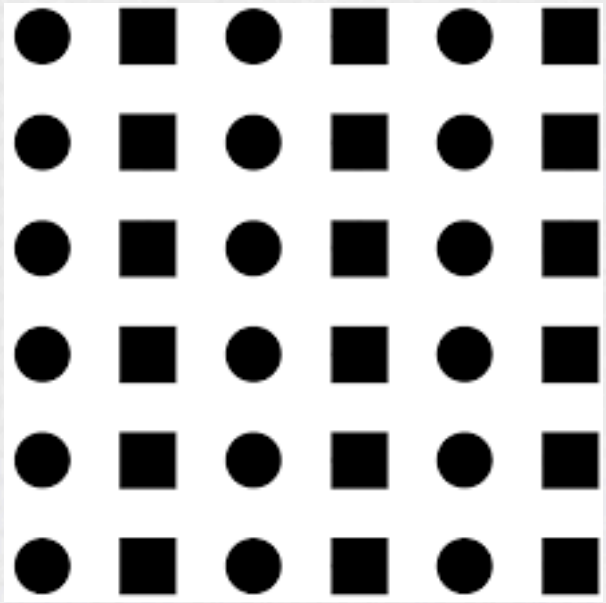
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS



BIG DATA
Analytics & Management

Lecture 9 (04/10, 04/12): Locality Sensitive Hashing

Decisions, Operations & Information Technologies
Robert H. Smith School of Business
Spring, 2017

A Common Metaphor

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Many problems can be expressed as finding "similar" sets.
- Examples:
  - ❑Pages with similar words
  - ❑Customers who purchased similar products
  - ❑Images with similar features
  - ❑Users who visited the similar websites

Finding similar documents

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Goal: Given a large number (millions or billions) of text documents, find pairs that are "near duplicates"
- Applications:
  - Mirror websites, or approximate mirrors
    - Don't want to show both in a search
  - Similar (not topic) news articles at many news sites
    - Cluster articles by "same story"
- Problems:
  - Many small pieces of one document can appear out of order in another (order matters)
  - Too many documents to compare all pairs
  - Documents are so large or so many that they cannot fit in main memory

Some math

- Suppose we need to find near-duplicate documents among N=1 million documents

- Naively, we'd have to compute pairwise similarities for every pair of docs

  i.e, $N(N-1)/2 \approx 5*10^{11}$ comparisons

  At 10^5 secs/day and 10^6 comparisons/sec,
  it would take 5 days

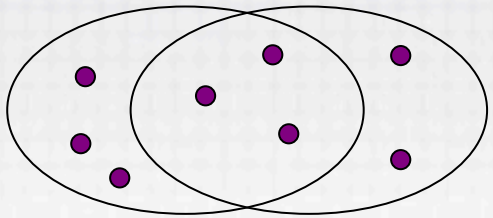
- For N = 10 million, it takes more than a year…

Distance measures

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- For each application, we first need to define what "distance" means
- Jaccard distance / similarity
  - The Jaccard Similarity of two sets is the size of their intersection / the size of their union:

$$sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$
$$d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$$



3 in intersection
8 in union
Jaccard similarity= 3/8
Jaccard distance = 5/8

3 essential steps for finding similar documents

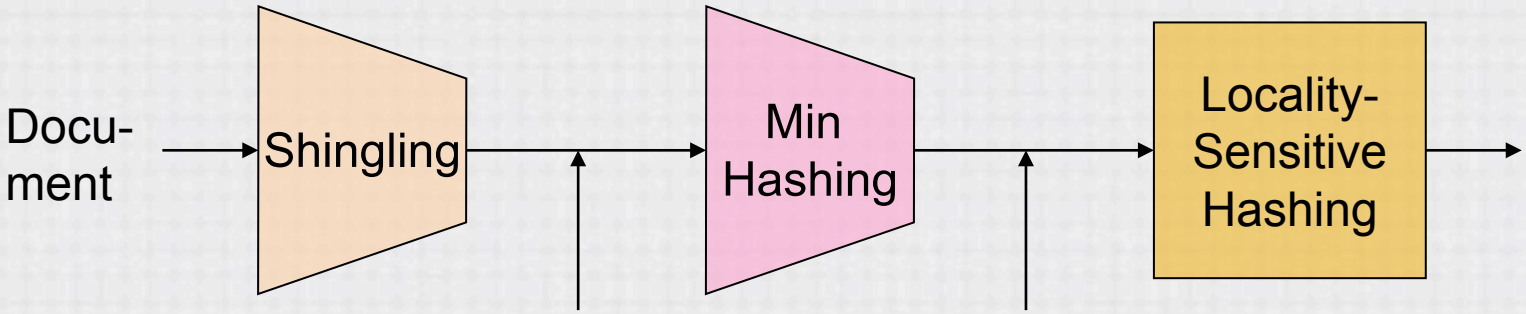
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

1. Shingling: Convert documents to sets

2. Minhashing: Convert large sets to short signatures, while preserving similarity

3. Locality-sensitive hashing: Focus on candidate pairs

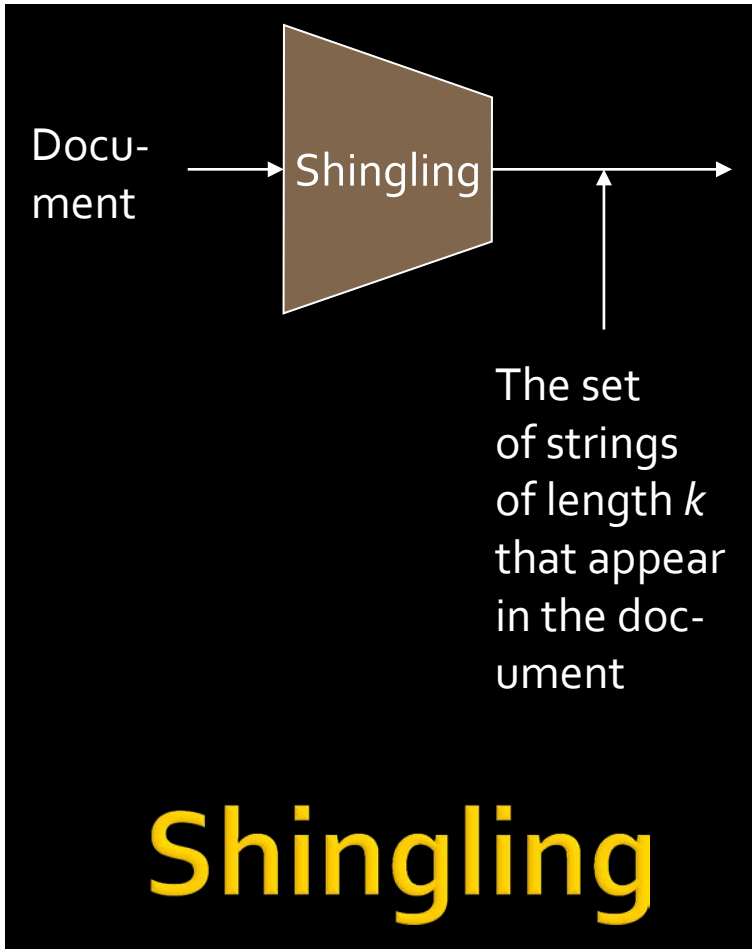
The Big Picture



The set of strings of length *k* that appear in the doc-ument

Signatures: short integer vectors that represent the sets, and reflect their similarity

Candidate pairs: those pairs of signatures that we need to test for similarity


Docu-
ment
Shingling
The set
of strings
of length k
that appear
in the doc-
ument
Shingling

Document as high-dim data

- Step 1: Shingling : Convert documents to sets

- Simple approaches:
  - Document = set of words appearing in document
  - Document = set of "important" words
  - Don't work well for this application. Why?

- Need to account for ordering of words!
- A different way: Shingles!

Definition of shingles

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- A *k-shingle* (or *k-gram*) for a document is a sequence of *k* tokens that appear in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = words for example
- Examples: k=2; document D_1 = a b c a b

  Set of 2-shingles: $S(D_1)$ = { ab, bc, ca }
  - Option: shingles as a bag(multiset), count ab twice: $S'(D_1)$ = { ab, bc, ca, ab }

Compressing shingles

- To compress long shingles, we can hash them to 4 bytes (for example)
- Represent a doc by the set of hash values of its k-shingles
  - Idea: two documents could appear to have shingles in common, when in fact only the hash-values were shared
- Examples: k=2; document D_1 = a b c a b

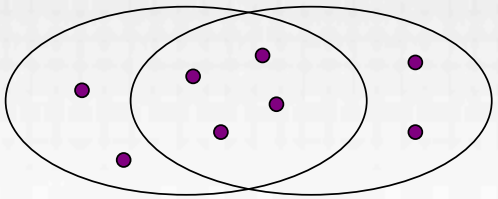
  Set of 2-shingles: $S(D_1)$ = { ab, bc, ca }

  Hash the shingles: $h(D_1)$ = {1, 6, 3}

Similarity metric for shingles

- Document D_1 = set of k-shingles $C_1 = S(D_1)$
- Equivalently, each document is a 0/1 vector in the space of k-shingles
  - Each unique shingle is a dimension
  - Vectors are very sparse
- A natural similarity measure is the Jaccard similarity:

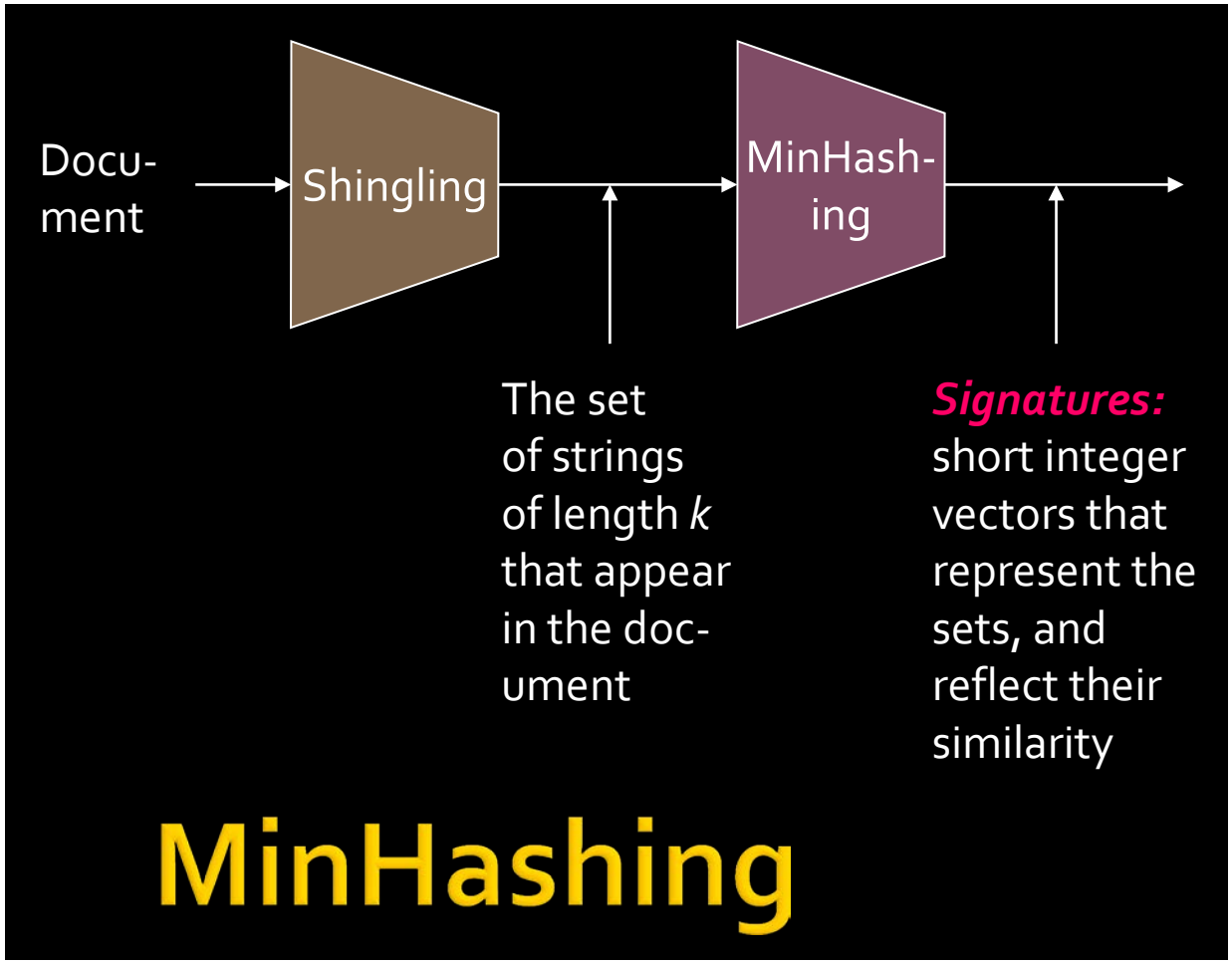
$$Sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



Assumptions

- Documents that have lots of shingles in common have similar text, even if the text appears in different order

- Caveat: You must pick k reasonable enough
  - E.g., token is character
  - K = 5 is OK for short documents
  - K = 10 is better for long documents


Docu-
ment
Shingling
MinHash-
ing
The set
of strings
of length k
that appear
in the doc-
ument
Signatures:
short integer
vectors that
represent the
sets, and
reflect their
similarity
MinHashing

Encoding documents as big vectors

- We can generate a universal set (dictionary)
  - E.g., all unique words
- Encode documents using 0/1 (bit, boolean) vectors
  - One dimension per element in the universal set
- Interpret set intersection as bitwise AND, and set union as bitwise OR
- Example: C_1 = 10111; C_2 = 10011
  - Size of intersection = 3; size of union = 4

    Jaccard similarity (not distance) = ¾
  - $d(C_1, C_2)$ = 1 - (Jaccard similarity) = ¼

From sets to boolean matrices

- 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 is the Jaccard similarity of the corresponding sets (rows with value 1)
  - Typical matrix is sparse!
- Each document is a column
  - Example: sim(c_1, c_2) = ?
    - Size of the intersection = 3; size of union = 6
      Jaccard similarity = 3/6
    - d(c_1, c_2) = 3/6

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

Outline: finding similar columns

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- So far:
  - Documents → Sets of shingles
  - Represent sets as Boolean vectors in a matrix
- Next Goal: Find similar columns
- Approach:
  - Signatures of columns: small summaries of columns
  - Examine pairs of signatures to find similar columns
    - Essential: similarities of signatures & columns are related
  - Optional: check that columns with similar signatures are really similar
- Warnings:
  - Comparing all signature pairs may take too much time: Job for LSH
    - These methods can produce false negatives, and even false positives (if the optional check is not made)

Hashing Columns (Signatures)

- Key idea: "hash" each column C to a small signature h(C), such that:
  - h(c) is small enough that the signature fits in RAM
  - sim(c_1, c_2) is the same as the "similarity" of signature h(c_1) and h(c_2)
- Goal: Find a hash function h(.) such that:
  - If sim(c_1, c_2) is high, then with high prob. h(c_1) = h(c_2)
  - If sim(c_1, c_2) is low, then with high prob. h(c_1) ≠ h(c_2)

Min-Hashing

- Goal: Find a hash function h(.) such that:
  - If sim(c_1, c_2) is high, then with high prob. $h(c_1)=h(c_2)$
  - If sim(c_1, c_2) is low, then with high prob. $h(c_1)\neq h(c_2)$
- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function
- There is a suitable hash function for Jaccard similarity: Min-hashing

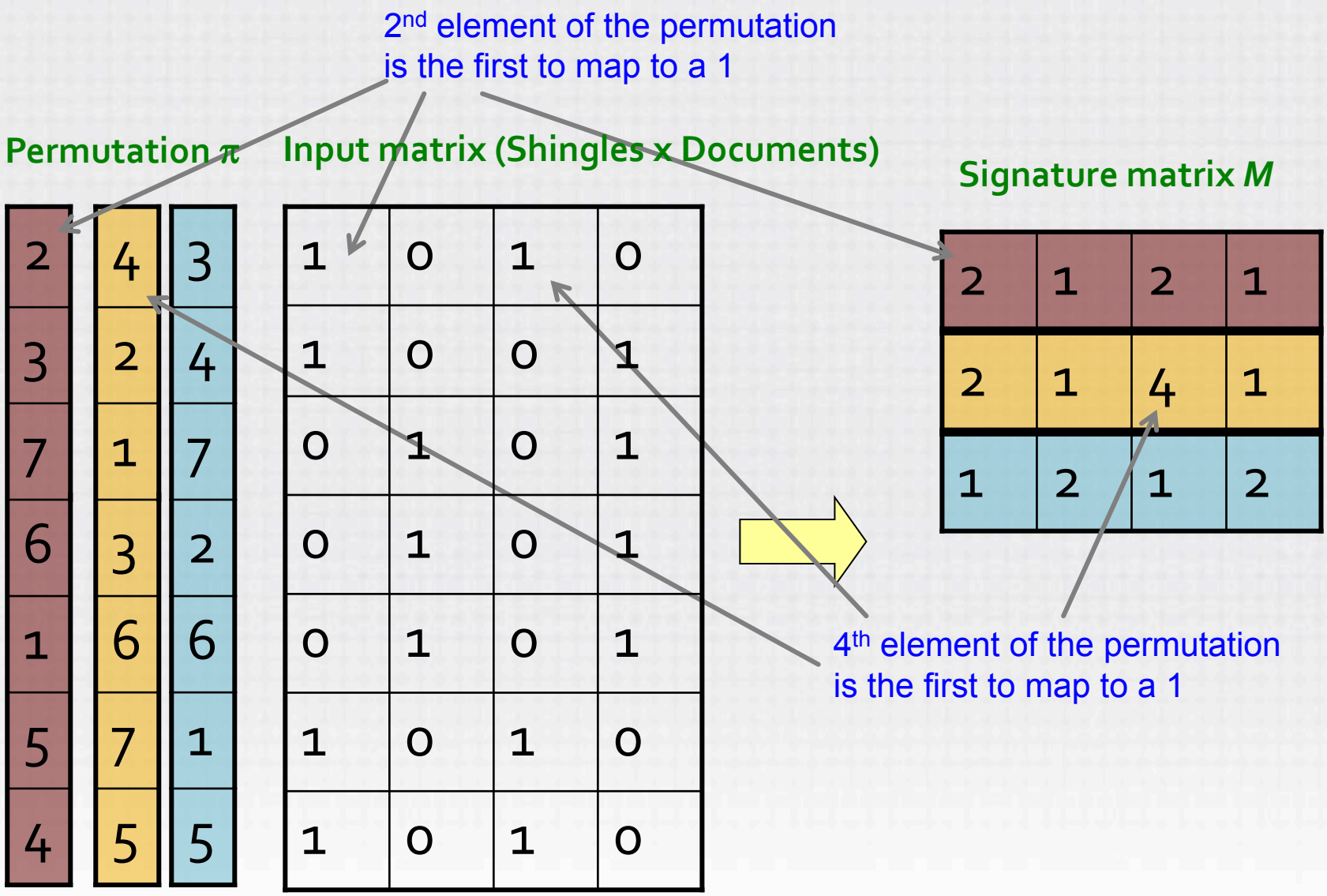
Min-Hashing

- Image the rows of the Boolean matrix permuted under random permutation π
- Define a "hash" function $h_\pi(c)$ = the number of the first (in the permuted order) row in which the column C has value 1:

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

- Use several (e.g., 100) independent hash functions to create a signature of a column

Min-Hashing Example



Surprising property

- Choose a random permutation π
- Claim:

$$\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$$

- Why?

0	0
0	0
1	**1**
0	0
0	1
1	0

Four types of rows

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Given columns C_1 and C_2, rows may be classified as:

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

  - a = # of rows of type A, etc.
- Note: $sim(C_1, C_2) = a/(a+b+c)$
- Then: $\mathbf{Pr}[h(C_1) = h(C_2)] = Sim(C_1, C_2)$
  - Look down the columns C_1 and C_2 until we see a 1
  - If it's a type-A row, the $h(C_1) = h(C_2)$
  - If it's a type B or type-C row, then not

Similarity for signatures

- We know: $\mathbf{Pr}[h(C_1) = h(C_2)] = Sim(C_1, C_2)$
- Now generalize to multiple hash functions
- The similarity of two signatures is the fraction of the hash functions in which they agree

Min-Hashing Example

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Permutation π

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

Input matrix (Shingles x Documents)

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*

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

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

Min-Hashing signatures

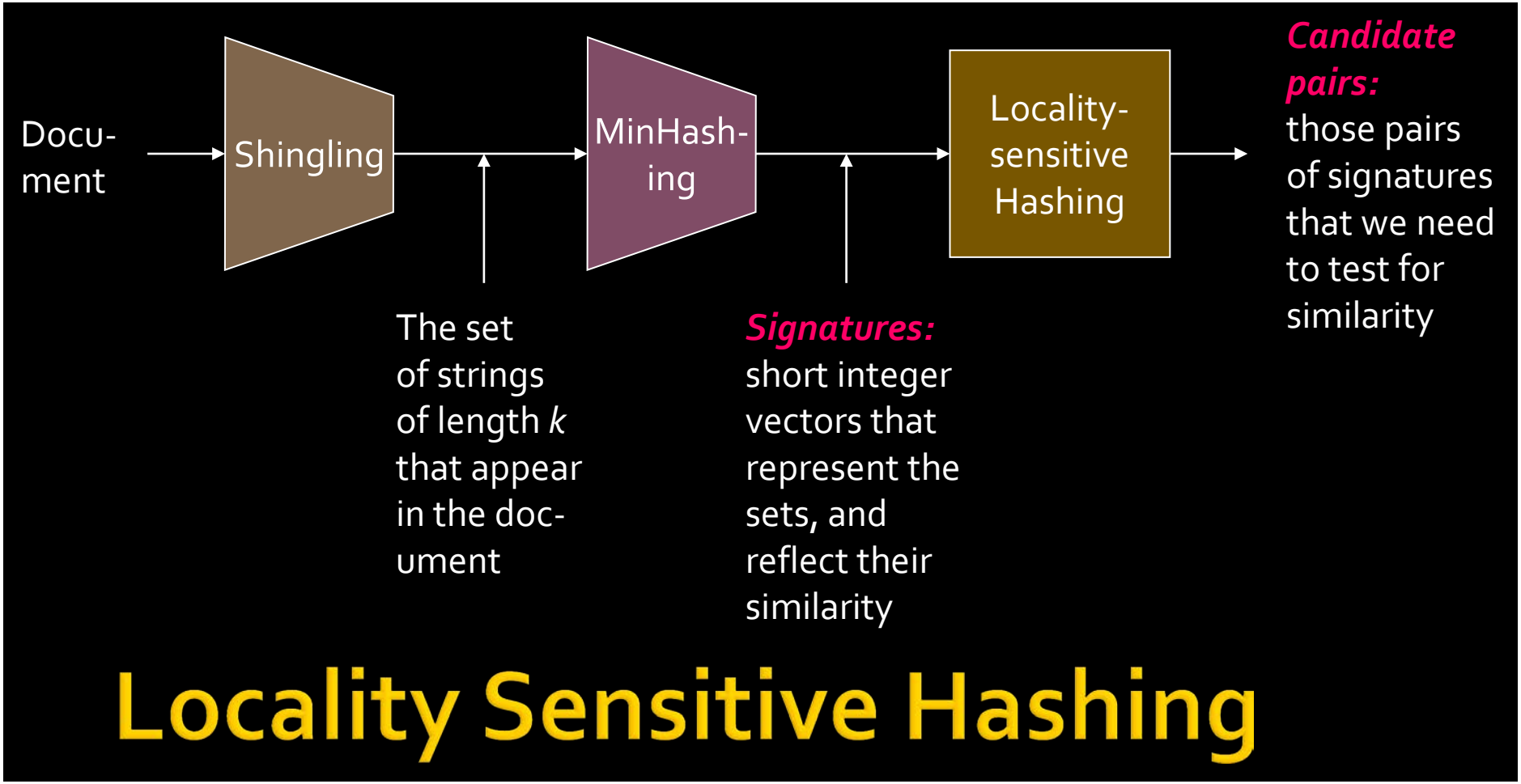
- Pick K = 100 random permutations of the the rows
- Think of sig(C) as a column vector
- sig(C)[i] = according to the i^{th} permutation, the index of the first row that has a 1 in column C

$$sig(\mathrm{C})[\mathrm{i}] = \min\left(\pi_{\mathrm{i}}(\mathrm{C})\right)$$

- Note: The sketch (signature) of document C is small: ~100 bytes
- We achieved our goal: compress long big vectors into short signatures

Implementation Trick

- Permuting rows even once is prohibitive
- Row hashing!
  - Pick K = 100 hash functions k_i
  - Ordering under k_i gives a random row permutation
- One-pass implementation
  - For each column C and hash function k_i keep a slot for the min-hash value
  - Initialize all sig(C)[i] = ∞
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:
      - If k(j)<sig(c)[i], then sig(c)[i] ← k_i(j)


Docu-
ment
Shingling
MinHash-
ing
Locality-
sensitive
Hashing
Candidate
pairs:
those pairs
of signatures
that we need
to test for
similarity
The set
of strings
of length k
that appear
in the doc-
ument
Signatures:
short integer
vectors that
represent the
sets, and
reflect their
similarity
Locality Sensitive Hashing

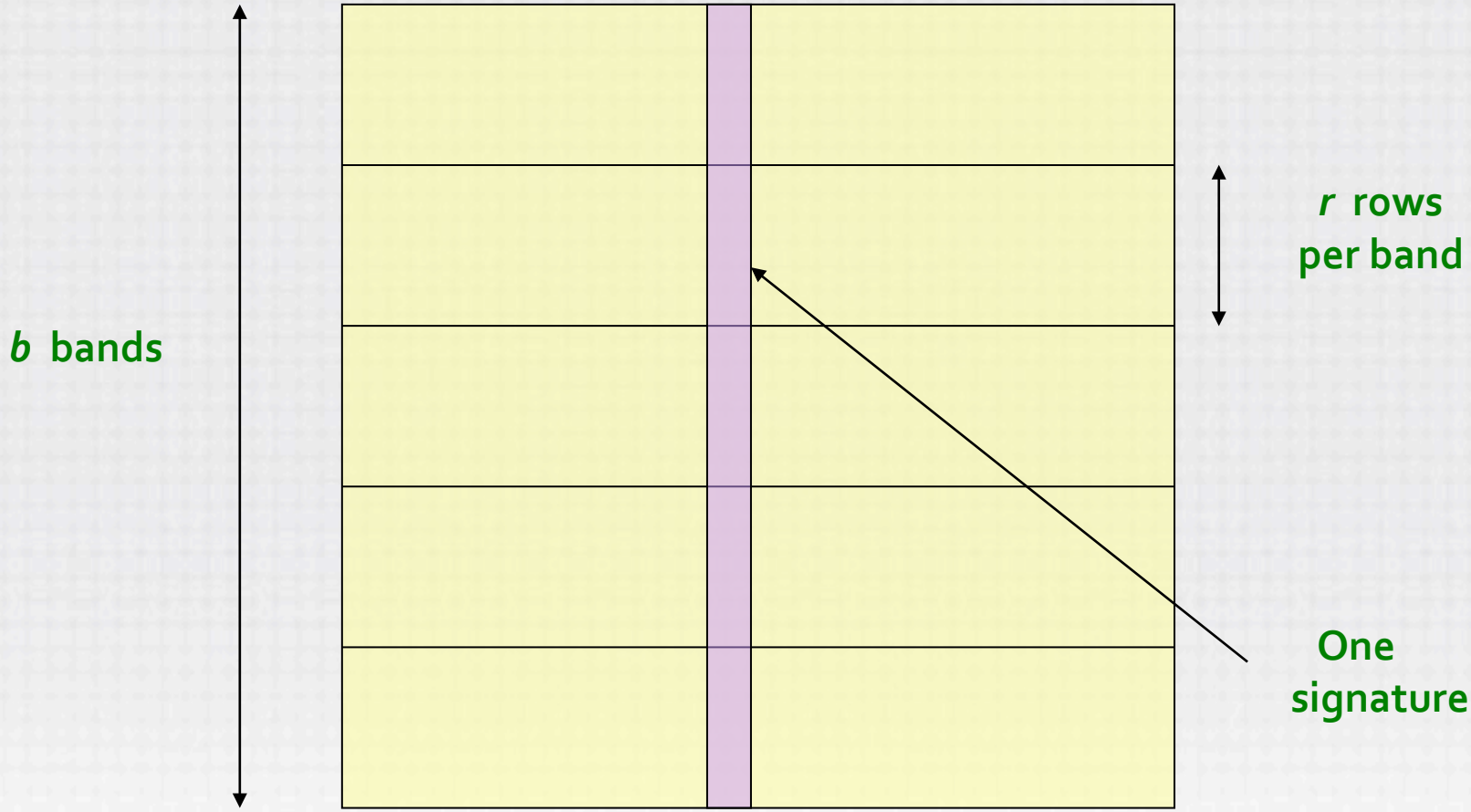
LSH for MinHash

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- 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

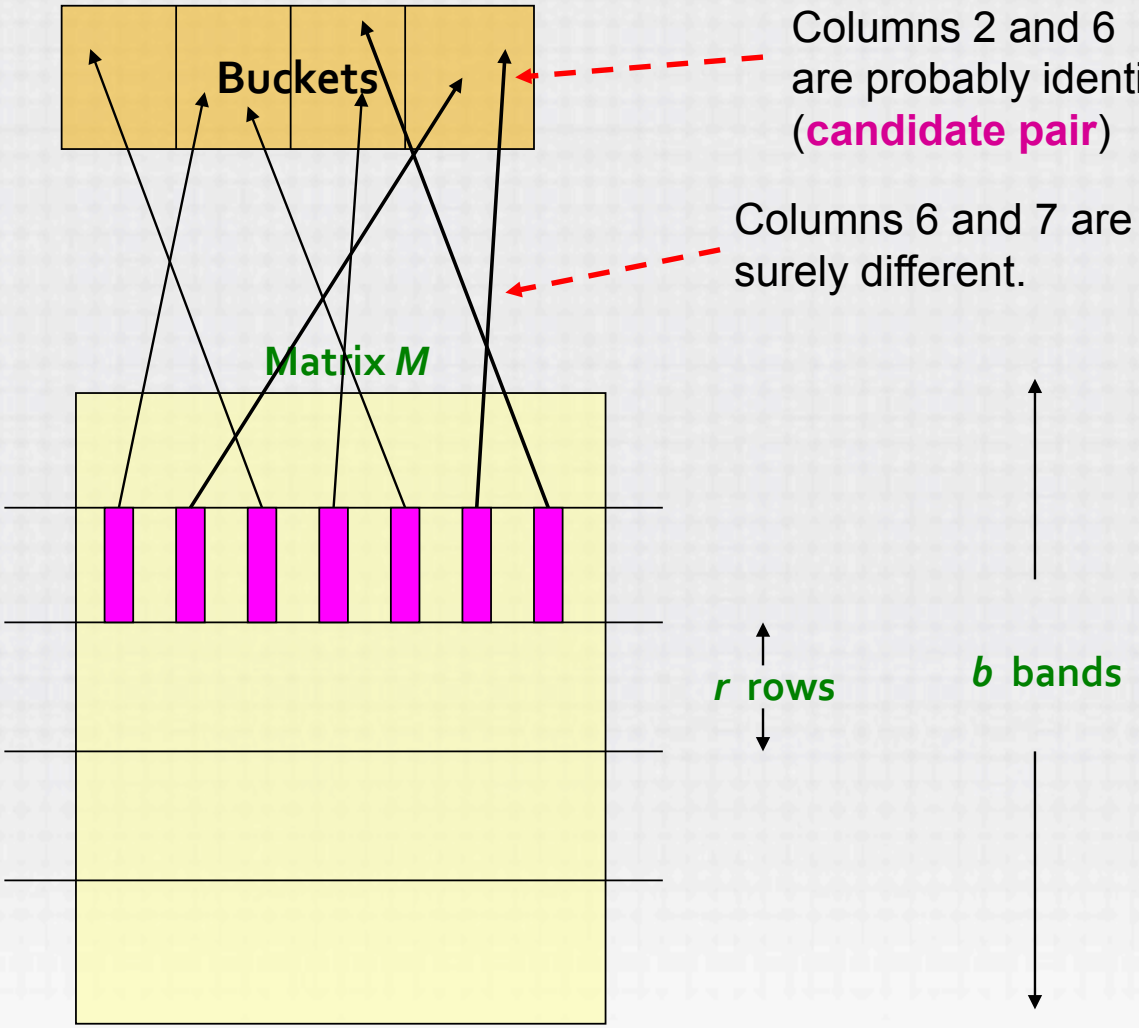
Partition M into b bands



Partition M into b bands

- Divide matrix M into b bands of r rows each
- For each band, hash its portion of each column to a hash table with k buckets
  - 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



Simplifying assumption

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- 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

Examples

- 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

C_1, C_2 are 80% similar

- Find pairs of documents >=0.8 similarity, set b=20, r=5
- Assume: sim(C_1, C_2) = 0.8
  - Since sim(c_1, c_2) >=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 could find 99.965% pairs of truly similar documents

C_1, C_2 are 30% similar

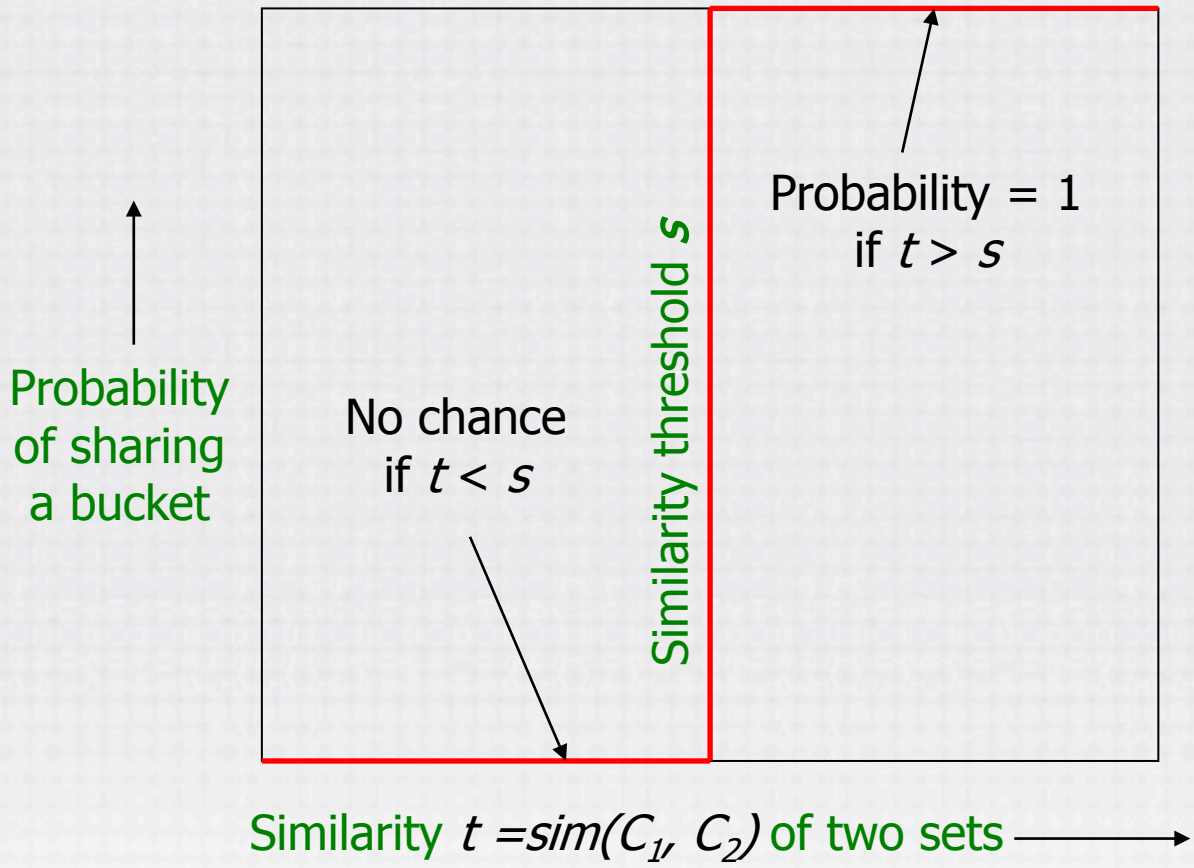
- Find pairs of documents <=0.3 similarity, set b=20, r=5
- Assume: sim(C_1, C_2) = 0.3
  - Since sim(c_1, c_2) <=s, we want c_1, c_2 to hash to No common buckets (all bands should be different)
- Probability c_1, c_2 identical in one particular band:
  $(0.3)^5 = 0.00243$
- Probability c_1, c_2 identical 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 but then it will turn out their similarity is below threshold s

Tradeoff in LSH

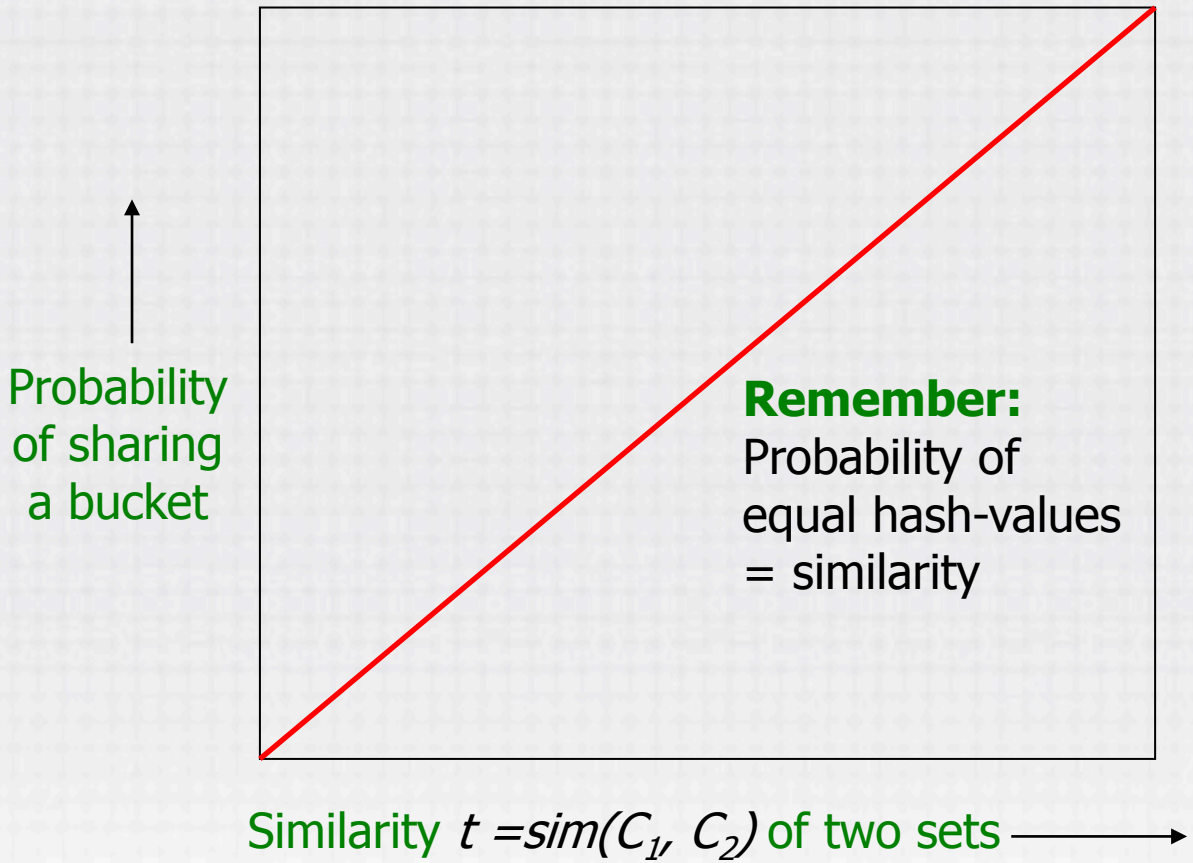
- Pick:
  - The number of min-hashes (rows of M)
  - The number of bands *b*, and
  - The number of rows *r* per band to balance false positives / negatives

- Example: if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

Analysis of LSH – what we want



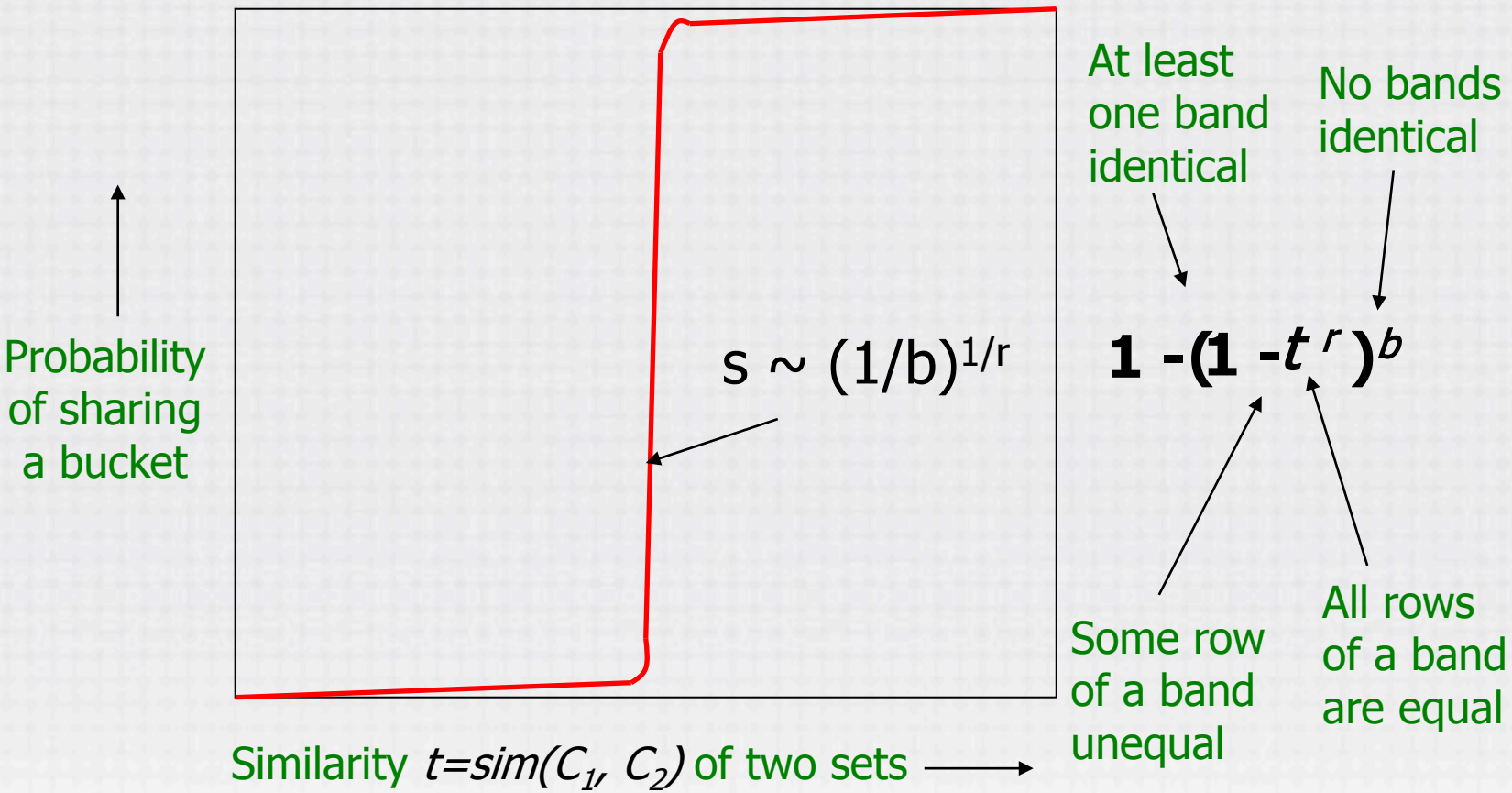
What 1 band of 1 row gives you



B bands, r rows/band

- Column 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 = $1-(1-t^r)^b$

What b bands of r rows gives you



Example: b = 20, r = 5

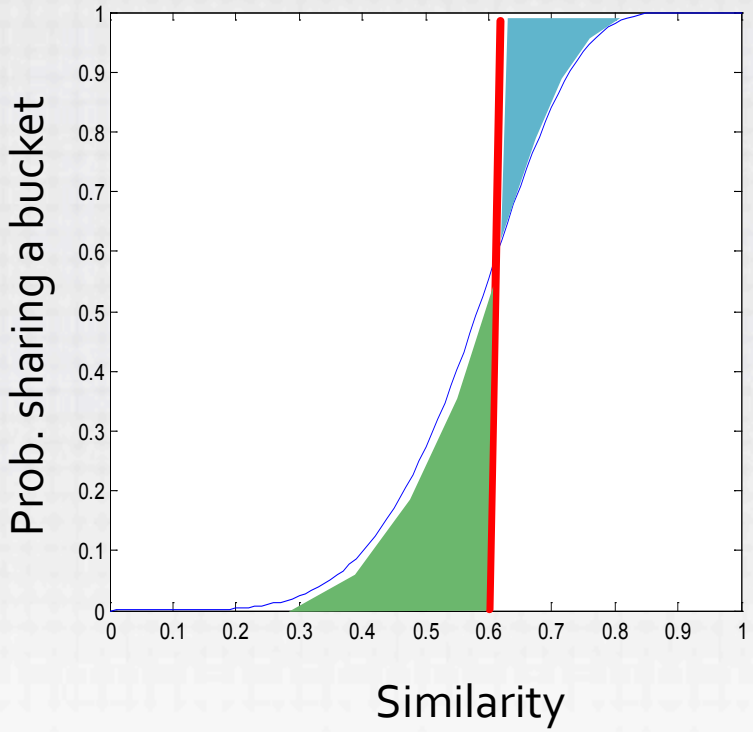
- Similarity threshold *s*
- Prob. that at least 1 band is identical:

s	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

Picking r and b: the S-curve

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Picking r and b to get the best S-curve
  - 50 hash-functions (r=5, b=10)



Blue area: False Negative rate
Green area: False Positive rate

LSH summary

- 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

Summary: 3 steps

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

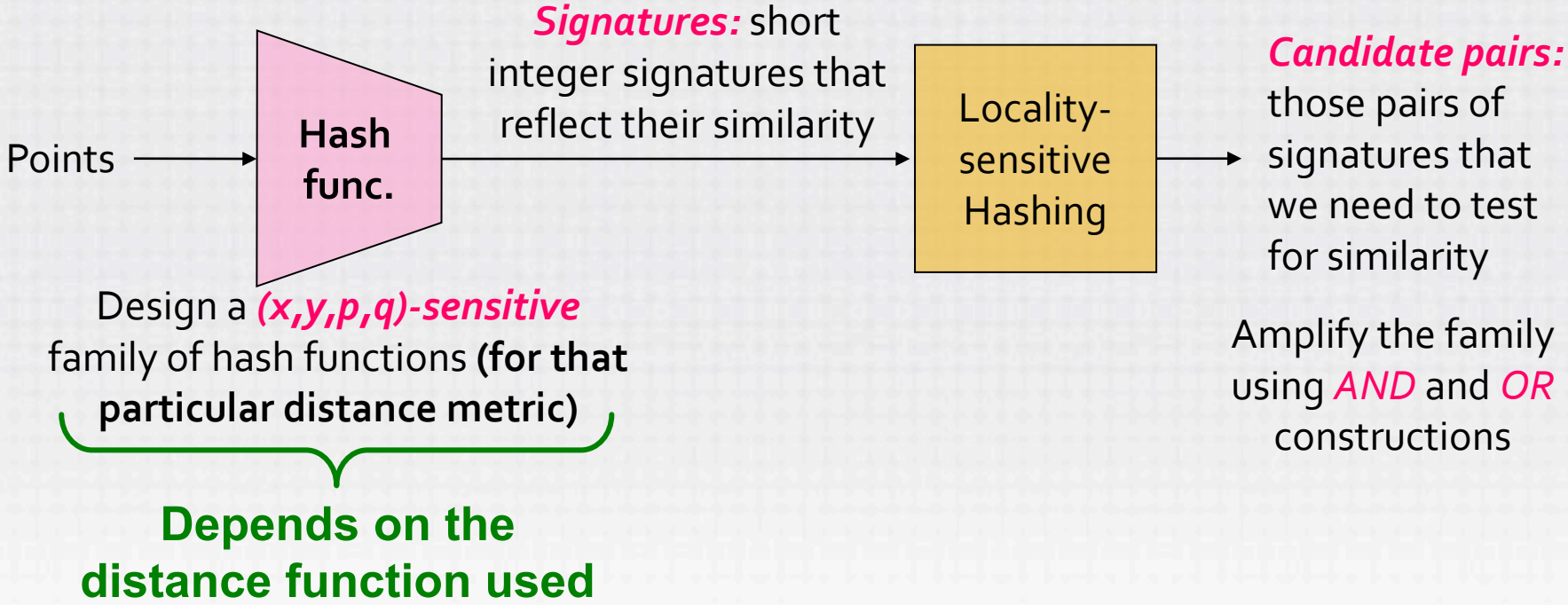
- **Shingling:** Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property:

$$\Pr[h_\pi(C1) = h_\pi(C2)] = sim(C1, C2)$$

  - 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**

LSH for other distance metrics

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

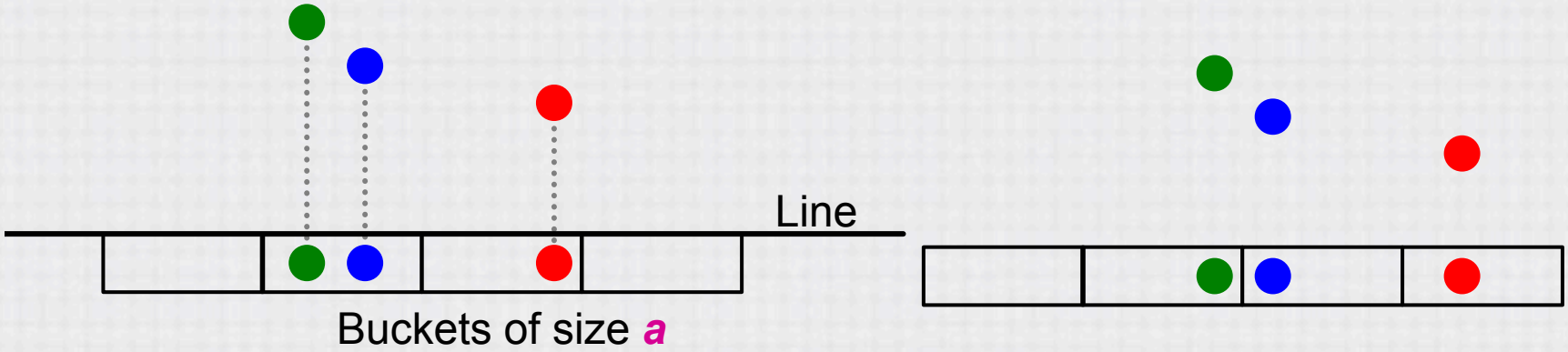


LSH for Euclidean distance

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

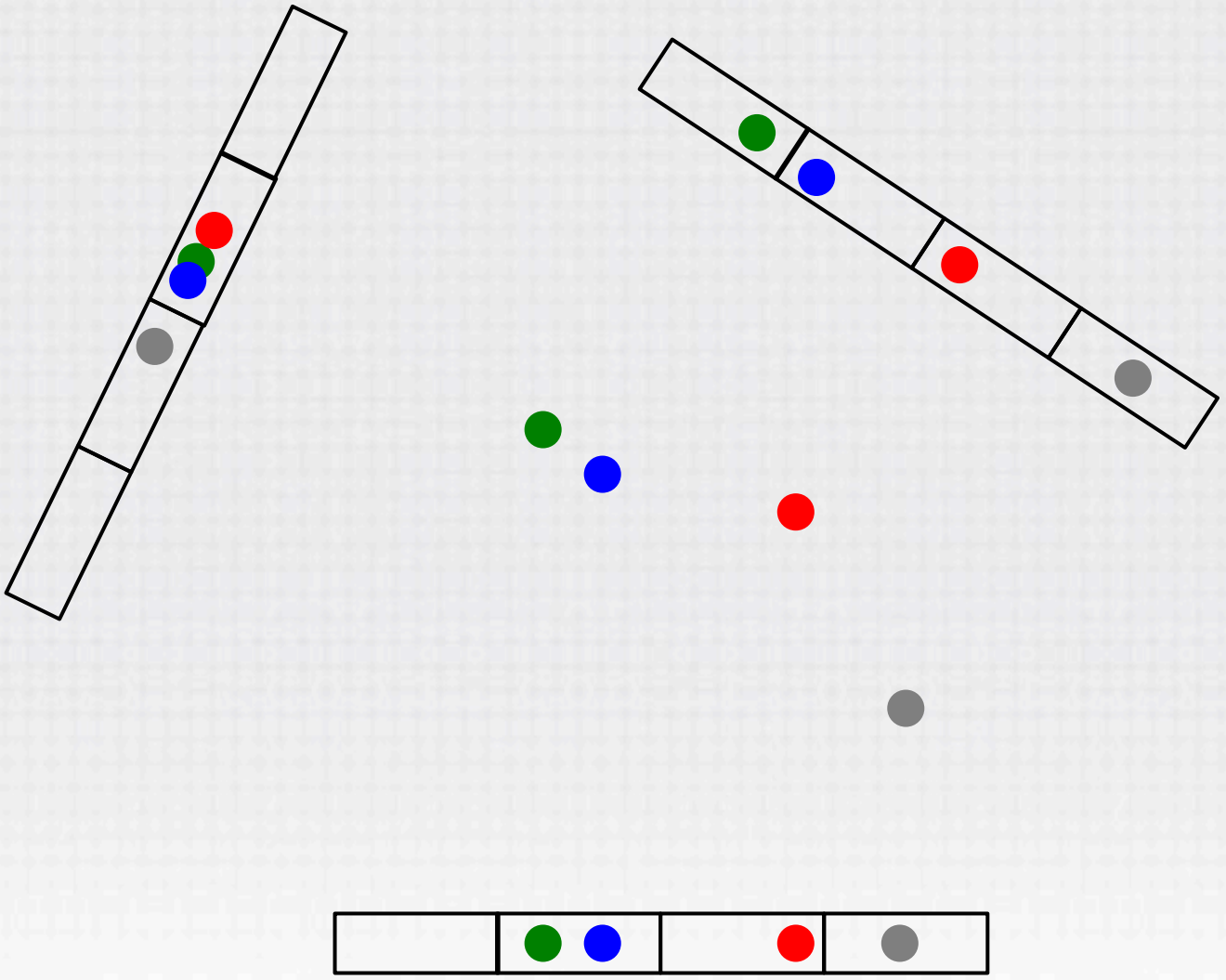
- Simple idea: Hash functions correspond to lines
- Partition the line into buckets of size a
- Hash each point to the bucket containing its projection onto the line
- Nearby points are always close; distant points are rarely in same bucket

Projection of points



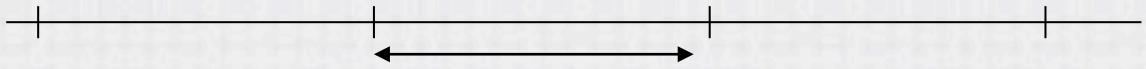
- Lucky case:
  - Points that are close hash in the same bucket
  - Distant points end up in different buckets
- Unlucky case:

Multiple projections



Projection of points

If ***d*** << ***a***, then the chance the points are in the same bucket is at least **1 – *d*/*a***.



Projection of points

If ***d*** **>>** ***a***, θ must be close to 90° for there to be any chance points go to the same bucket.