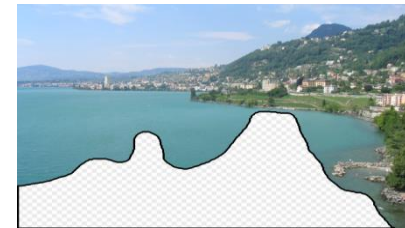


Locality Sensitive Hashing (LSH)

A Common Metaphor

- Many problems can be expressed as finding “similar” sets:
 - Find near-neighbors in high-dimensional space
- **Examples:**
 - Pages with similar words
 - For duplicate detection, classification by topic
 - Customers who purchased similar products
 - Products with similar customer sets
 - Images with similar features
 - Users who visited similar websites



Locality Sensitive Hashing (LSH)

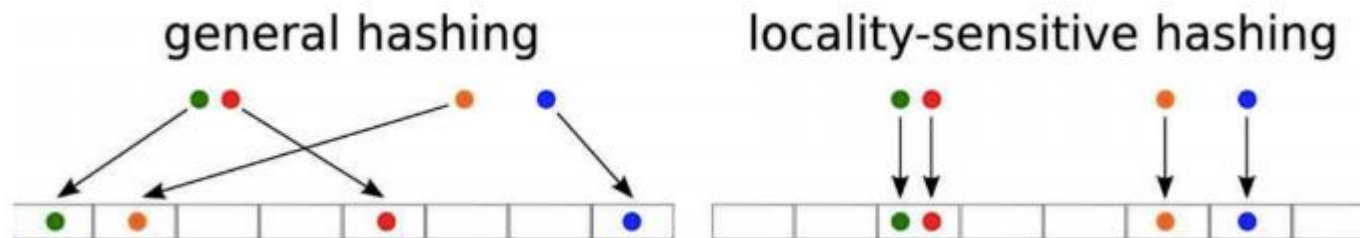
- The task of finding nearest neighbours is very common.
- You can think of applications **like finding duplicate or similar documents, audio/video search**. Although using brute force to check for all possible combinations will give you the exact nearest neighbour but it's not scalable at all.
- Approximate algorithms to accomplish this task has been an area of active research. Although these algorithms don't guarantee to give you the exact answer, more often than not they'll provide a good approximation. These algorithms are faster and scalable.
- Locality sensitive hashing (LSH) is a procedure for finding similar pairs in a large dataset. For a dataset of size N , the brute force method of comparing every possible pair would take $N!/(2!(N-2)!) \sim N^2/2 = O(N^2)$ time. The LSH method aims to cut this down to $O(N)$ time.

Locality Sensitive Hashing (LSH)

- Locality sensitive hashing (LSH) is one such algorithm. LSH has many applications, including:
 - Near-duplicate detection: LSH is commonly used to deduplicate large quantities of documents, webpages, and other files.
 - Genome-wide association study: Biologists often use LSH to identify similar gene expressions in genome databases.
 - Large-scale image search: Google used LSH along with PageRank to build their image search technology VisualRank.
 - Audio/video fingerprinting: In multimedia technologies, LSH is widely used as a fingerprinting technique A/V data.

Locality Sensitive Hashing (LSH)

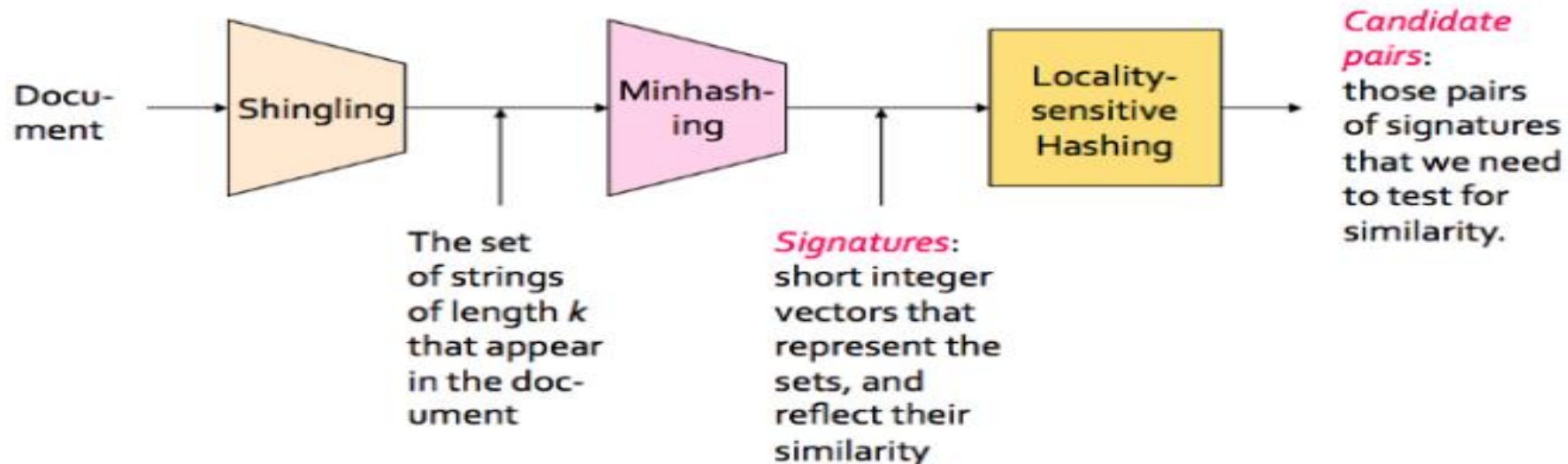
- LSH refers to a family of functions (known as LSH families) to hash data points into buckets so that **data points near each other are located in the same buckets with high probability**, while data points far from each other are likely to be in different buckets. This makes it easier to identify observations with various degrees of similarity.

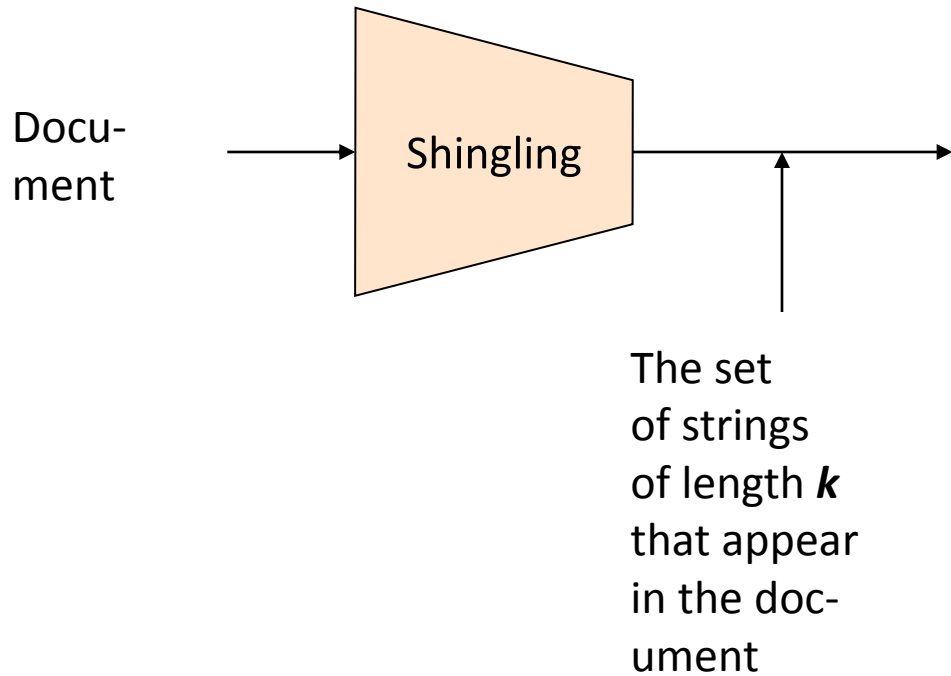


- Finding similar documents
 - how we can leverage LSH in solving an actual problem. The problem that we're trying to solve: **Goal:** You have been given a large collections of documents. You want to find “near duplicate” pairs.

Locality Sensitive Hashing (LSH)

- In the context of this problem, we can break down the LSH algorithm into 3 broad steps:
- **Shingling** - Convert documents to sets
- **Min hashing** – Convert large sets to short signatures while preserving similarity. Signature estimate the Jaccard similarity of sets.
- **Locality-sensitive hashing** - Focus on pairs of signatures likely to be from similar documents





Shingling

Step 1: *Shingling*: Convert documents to sets

In order to quantify a document, we need to vectorize it. One method for doing this, is to enumerate all of its **k-shingles**. A **k-shingle** is just any k -consecutive characters occurring in the document.

Locality Sensitive Hashing (LSH)

- **Shingling**

- In this step, we convert each document into a set of characters of length k (also known as k -shingles or k -grams). The key idea is to represent each document in our collection as a set of k -shingles.
- For ex: One of your document (D): “Nadal”. Now if we’re interested in 2-shingles, then our set: {Na, ad, da, al}. Similarly set of 3-shingles: {Nad, ada, dal}.
- Similar documents are more likely to share more shingles
- k value of 8–10 is generally used in practice. A small value will result in many shingles which are present in most of the documents (bad for differentiating documents)

Finding Similar Items

- **Simple approaches:**
 - Document = set of words appearing in document
 - Document = set of “important” words
- A k-shingle (or k-gram) for a document is a sequence of k tokens that appears in the doc
 - Tokens can be characters, words or something else, depending on the application
 - Assume tokens = characters for examples
 - **Example: $k=2$; document $D_1 = \text{abcab}$**
Set of 2-shingles: $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$

Finding Similar Items

- **Finding Similar Documents (3 – Steps)**
- **Step 1: Shingling: Convert documents to sets**
- "a rose is a rose is a rose"
 - The set of all contiguous sequences of 4 tokens (Thus $4=n$, thus 4-grams) is
 - { (a,rose,is,a), (rose,is,a,rose), (is,a,rose,is),
(a,rose,is,a), (rose,is,a,rose) }
 - reduced to { (a,rose,is,a), (rose,is,a,rose),
(is,a,rose,is) }.

Finding Similar Items

- Choosing the Shingle Size
 - k should be picked large enough that the probability of any given shingle appearing in any given document is low.
 - Effect of k : $k = 1$, most Web pages will have most of the common characters and few other characters, so almost all Web pages will have high similarity.
 - Short (E-mail): $k = 5$
 - large documents, such as research articles: choice $k = 9$

Choosing k

- Let $k = 5$
- Suppose that only letters and general white space characters appear in emails.
- If so, there would be $27^5 = 14,348,907$ possible shingles.
- Since the typical email is much smaller than 14 million characters long, we would expect $k = 5$ to work well and it does.
- A rule of thumb is to imagine there are 20 characters and estimate the number of shingles as 20^k . For large documents (research articles), a choice of $k = 9$ is considered to be safe

Finding Similar Items

- Matrix Representation of Sets
 - To construct small signatures from large sets,
 - First : visualize a collection of sets as their characteristic matrix
 - The **columns of the matrix** correspond to the sets,
 - **The rows correspond to elements** of the universal set from which elements of the sets are drawn.

Finding Similar Items

- Matrix Representation of Sets

– Here, $S_1 = \{a, d\}$, $S_2 = \{c\}$, $S_3 = \{b, d, e\}$, and $S_4 = \{a, c, d\}$.

rows correspond to
elements of the
universal set

columns of the matrix
correspond to the sets
..like $S_1 = \{a, b\}$

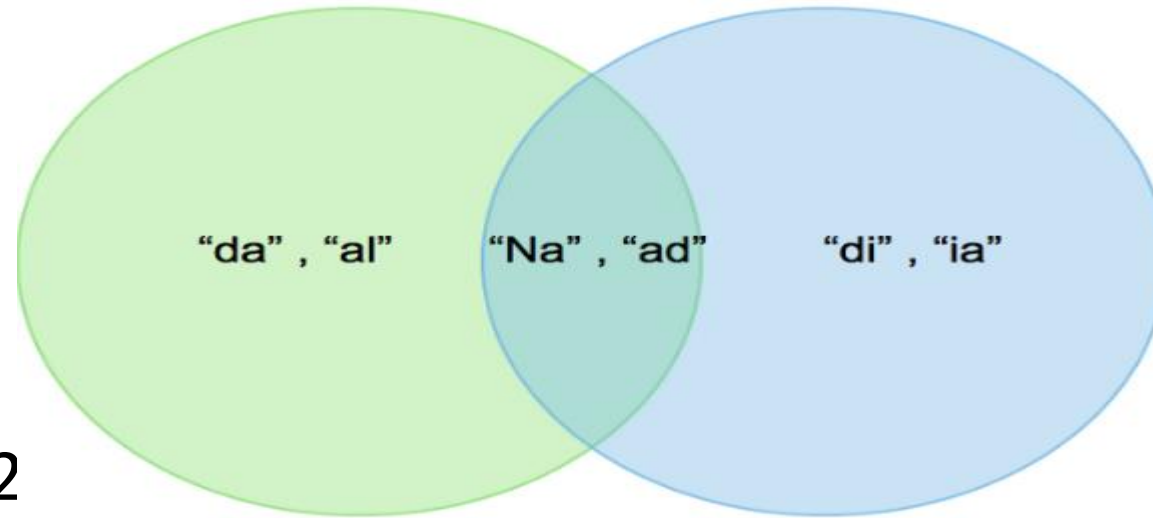
<i>Element</i>	S_1	S_2	S_3	S_4
<i>a</i>	1	0	0	1
<i>b</i>	0	0	1	0
<i>c</i>	0	1	0	1
<i>d</i>	1	0	1	1
<i>e</i>	0	0	1	0

Fig 1: Matrix Representation of sets

Locality Sensitive Hashing (LSH)

- Jaccard Index
 - The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union:
$$\text{sim}(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$
 - **Jaccard distance:** $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$
 - Suppose A: “Nadal” and B: “Nadia”, then 2-shingles representation will be: A: {Na, ad, da, al} and B: {Na, ad, di, ia}.

Locality Sensitive Hashing (LSH)



- Jaccard Index = 2
- More number of common shingles will result in bigger Jaccard Index and hence more likely that the documents are similar.
- Let's discuss 2 big issues that we need to tackle:
 - Time complexity
 - Space complexity

Locality Sensitive Hashing (LSH)

- **Time complexity**

- Now you may be thinking that we can stop here. But if you think about the scalability, doing just this won't work. For a collection of n documents, you need to do $n*(n-1)/2$ comparison, basically **$O(n^2)$** . Imagine you have 1 million documents, then the number of comparison will be $5*10^{11}$.

- **Space complexity**

- The document matrix is a sparse matrix and storing it as it is will be a big memory overhead.
- One way to solve this is hashing.

Locality Sensitive Hashing (LSH)

- **Hashing**
- The idea of hashing is to convert each document to a small signature using a hashing function H . Suppose a document in our corpus is denoted by d . Then:
- $H(d)$ is the signature and it's small enough to fit in memory
- If $\text{similarity}(d_1, d_2)$ is high then $\text{Probability}(H(d_1) == H(d_2))$ is high
- If $\text{similarity}(d_1, d_2)$ is low then $\text{Probability}(H(d_1) == H(d_2))$ is low
- Choice of hashing function is tightly linked to the similarity metric we're using. For Jaccard similarity the appropriate hashing function is min-hashing.

Locality Sensitive Hashing (LSH)

- Computing Minhash values
 - Rows are permuted randomly
 - Minhasfunc $h(C)$ = the number of the first (n the permuted order) row in which column C has 1.
 - Independent hash functions to create signature of each column.

Minhashing

- Let the order of rows **beadc** be the random permutation for the matrix (fig 1)
- It defines a minhash function h that maps sets to rows.

<i>Element</i>	S_1	S_2	S_3	S_4
<i>b</i>	0	0	1	0
<i>e</i>	0	0	1	0
<i>a</i>	1	0	0	1
<i>d</i>	1	0	1	1
<i>c</i>	0	1	0	1

Minhashing

- To compute *the minhash value of set S_1 according to h* .
- Check for first '1' in the column:
 - The first column, which is the column for set S_1 , has 0 in row b,
 - So we proceed to row e, the second in the permuted order. There is again a 0 in the column for S_1 ,
 - So we proceed to row a, where we find a 1.
 - **$h(S_1) = a$.**
 - **(Obtained similarly) $h(S_2) = c$, $h(S_3) = b$, and $h(S_4) = a$**

<i>Element</i>	S_1	S_2	S_3	S_4
<i>b</i>	0	0	1	0
<i>e</i>	0	0	1	0
<i>a</i>	1	0	0	1
<i>d</i>	1	0	1	1
<i>c</i>	0	1	0	1

Minhashing

- Let **Random number n** (say, 100s or 1000s) be the
- Let minhash functions determined by these permutations h_1, h_2, \dots, h_n .
- From the column representing set S ,
 - – construct the minhash signature for S , the vector $[h_1(S), h_2(S), \dots, h_n(S)]$. List hash-values as columns

<i>Element</i>	S_1	S_2	S_3	S_4
<i>b</i>	0	0	1	0
<i>e</i>	0	0	1	0
<i>a</i>	1	0	0	1
<i>d</i>	1	0	1	1
<i>c</i>	0	1	0	1

a	c	b	a
---	---	---	---

Minhashing

- **Minhashing and Jaccard Similarity**
- The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the jaccard similarity of those sets.
- X rows (1's in both cols)
- Y rows (1 in one of the columns, 0 in the other)
- Z rows (0 in both cols)

C_1		C_2		
0	1		*	
1	0		*	
1	1	*	*	
0	0			
1	1	*	*	
0	1		*	

$\text{Sim}(C_1, C_2) = \frac{2}{5} = 0.4$

Minhashing

- Min hashing (Example-2)
- **Step 1:** Random permutation (π) of *row index* of document shingle matrix.

Permutation π Input matrix (Shingles x Documents)

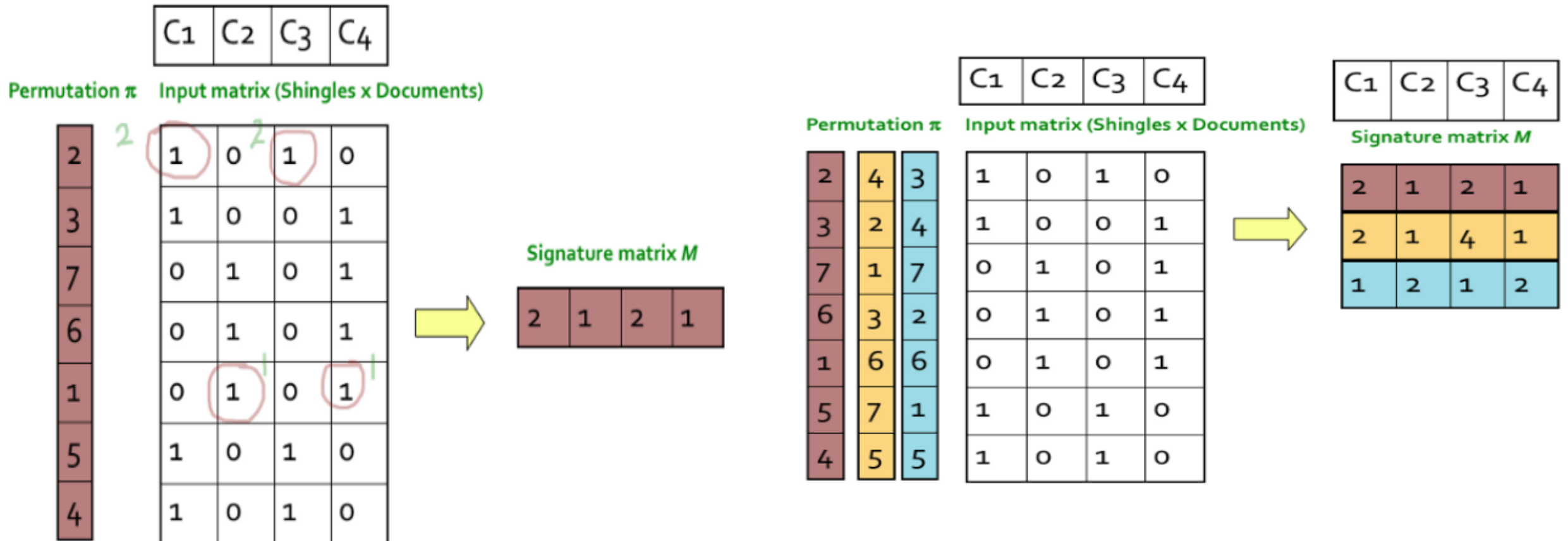
2	1	0	1	0
3	1	0	0	1
7	0	1	0	1
6	0	1	0	1
1	0	1	0	1
5	1	0	1	0
4	1	0	1	0

- **Step 2:** Hash function is the index of the first (in the permuted order) row in which column C has value 1. Do this several time (use different permutations) to create signature of a column.

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

Minhashing

- Min hashing



Minhashing

- Algorithm

- Let $SIG(i, c)$ be the element of the signature matrix for the i th hash function and column c . Initially, set $SIG(i, c)$ to ∞ for all i and c

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 $h_i(r) < SIG(i, c)$ then

$SIG(i, c) = h_i(r)$

Minhashing


- Algorithm

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Row	S_1	S_2	S_3	S_4	$x+1 \bmod 5$	$3x+1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Handwritten notes: $h_1(x)$ points to the first column of the second table, and $h_2(x)$ points to the second column.

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞



	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Row	S_1	S_2	S_3	S_4	$x+1 \bmod 5$	$3x+1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Handwritten notes: Blue boxes highlight the values 1 in the S_1 and S_4 columns of the first table, and the corresponding values in the second table.

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1



	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

Handwritten notes: Blue boxes highlight the values 3 and 2 in the S_2 column of the second table.

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1



	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

Handwritten notes: Blue boxes highlight the values 0 in the S_1 , S_3 , and S_4 columns of the second table.

Row	S_1	S_2	S_3	S_4	$x+1 \bmod 5$	$3x+1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Handwritten notes: Blue boxes highlight the values 1, 0, 1, 1 in the S_2 column and the corresponding values in the $x+1 \bmod 5$ and $3x+1 \bmod 5$ columns.

Row	S_1	S_2	S_3	S_4	$x+1 \bmod 5$	$3x+1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Handwritten notes: Blue boxes highlight the values 0, 1, 1, 0 in the S_2 column and the corresponding values in the $x+1 \bmod 5$ and $3x+1 \bmod 5$ columns.

Minhashing

- Algorithm $P[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

$$\text{sim}(S_1, S_2) = 1 \cdot 0$$

Row	S_1	S_2	S_3	S_4	$x+1 \bmod 5$	$3x+1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

$$\text{Jaccard sim} = \frac{2}{3}$$

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

$$\text{sim}(S_1, S_2) = \frac{1}{2}$$

Row	S_1	S_2	S_3	S_4	$x+1 \bmod 5$	$3x+1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

$$\text{sim}(S_1, S_3) = \frac{1}{4}$$

Minhashing

- Algorithm $P[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

$$\text{sim}(S_1, S_2) = 0$$

Row	S_1	S_2	S_3	S_4	$x + 1 \mod 5$	$3x + 1 \mod 5$
✓ 0	1	0	0	1	1	1
1	0	0	1	0	2	4
✗ 2	0	1	0	1	3	2
✗ 3	1	0	1	1	4	0
4	0	0	1	0	0	3

$$\text{sim}(S_1, S_2) = 0$$

Min-Hashing Example

Permutation π

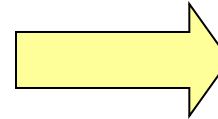
2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

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

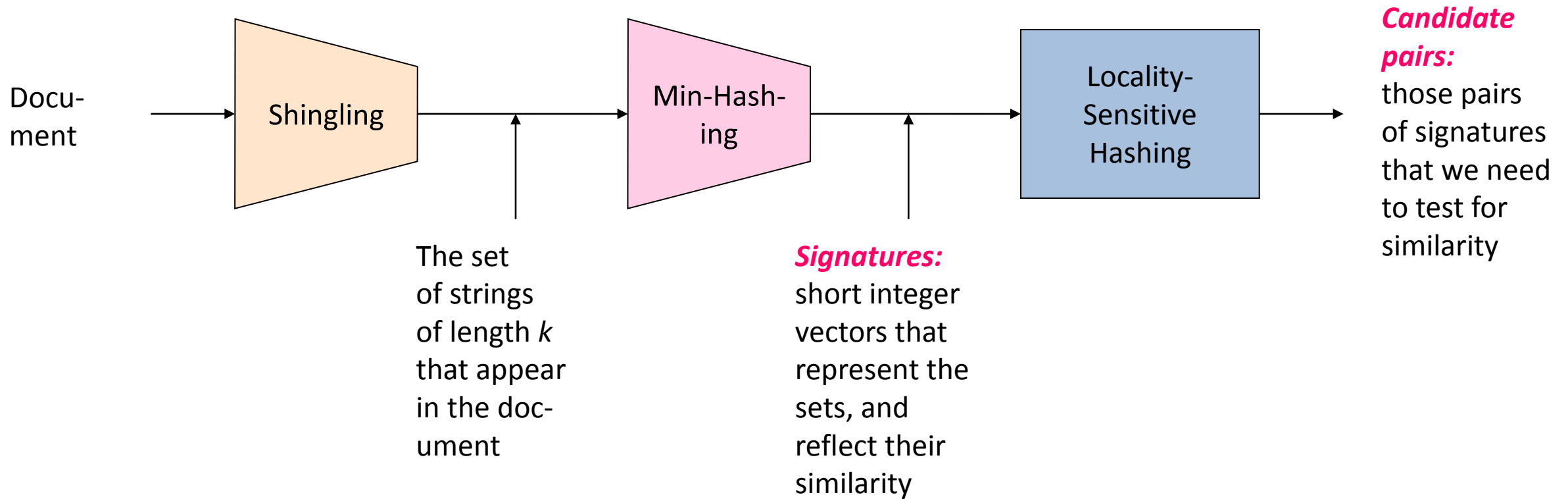


Similarities:

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

Locality Sensitive Hashing (LSH)

- **Locality-sensitive hashing- The band structure procedure.**
- Goal: Find documents with Jaccard similarity of at least t
- The general idea of LSH is to find an algorithm such that if we input signatures of 2 documents, it tells us that those 2 documents form a candidate pair or not i.e. their similarity is greater than a threshold t . Remember that we are taking similarity of signatures as a proxy for Jaccard similarity between the original documents.
- The intuition is this: instead of comparing every pair of elements, what if we could just hash them into buckets, and hopefully elements that map to the same buckets will be the right level of “close” to each other. The level of “close” is precisely the desired similarity threshold



Locality Sensitive Hashing

Step 3: *Locality-Sensitive Hashing:*

Focus on pairs of signatures likely to be from similar documents

LSH: First Cut

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

- 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*

Candidates from Min-Hash

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

- Pick a similarity threshold s ($0 < s < 1$)
- Columns \mathbf{x} and \mathbf{y} of \mathbf{M} are a **candidate pair** if their signatures agree on at least fraction s of their rows:
 $\mathbf{M}(i, \mathbf{x}) = \mathbf{M}(i, \mathbf{y})$ for at least frac. s values of i
 - We expect documents \mathbf{x} and \mathbf{y} to have the same (Jaccard) similarity as their signatures

LSH for Min-Hash

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

- **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

LSH for Min-Hash

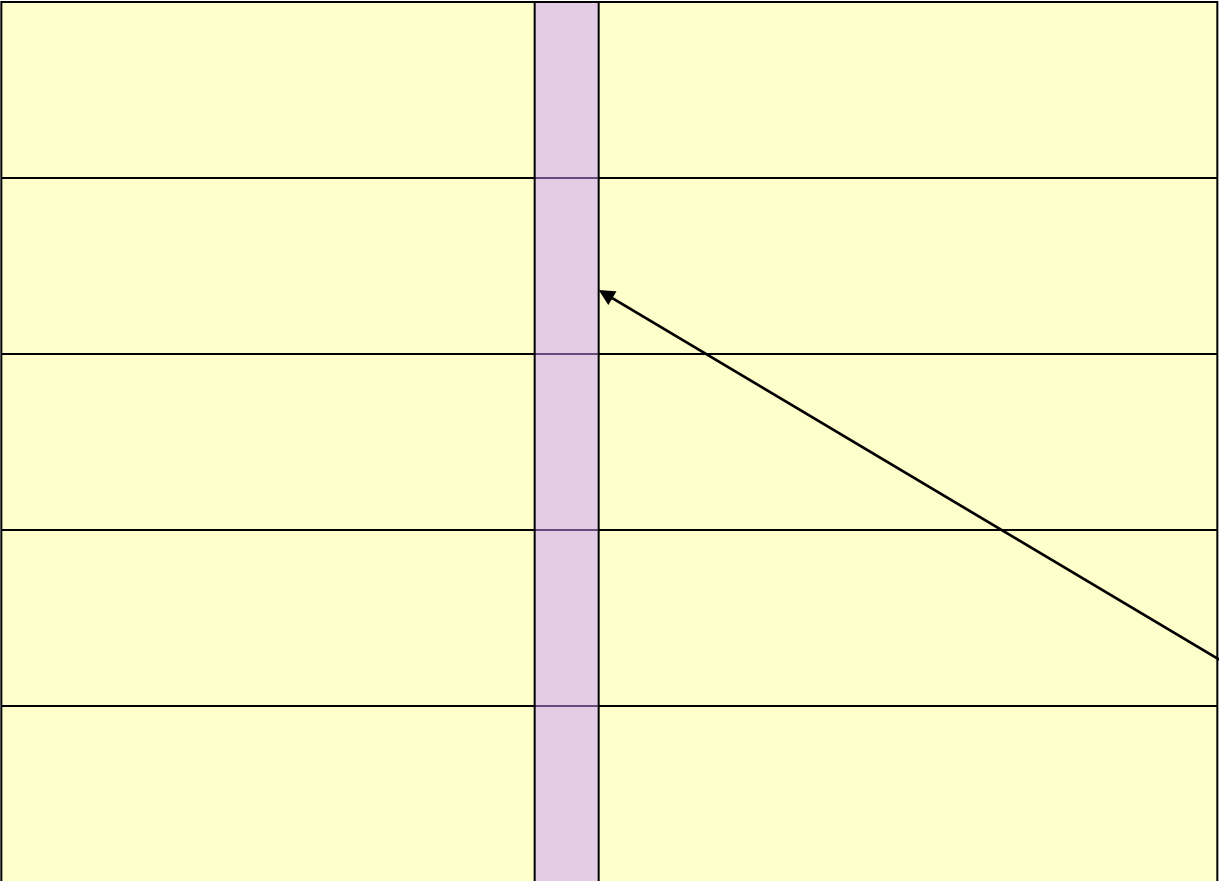
2	1	4	1
1	2	1	2
2	1	2	1

- **Band Method**
- n is the number of hash functions, i.e. the dimension K of the signature matrix. Also define:
 - b as the number of bands
 - r as the number of rows per band
 - and it's apparent that $n=b*r$.
- Two rows are considered **candidate pairs** (meaning they have the possibility to have a similarity score above the desired threshold), if the two rows have **at least one** band in which all the rows are identical
- Hopefully the intuition behind b and r should make sense here; **increasing b gives rows more chances to match**, so it let's in candidate pairs **with lower similarity scores**. **Increasing r makes the match criteria stricter**, restricting to **higher similarity scores**. r and b do opposite things

Partition M into b Bands

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

b bands



r rows
per band

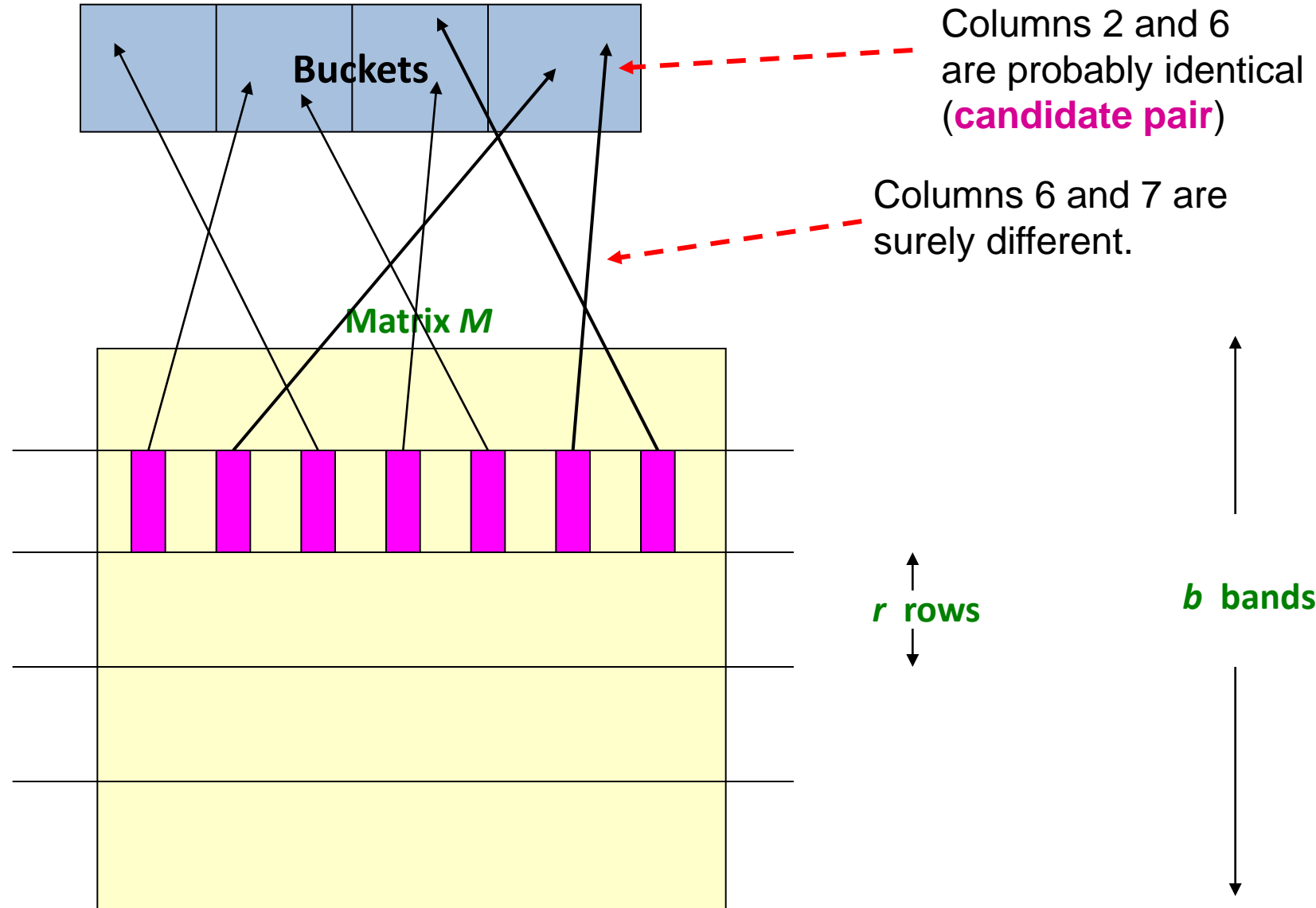
One
signature

Signature matrix M

Partition M 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
 - 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



Example of Bands

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

Assume the following case:

- Suppose 100,000 columns of \mathbf{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

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

- 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**

C_1, C_2 are 30% Similar

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

- 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 Involves a Tradeoff

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

- **Pick:**
 - The number of Min-Hashes (rows of \mathbf{M})
 - The number of bands \mathbf{b} , and
 - The number of rows \mathbf{r} per bandto 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

b bands, 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 = $1 - (1 - t^r)^b$

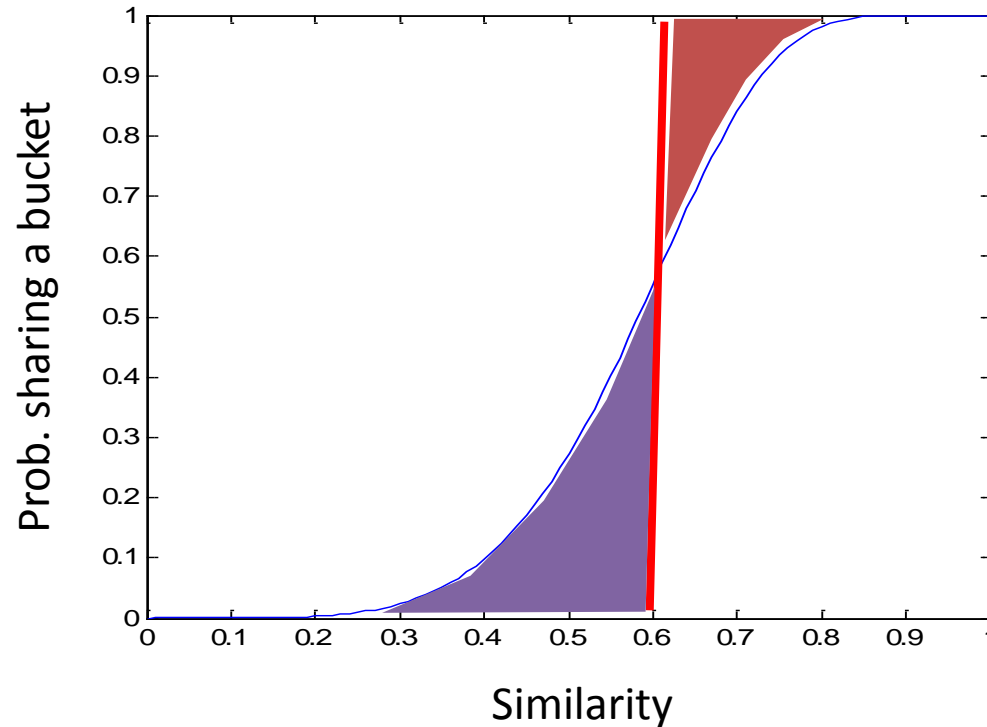
Example: $b = 20$; $r = 5$

- 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

- 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

- **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 $\geq s$

Thank you.