$, that $h(x) = h(y)$ is at most p_2

We can do LSH with Such Locally Sensitive (LS) Families of Hash Functions

Examples



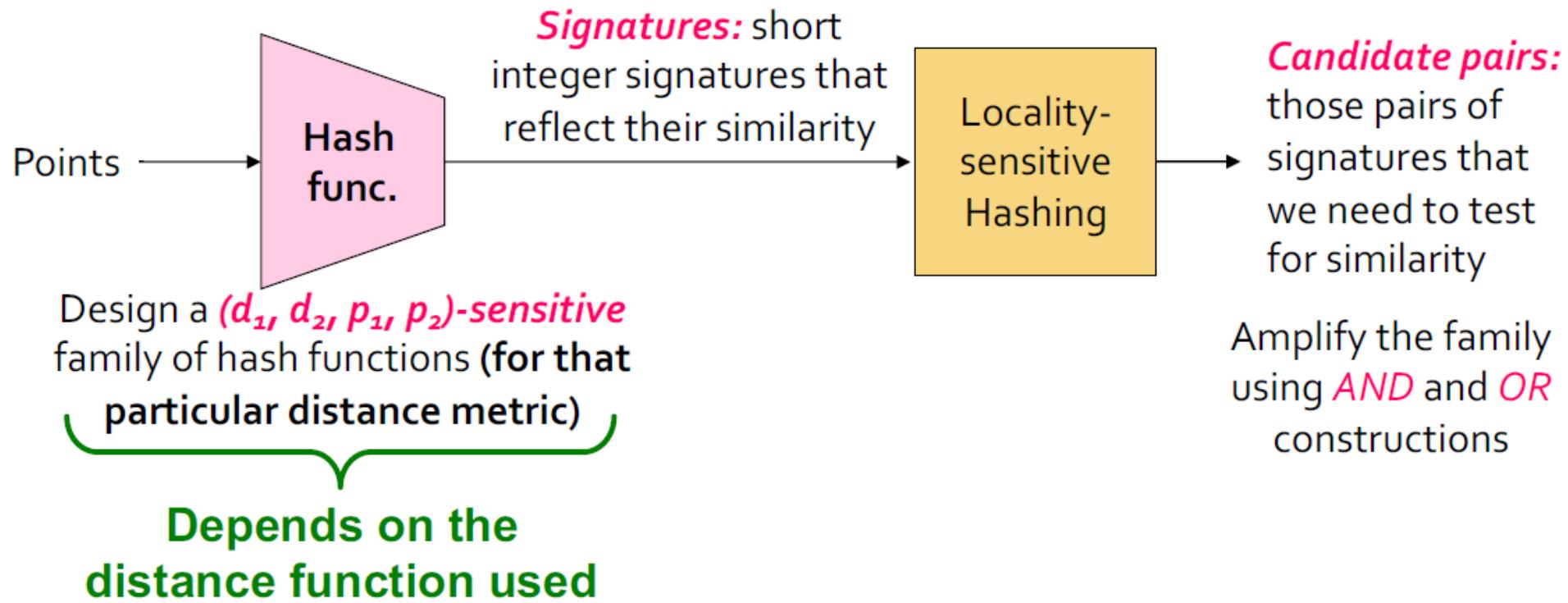
For all $h \in H$,
 $P[h(x) = h(y_1)] \geq p_1$
 $P[h(x) = h(y_2)] \leq p_2$



Example of LS Family: Min-Hash

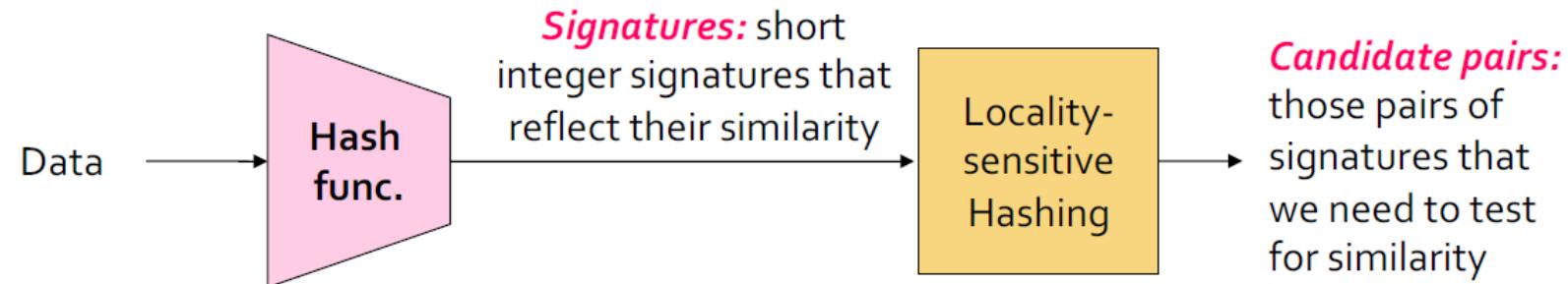
- Let:
 - S = space of all sets,
 - d = Jaccard distance,
 - H is family of Min-Hash functions for all permutations of rows
- Then for any hash function $h \in H$:
 - $\Pr[h(x) = h(y)] = 1 - d(x, y)$
- Restate theorem about Min-Hashing in terms of distances rather than similarities
 - **Claim:** Min-hash H is a $(1/3, 2/3, 2/3, 1/3)$ -sensitive family for S and d .
 - If $distance \leq \frac{1}{3}$ then $similarity \geq \frac{2}{3}$ Then probability that Min-Hash values agree is $\geq \frac{2}{3}$
- **For Jaccard similarity, Min-Hashing gives a $(d_1, d_2, (1-d_1), (1-d_2))$ -sensitive family for any $d_1 < d_2$.**

LSH for other distance functions

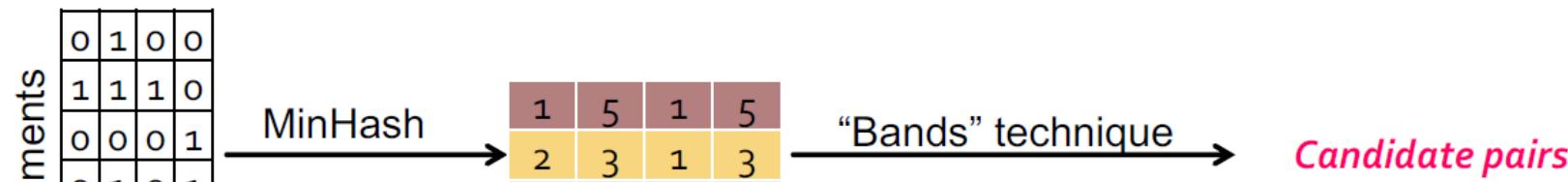


- LSH methods for other distance metrics:
 - Cosine distance: Random hyperplanes
 - Euclidean distance: Project on lines

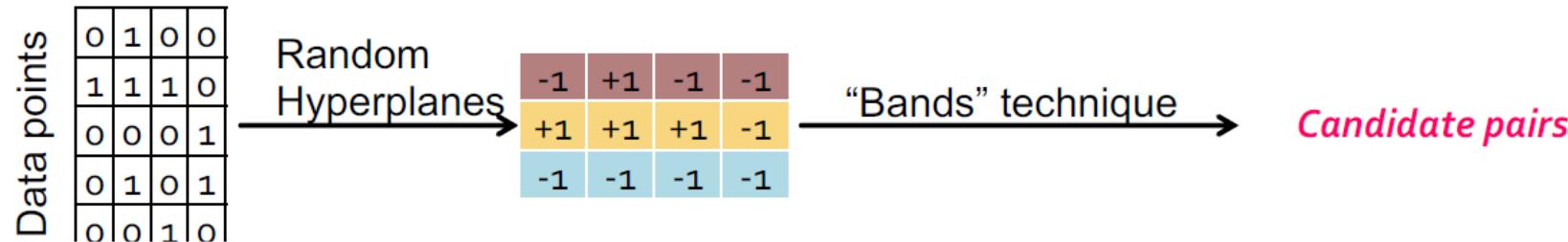
High Level Technique



Text Data



Images



Cosine Distance

Useful metric for
a variety of
datasets

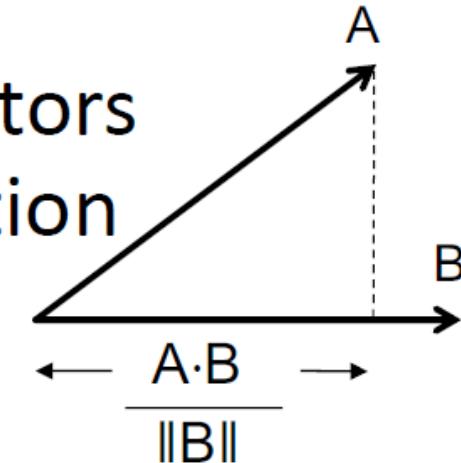
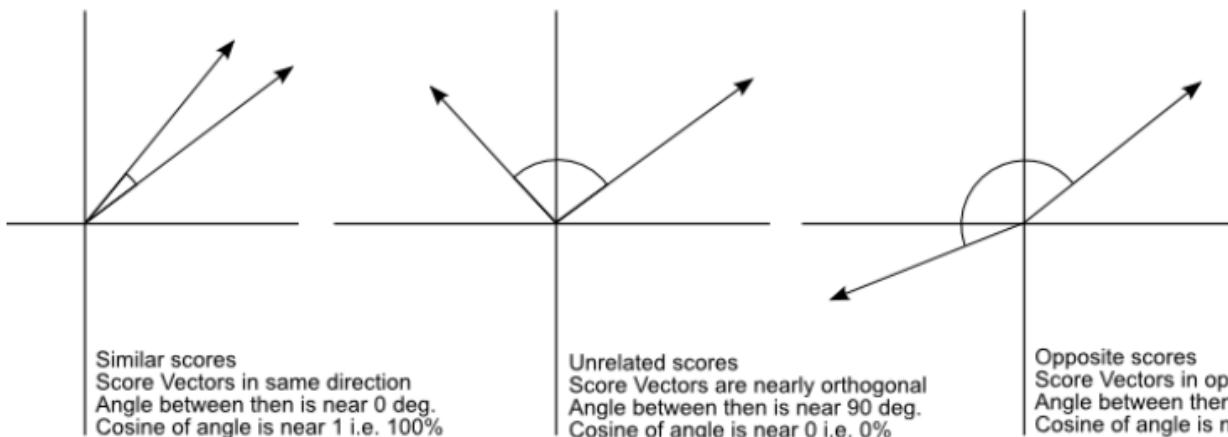
- **Cosine distance** = angle between vectors from the origin to the points in question

$$d(A, B) = \theta = \arccos(A \cdot B / \|A\| \cdot \|B\|)$$

- Has range $[0, \pi]$ (equivalently $[0, 180^\circ]$)
- Can divide θ by π to have distance in range $[0, 1]$

- **Cosine similarity** = $1 - d(A, B)$

- But often defined as **cosine sim**: $\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$

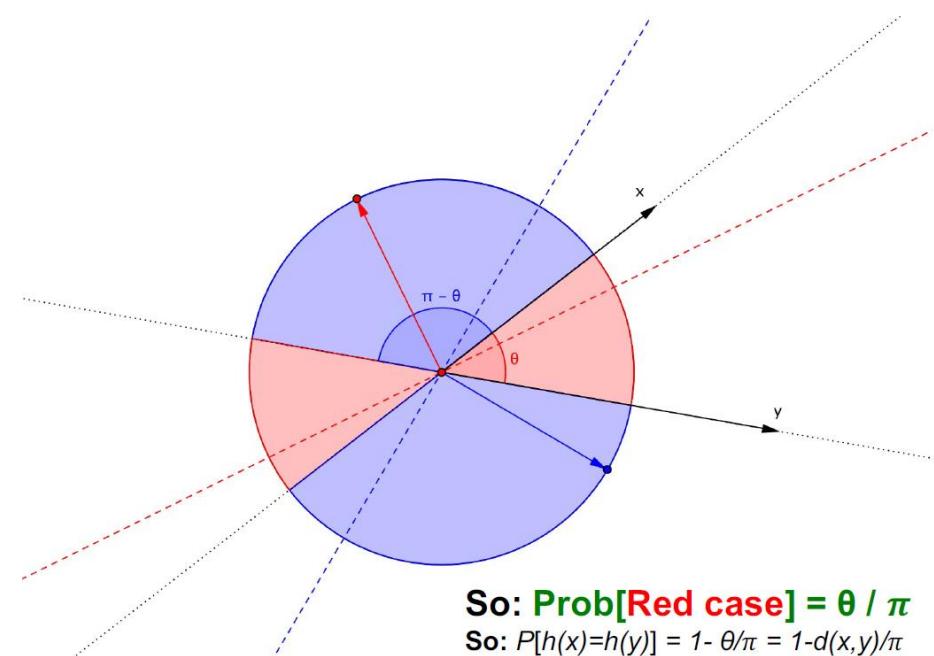
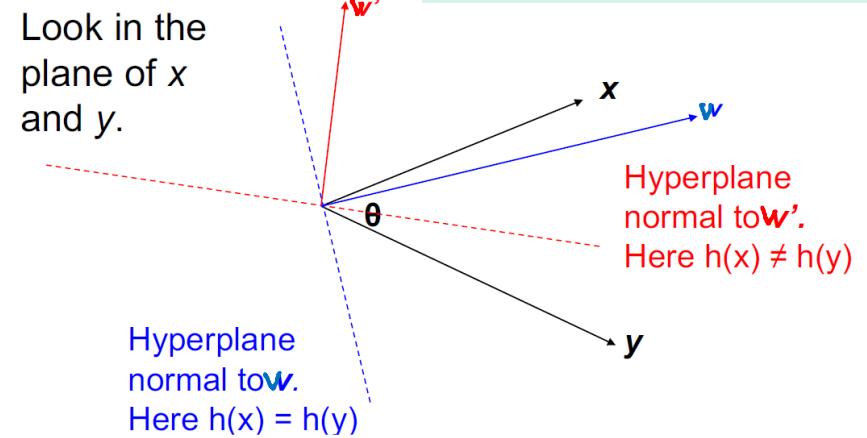


- Has range $-1 \dots 1$ for general vectors
- Range $0..1$ for non-negative vectors (angles up to 90°)

Random Hyperplane Method

- For cosine distance: technique called **Random Hyperplanes**
 - Technique similar to Min-Hashing
- Each vector w determines a hash function h_w with two buckets
 - $h_w(x) = +1 \text{ if } w \cdot x \geq 0 ; = -1 \text{ if } w \cdot x < 0$
 - LS-family H = set of all functions derived from any vector
- **Claim:** For points x and y ,
 - $\Pr[h(x) = h(y)] = 1 - d(x,y) / \pi$
- **Random Hyperplanes** method is a $(d_1, d_2, (1-d_1/\pi), (1-d_2/\pi))$ -sensitive family for any d_1 and d_2
- **Reminder:** (d_1, d_2, p_1, p_2) -sensitive
 - 1. If $d(x,y) < d_1$, then prob. that $h(x) = h(y)$ is at least p_1
 - 2. If $d(x,y) > d_2$, then prob. that $h(x) = h(y)$ is at most p_2

Hyperplanes can be defined using their normal (weight vector, w) and $b=0$

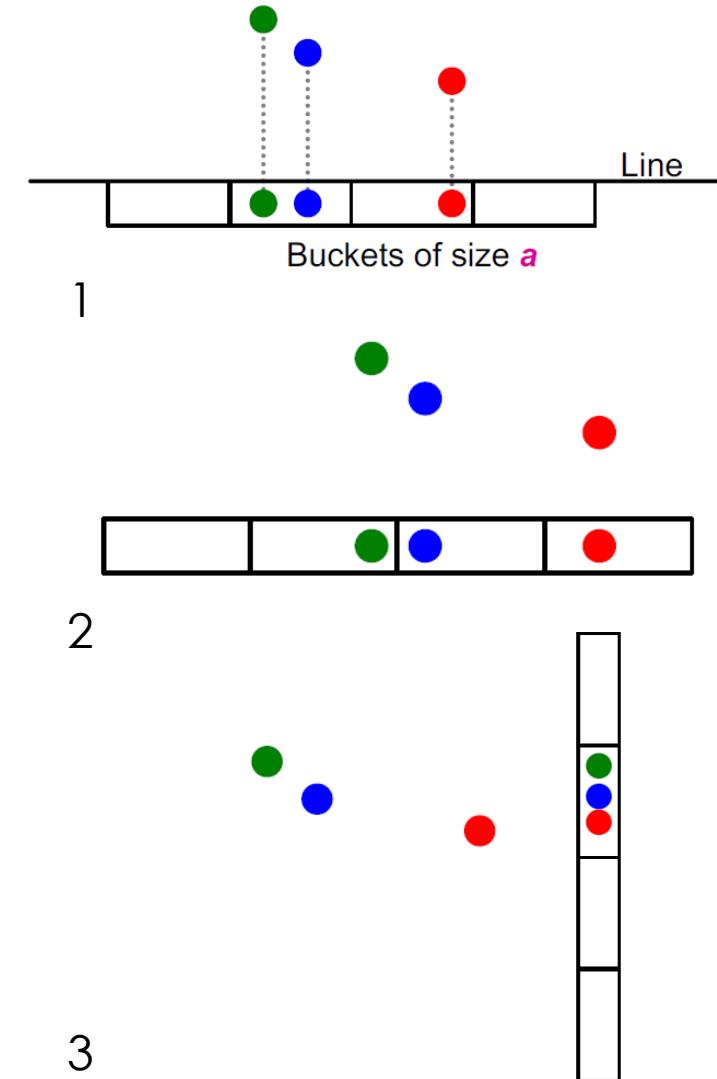


Generating Random Hyperplanes

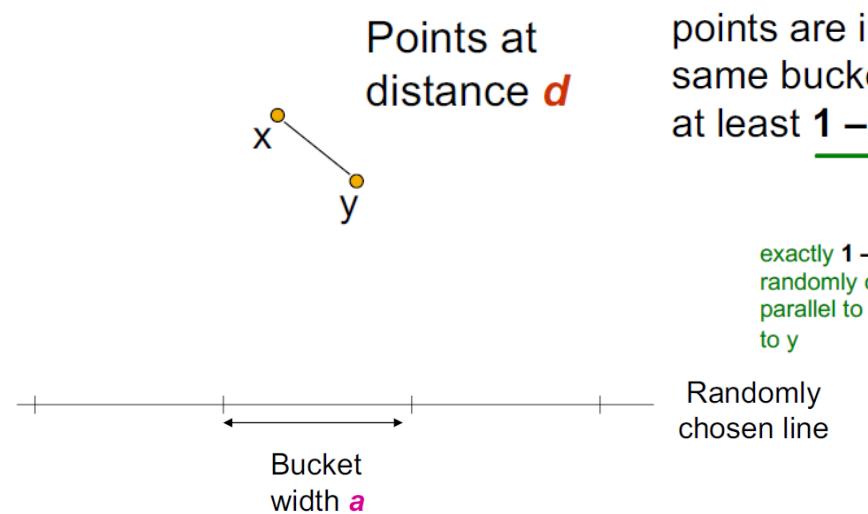
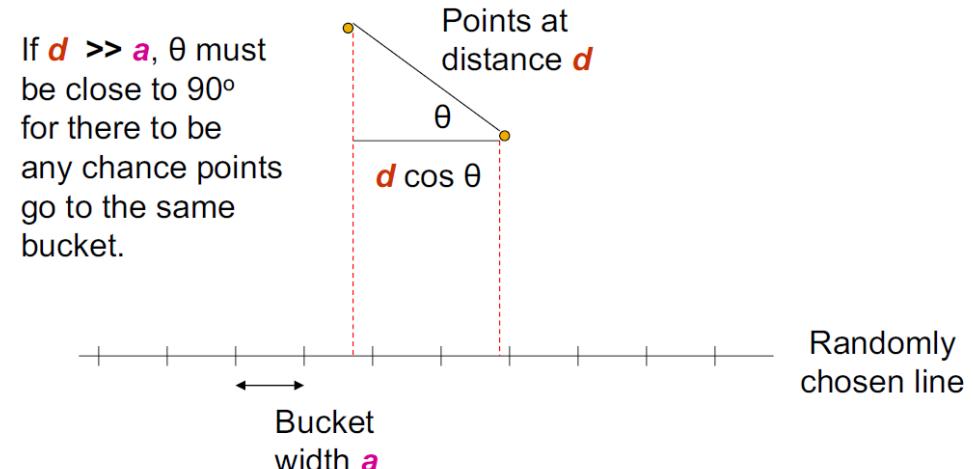
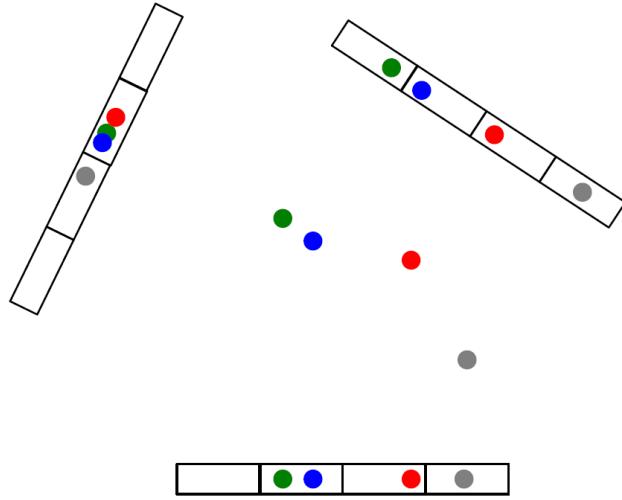
- Expensive to pick a random vector in D dimensions for large D
 - Would have to generate D random numbers
- **A more efficient approach**
 - It suffices to consider only vectors v consisting of +1 and -1 components
- **Why?** Assuming data is random, then vectors of +/-1 cover the entire space evenly (and does not bias in any way)
- Pick some number of random vectors, and hash your data for each vector
 - The result is a **signature (*sketch*)** of +1's and -1's for each data point
- Can be used for LSH like we used the Min-Hash signatures for Jaccard distance
 - Amplify using **AND/OR** constructions (banding method)

LSH for Euclidean distance

- Idea: Hash functions correspond to lines
 - Partition the line into buckets of size a
- Hash each point to the bucket containing its projection onto the line
 - An element of the “Signature” is a bucket id for that given projection line
 - **Nearby points are always close**; distant points are rarely in same bucket
- 1 “Lucky” case:
 - Points that are close hash in the same bucket
 - Distant points end up in different buckets
- Two “unlucky” cases:
 - 2: unlucky quantization
 - 3: unlucky projection



Multiple Projections



If $d \ll a$, then the chance the points are in the same bucket is at least $1 - d/a$.

exactly $1 - d/a$ when the randomly chosen line is parallel to the line from x to y

L_s family for Euclidean distance

- If points are distance $d < a/2$, prob. they are in same bucket $\geq 1 - d/a = 1/2$
- If points are distance $d > 2a$ apart, then they can be in the same bucket only if $d \cos \theta \leq a$
 - $\cos \theta \leq 1/2$
 - $60^\circ < \theta < 90^\circ$, i.e., at most $1/3$ probability
- Yields a **$(a/2, 2a, 1/2, 1/3)$ -sensitive** family of hash functions for any a
 - Amplify using AND-OR cascades