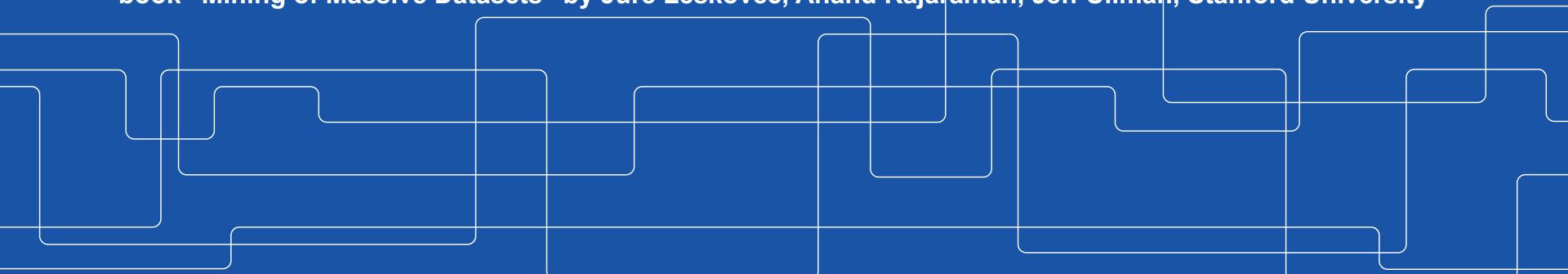




# Finding Similar Items: Locality Sensitive Hashing

Vladimir Vlassov

Acknowledgement: Some of the slides are adopted from the slide provided at <http://www.mmds.org> for the book “Mining of Massive Datasets” by Jure Leskovec, Anand Rajaraman, Jeff Ullman, Stanford University





# Outline

- Applications
- Shingling
- Min-hashing
- LSH: Locality-Sensitive Hashing
- Distance Measures



# A Fundamental Data-Mining Problem

***Examine data for “similar” items***

based on some similarity or distance measure, such as Jaccard distance/similarity, Euclidean distances, cosine distance, edit distance, Hamming distance



# A Fundamental Data-Mining Problem (cont.)

## *Find “similar” items*

*Given*

- high-dimensional data points  $\mathbf{x}_1, \mathbf{x}_2, \dots$ ;
- a distance function  $d(\mathbf{x}_i, \mathbf{x}_j)$  that quantifies the “distance” between  $\mathbf{x}_i$  and  $\mathbf{x}_j$ ;
- a distance threshold  $s$  – the max size of near neighbors

*Find all pairs of data points  $(\mathbf{x}_i, \mathbf{x}_j)$  that are within the distance threshold:  $d(\mathbf{x}_i, \mathbf{x}_j) \leq s$ , i.e., at most  $s$  apart from each other*

- Naïve solutions is  $O(N^2)$
- Some optimization allows  $O(N)$



# Jaccard Similarity

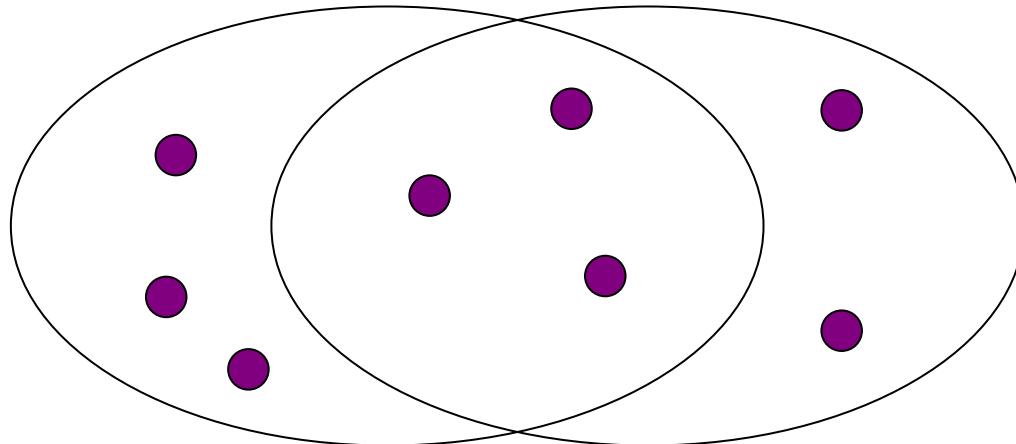
Many data-mining problems can be expressed as finding  
**“similar” sets**

**The Jaccard similarity of two sets**, C1 and C2, is the fraction of common items, i.e., the fraction of their intersection – size of their intersection divided by the size of their union:

$$\text{sim}(C1, C2) = |C1 \cap C2| / |C1 \cup C2|$$

**Jaccard distance**:  $d(C1, C2) = 1 - |C1 \cap C2| / |C1 \cup C2|$

# Jaccard Similarity



3 in intersection

8 in union

Jaccard similarity = 3/8

Jaccard distance = 5/8



# Similar Documents

Given a body of documents, e.g., the Web, find pairs of ***textually similar documents*** with a lot of text in common (***near duplicate pairs***)

## Examples:

- ***Mirror sites***, or approximate mirrors
  - quite similar, but are rarely identical
  - Don't want to show both in a search result.
- ***Plagiarism***, including large quotations.
- ***Similar news articles*** at many news sites.
  - Articles from the same source, e.g. the Associated Press
  - Cluster articles by “same story”



# Collaborative filtering

Find pairs of “**similar**” **customers** with similar tastes – having similar “baskets” with rather high Jaccard similarity ( $\geq 20\%$ )

*Dual*: Find pairs of “**similar**” **products** bought by similar sets of customers

Combine similarity-finding with clustering to group mutually-similar products.

- A more powerful notion of customer-similarity by asking whether they made purchases within many of the same groups.

**Example**: NetFlix users with similar tastes in movies, for recommendation systems. (*Dual*: movies with similar sets of fans).



# Finding Similar Documents

## Problems:

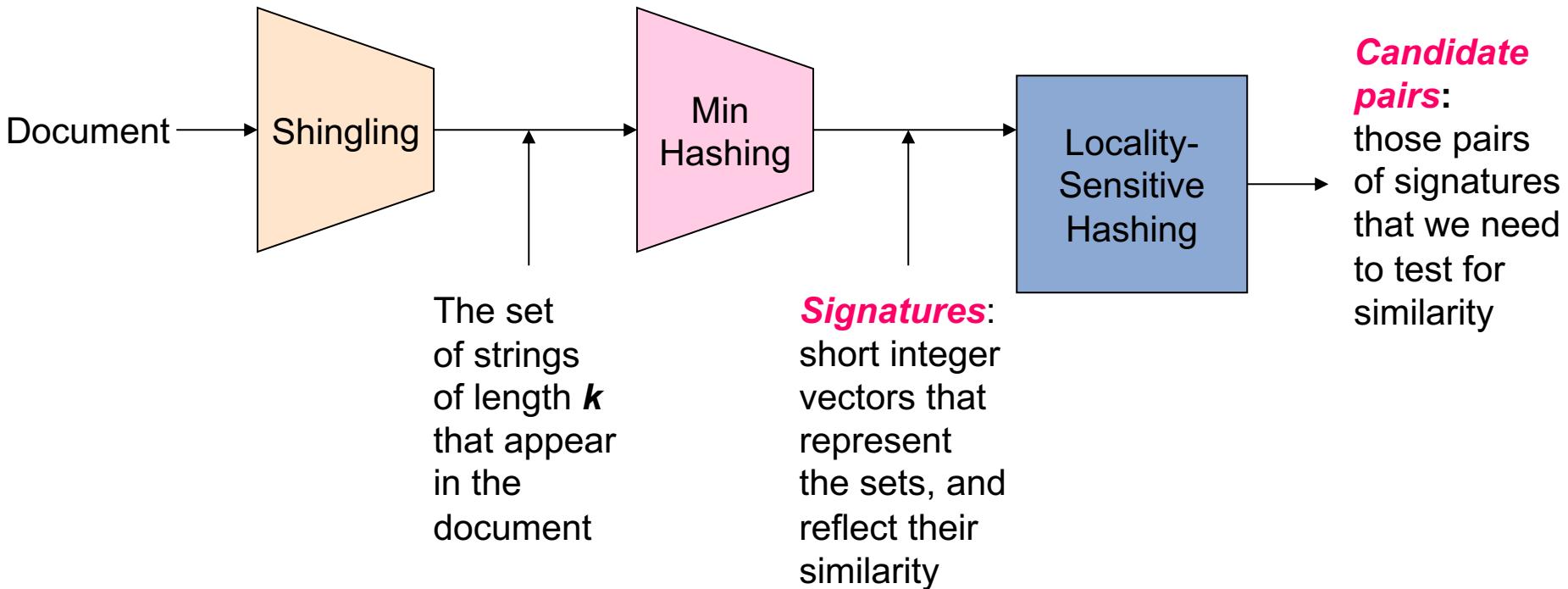
- Common text, not common topic.
- Many small pieces of one document may appear out of order in another.
  - Special cases are easy, e.g., identical documents, or one document contained in another
- *Too many documents* to compare all pairs, or *too many pairs*
- Documents are so large or so many that they *cannot fit in main memory*



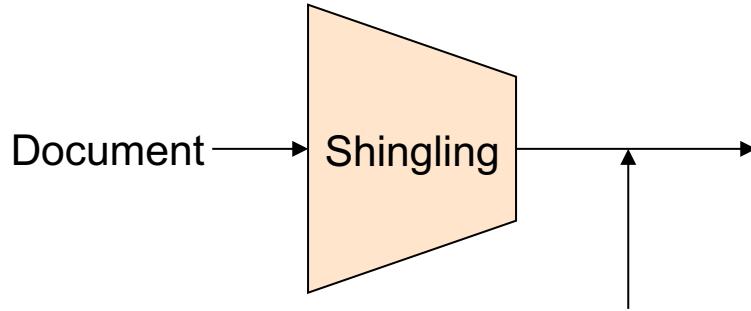
# 3 Essential Techniques for Similar Documents

1. ***Shingling***: Convert documents to sets.
2. ***Minhashing***: Convert large sets to short signatures, while preserving similarity.
3. ***Locality-sensitive hashing***: Focus on pairs of signatures likely to be from similar documents
  - Candidate pairs

# The Big Picture



## Step 1: *Shingling*: Convert documents to sets



The set  
of strings  
of length  $k$   
that appear  
in the  
document



# Documents as Sets

**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!***

# Define: Shingles

A  **$k$ -shingle** (or  **$k$ -gram**) for a document is a sequence of  $k$  tokens that appears in the document

- 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:  $S(D_1) = \{\text{ab}, \text{bc}, \text{ca}\}$

- **Option:** Shingles as a bag (multiset), count ab twice:  
 $S'(D_1) = \{\text{ab}, \text{bc}, \text{ca}, \text{ab}\}$

# Similarity Metric for Shingles

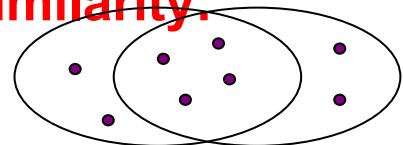
**Document D is a set of its k-shingles C=S(D)**

Equivalently, each document is a 0/1 vector in the space of  $k$ -shingles

- Each unique shingle is a dimension – an element of the vector representing the document  
(1 – the shingle is in the document; 0 – it is not)
- Vectors are very sparse

**A natural similarity measure is the Jaccard similarity:**

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



# Working Assumption and Observation

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

- Changing a word only affects  $k$ -shingles within distance  $k$  from the word.
- Reordering paragraphs only affects the  $2k$  shingles that cross paragraph boundaries.
- **Example:**  $k = 3$ , “The dog *which* chased the cat” versus “The dog *that* chased the cat”.
  - Only 3-shingles replaced are *g\_w*, *\_wh*, *whi*, *hic*, *ich*, *ch\_*, *h\_c*.

# Shingle Size

**Caveat:**  $k$  should be large enough, or most documents will have most shingles.

In other words,  $k$  should be large enough that the probability of any given shingle appearing in any given document is low.

- $k = 5$  is OK for short documents, e.g. emails
- $k = 9, 10$  is better for long documents



# Example

Find similarity of  $D_1 = \text{"editorial"}$  and  $D_2 = \text{"factorial"}$

$k=1$

$$\text{sim}(\{e, d, i, t, o, r, a, l\}, \{f, a, c, t, o, r, i, l\}) = 6/10 = 0.6$$

$k=5$

$$\text{sim}(\{\text{edito, ditor, itori, toria, orial}\}, \{\text{facto, actor, ctori, toria, orial}\}) = 2 / 8 = 0.25$$

$k=9$

$$\text{sim}(\{\text{editorial}\}, \{\text{factorial}\}) = 0 / 2 = 0$$



# Compressing Shingles

To **compress long shingles**, one can **hash** them to 4B to fit in integer

**Example:**  $k=2$ ; document  $D1 = \text{abcab}$

Set of 2-shingles:  $S(D1) = \{ \text{ab}, \text{bc}, \text{ca} \}$

Hash the singles:  $h(D1) = \{ 1, 5, 7 \}$

**Represent a document by the set of hash values of its  $k$ -shingles**

- For instance, one could construct the set of 9-shingles for a document and then map each of those 9-shingles to a bucket number in the range 0 to  $2^{32} - 1$ .
  - Each shingle is represented by 4B integer instead of 9B string.



# Motivation for Minhash/LSH

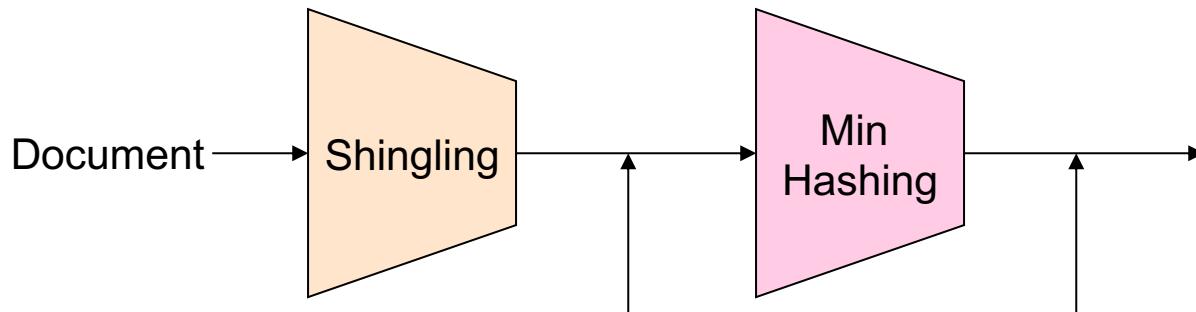
Suppose we need to find near-duplicate documents among  $N = 1$  million documents

Naïvely, we would have to compute pairwise Jaccard similarities for every pair of docs – complexity  $O(N^2)$

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

## Step 2: *Minhashing*: Convert large sets to short signatures, while preserving similarity



The set  
of strings  
of length  $k$   
that appear  
in the  
document

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



# Basic Data Model: Sets

Many similarity problems can be formalized as ***finding subsets of some universal set that have significant intersection.***

Examples include:

1. Documents represented by their sets of shingles (or hashes of those shingles).
2. Similar customers or products.

# Remind: Jaccard Similarity of Sets

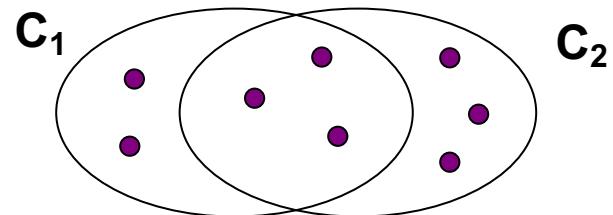
The **Jaccard similarity** of two sets is the size of their intersection divided by the size of their union.

$$\text{Sim}(\mathbf{C}_1, \mathbf{C}_2) = |\mathbf{C}_1 \cap \mathbf{C}_2| / |\mathbf{C}_1 \cup \mathbf{C}_2|$$

$$\text{Jaccard distance: } d(\mathbf{C}_1, \mathbf{C}_2) = 1 - \text{Sim}(\mathbf{C}_1, \mathbf{C}_2)$$

## Example

- Size of intersection = 3; size of union = 8,
- Jaccard similarity:  $\text{sim}(\mathbf{C}_1, \mathbf{C}_2) = 3/8$
- Distance:  $d(\mathbf{C}_1, \mathbf{C}_2) = 1 - \text{sim}(\mathbf{C}_1, \mathbf{C}_2) = 5/8$



# From Sets to Boolean Matrices

Visualise a collection of sets as *characteristic matrix*

**Rows** correspond to elements (shingles) of the universal set.

**Columns** correspond to sets (documents)

- 1 in row  $e$  and column  $S$  if and only if  $e$  is a member of  $S$ , i.e.  $e \in S$

*Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)*

- Can be computed using bitwise AND (for intersection) and bitwise OR (for union)
- **Typical matrix is sparse!**

	Documents			
Shingles	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	

# Example: Jaccard Similarity of Columns (Sets)

	C <sub>1</sub>	C <sub>2</sub>	
s <sub>1</sub>	0	1	*
s <sub>2</sub>	1	0	*
s <sub>3</sub>	1	1	*
s <sub>4</sub>	0	0	
s <sub>5</sub>	1	1	*
s <sub>6</sub>	0	1	*

$$\text{Sim}(C_1, C_2) = 2 / 5 = 0,4$$

Encode sets using 0/1 (bit, Boolean) vectors  
Interpret set intersection as bitwise AND, and  
set union as bitwise OR

$$C_1 = 011010; C_2 = 101011$$

Size of intersection = 2; size of union = 5,  
Jaccard similarity (not distance)  $\text{sim}(C_1, C_2) = 2/5$   
Distance:  $d(C_1, C_2) = 1 - (\text{Jaccard similarity}) = 3/5$

# Four Types of Rows

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

$C_1$	$C_2$		
1	1	<i>a</i>	<i>1 in both columns</i>
1	0	<i>b</i>	<i>columns are different</i>
0	1	<i>c</i>	
0	0	<i>d</i>	<i>0 in both columns</i>

Denote,  $A$  = the number of rows of type *a*;

$B$  = the number of rows of type *b*, etc.

$$\text{Sim } (C_1, C_2) = A / (A + B + C).$$



# Outline: Finding Similar Columns

## So far:

- Documents → Sets of shingles
- Represent sets as Boolean column-vectors in a matrix

**Next goal:** Find similar columns while computing *small signatures*

- **Similarity of columns == similarity of signatures**



# Outline: Finding Similar Columns (cont)

1. Compute ***signatures of columns*** = small summaries of columns.
2. Examine ***pairs of signatures*** to find similar signatures.
  - **Essential:** Similarities of signatures and columns are related.
3. Optional: check that columns with similar signatures are really similar.

## Warnings:

- Comparing all 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)



# MinHashing

Let  $h$  be a hash function that maps the members of a set  $S$  to distinct integers.

MinHash  $h_{\min}(S)$  is the member  $e \in S$  with the minimum value of  $h(e)$

If for two sets  $S_1$  and  $S_2$ ,  $h_{\min}(S_1) = h_{\min}(S_2) = e$ , then  $e \in S_1 \cap S_2$

The probability of this to be true is  $|S_1 \cap S_2| / |S_1 \cup S_2|$ , i.e. the **Jaccard similarity**:

$$\Pr[ h_{\min}(S_1) = h_{\min}(S_2) ] = \text{Sim}(S_1, S_2)$$

# MinHash Signature of a Set

## Key idea:

- Take  $k$  (e.g.  $k = 100$ ) independent hash functions, e.g.,  
$$h(x) = (ax + b) \% c$$
- Apply the functions to the elements (the row numbers with value 1), to compute a vector of  $k$  minHash values for a set  $S$ .

*The resulting column-vector of  $k$  minHash values is a signature of the set  $S$*

# MinHashing using Permutations

- Permute the rows.
- Define ***minhash function*** for this permutation,  $h(C)$  = the number of the first (in the permuted order) row in which column C has 1.
- Apply, to all columns, several (e.g., 100) randomly chosen permutations to create a ***signature*** for each column.
- Result is a ***signature matrix***: columns = sets (documents), rows = minhash values, in order for that column.

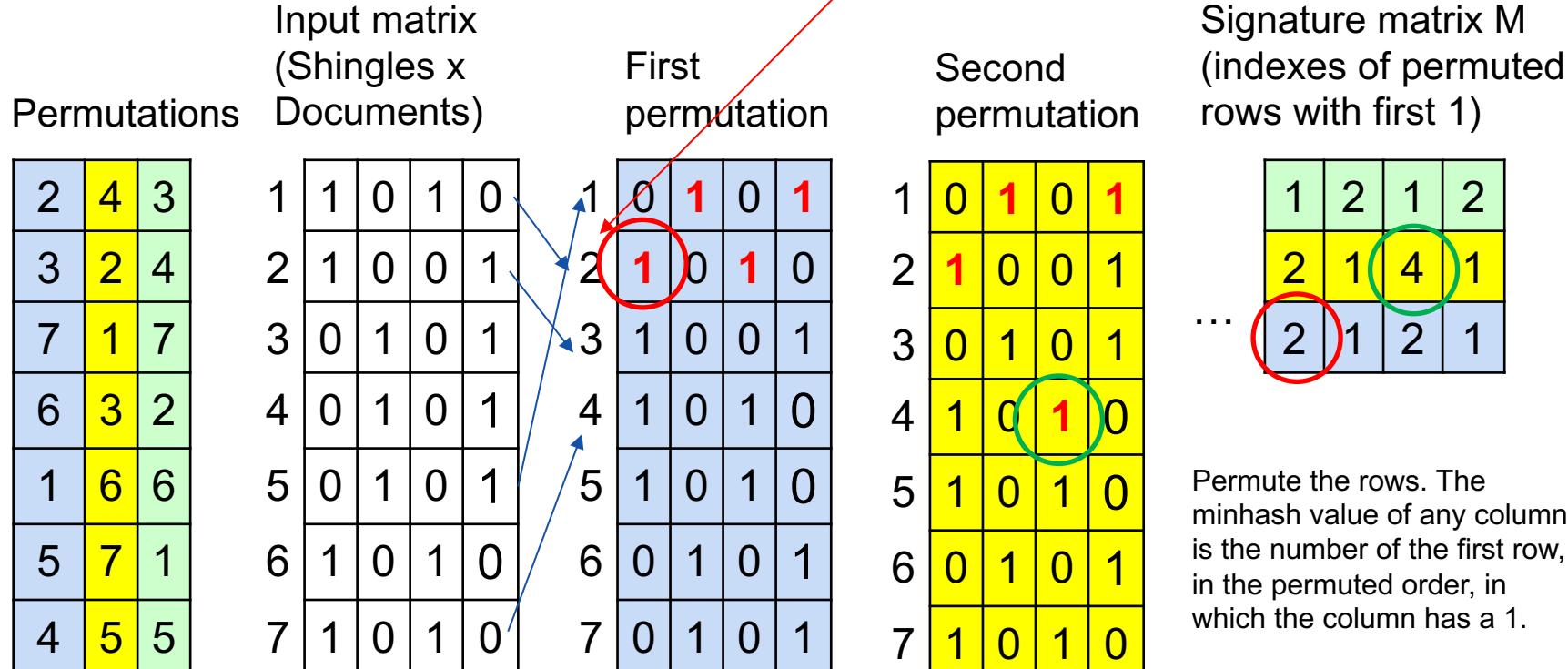
***Using several permutations is equivalent to using several hash functions of the type  $h(x) = (ax + b)\%c$***



# Signature Matrix

Thus, we can form from characteristic matrix  $\mathbf{M}$  a ***signature matrix***, in which the  $i$ -th column of  $\mathbf{M}$  is replaced by the minhash signature for (the set of) the  $i$ -th column.

# Minhashing – Example





# Similarity of Signatures

The ***similarity of two signatures*** is the fraction of the minHash functions in which they agree

$$\Pr[ h_{\min}(S_1) = h_{\min}(S_2) ]$$

- i.e. the number of rows in the signature matrix with the same values in  $S_1$  and  $S_2$  columns divided by the total number of rows in the signature ( $k$ ).

For two sets  $S_1$  and  $S_2$ , ***the similarity of two signatures is an estimate of the Jaccard similarity*** of the two sets.

$$\Pr[ h_{\min}(S_1) = h_{\min}(S_2) ] \approx \text{Sim}(S_1, S_2)$$

# Similarity of Signatures (cont)

- The ***similarity of signatures*** is the fraction of the minhash functions (rows) in which they agree.
- Thus, the expected similarity of two signatures equals the Jaccard similarity of the columns or sets that the signatures represent.
  - And the longer the signatures, the smaller will be the expected error.

# Similarity of Signatures – Example

Permutations	Input matrix (Shingles x Documents)	Signature matrix M (indexes of permuted rows with first 1)																																																																				
<table border="1"> <tr><td>2</td><td>4</td><td>3</td></tr> <tr><td>3</td><td>2</td><td>4</td></tr> <tr><td>7</td><td>1</td><td>7</td></tr> <tr><td>6</td><td>3</td><td>2</td></tr> <tr><td>1</td><td>6</td><td>6</td></tr> <tr><td>5</td><td>7</td><td>1</td></tr> <tr><td>4</td><td>5</td><td>5</td></tr> </table>	2	4	3	3	2	4	7	1	7	6	3	2	1	6	6	5	7	1	4	5	5	<table border="1"> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>3</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>4</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>5</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>6</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>7</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> </table>	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	<table border="1"> <tr><td>1</td><td>2</td><td>1</td><td>2</td></tr> <tr><td>2</td><td>1</td><td>4</td><td>1</td></tr> <tr><td>2</td><td>1</td><td>2</td><td>1</td></tr> </table>	1	2	1	2	2	1	4	1	2	1	2	1
2	4	3																																																																				
3	2	4																																																																				
7	1	7																																																																				
6	3	2																																																																				
1	6	6																																																																				
5	7	1																																																																				
4	5	5																																																																				
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																																																																		
1	2	1	2																																																																			
2	1	4	1																																																																			
2	1	2	1																																																																			

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0,75	0,75	0	0
Sig/Sig	0,67	1	0	0

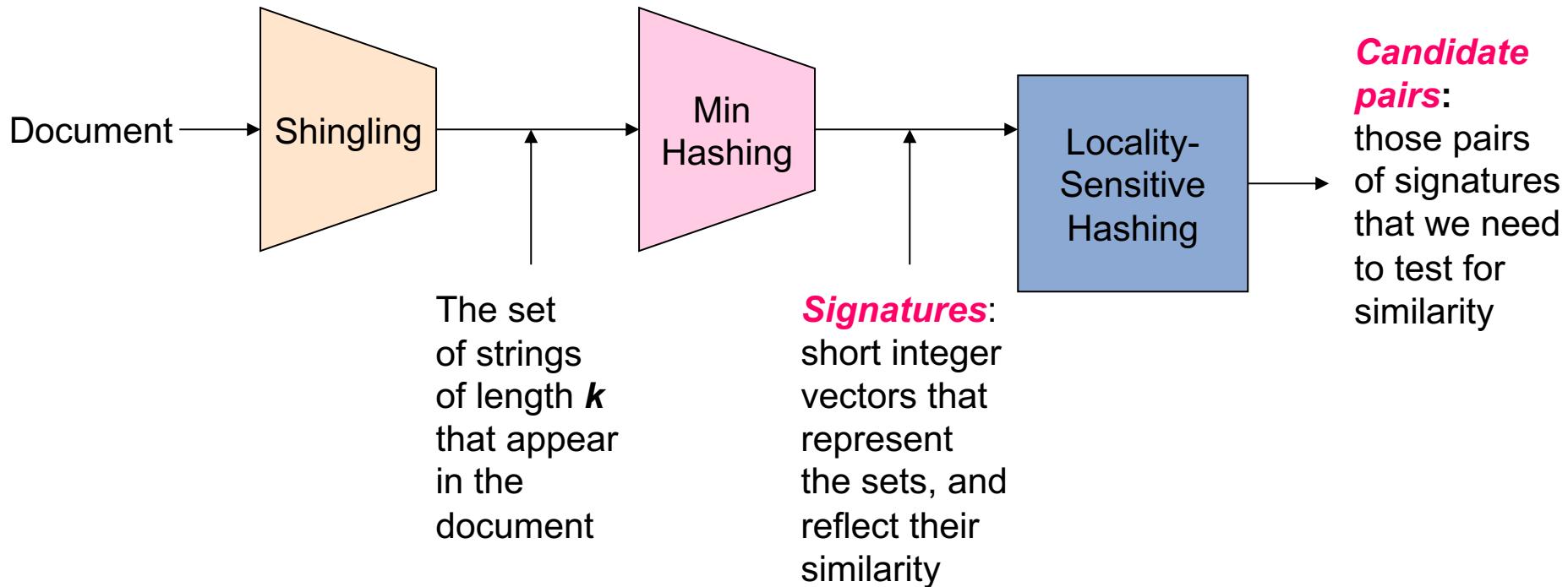
# Scalability Issue with Permutations

- Suppose 1 billion rows (shingles).
- Hard to pick a random permutation of 1...billion.
- Also, representing a random permutation requires 1 billion entries.
- And accessing rows in permuted order may lead to thrashing.
  - Better yet to use several hash functions, say, 100, to “generate” 100 “permutations” of rows and compute minimal hash value for elements (rows with 1) of each column corresponding to a set

# Implementation

```
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  $h_i(r)$  is smaller than  $M(i, c)$  then
                     $M(i, c) := h_i(r)$ ;
end;
```

# Step 3: Locality-Sensitive Hashing





# Scalability Issue

Assume we have built a signature matrix

Need to compare signatures (columns) in pairs for similarity –  $O(N^2)$

Checking all pairs is hard

Example:  $10^6$  columns implies ( $10^6$  by 2)

$\approx 5 \times 10^{11}$  column-comparisons.

At 1 microsecond/comparison: 6 days.

$$\frac{n!}{k!(n-k)!}$$



# LSH: Locality-Sensitive Hashing

- **General idea:** Generate from the collection of all elements (signatures in our example) a small list of ***candidate pairs***: pairs of elements whose similarity must be evaluated.
- **For signature matrices:** Hash columns to many buckets and make elements of the same bucket candidate pairs.



# Candidate Generation from Minhash Signatures

- Pick a similarity threshold  $s$  ( $0 < s < 1$ ).
- We want a pair of columns  $c$  and  $d$  of the signature matrix  $M$  to be a ***candidate pair*** if and only if their signatures agree in at least fraction  $s$  of the rows.
  - I.e.,  $M(i, c) = M(i, d)$  for at least fraction  $s$  values of  $i$ .

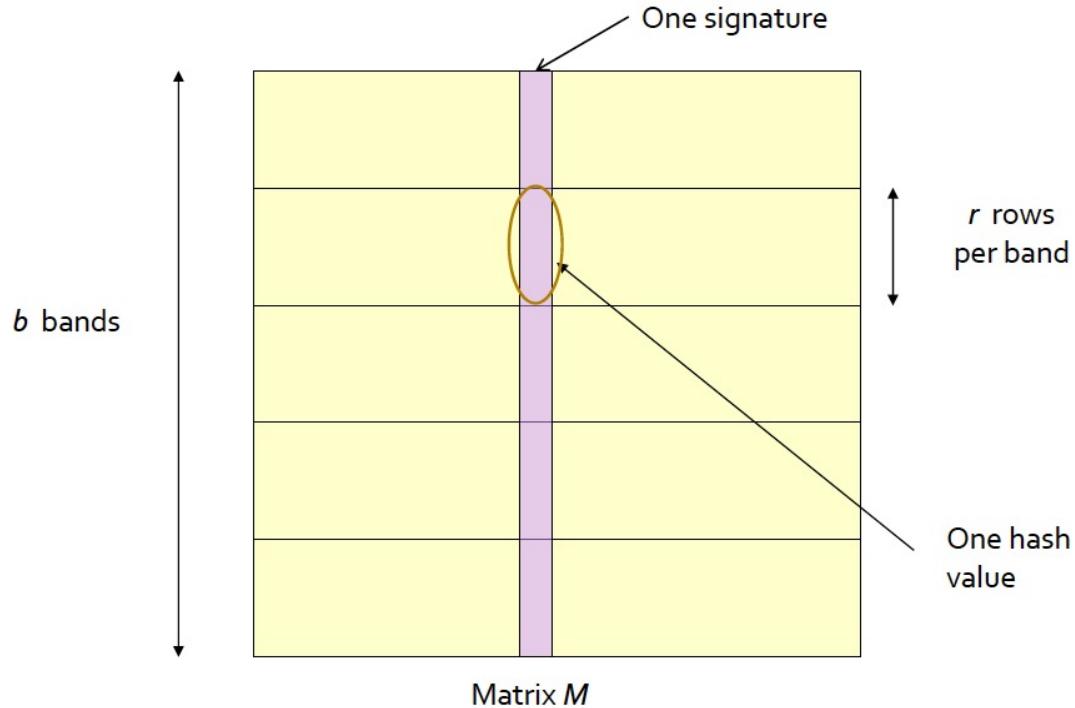


# LSH for MinHash Signatures

**Big idea:** hash columns of signature matrix  $M$  several times.  
Arrange that (only) similar columns are likely to hash to the same bucket.

***Candidate pairs*** are those that hash ***at least once*** to the ***same bucket***.

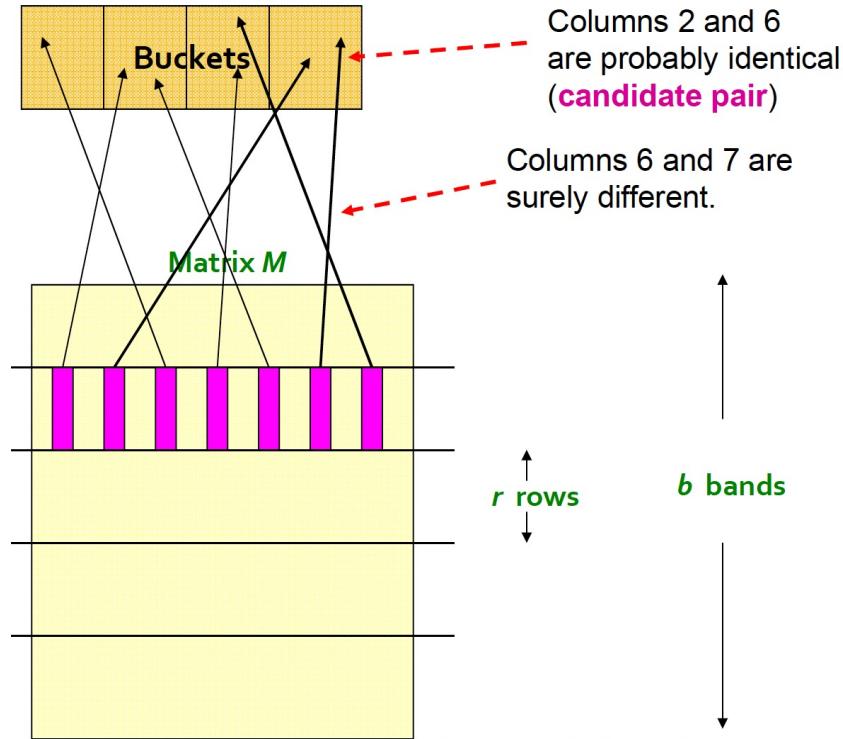
# Partition into $b$ Bands



# Partition into $b$ Bands (cont)

- 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  $\geq 1$  band.
- Tune  $b$  and  $r$  to catch most similar pairs, but few non-similar pairs.

# Hashing Bands





# Simplifying Assumption

- 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



# Example: Bands

- Suppose 100,000 columns.
- Signatures of 100 integers.
- Therefore, signatures take 40Mb.
  - They fit easily into main memory.
- Want all 80%-similar pairs of documents.
- 5,000,000,000 pairs of signatures can take a while to compare.
- Choose 20 bands of 5 rows/band.

# Suppose $C_1$ and $C_2$ are 80% Similar

- Find pairs of at least  $s = 0.8$  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 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 underlying sets are **false negatives** (we miss them)
  - **We would find 99,965% pairs of truly similar documents**

# Suppose $C_1$ and $C_2$ are 30% Similar

- Find pairs of at least  $s = 0.8$  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 any one band:**  $(0.3)^5 = 0.0243$ .
- Probability  $C_1, C_2$  identical in at least 1 of 20 bands:  
 $1 - (1 - 0.0243)^{20} = 0.0474$ 
  - i.e., approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs (**false positives**)
    - 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  $s$

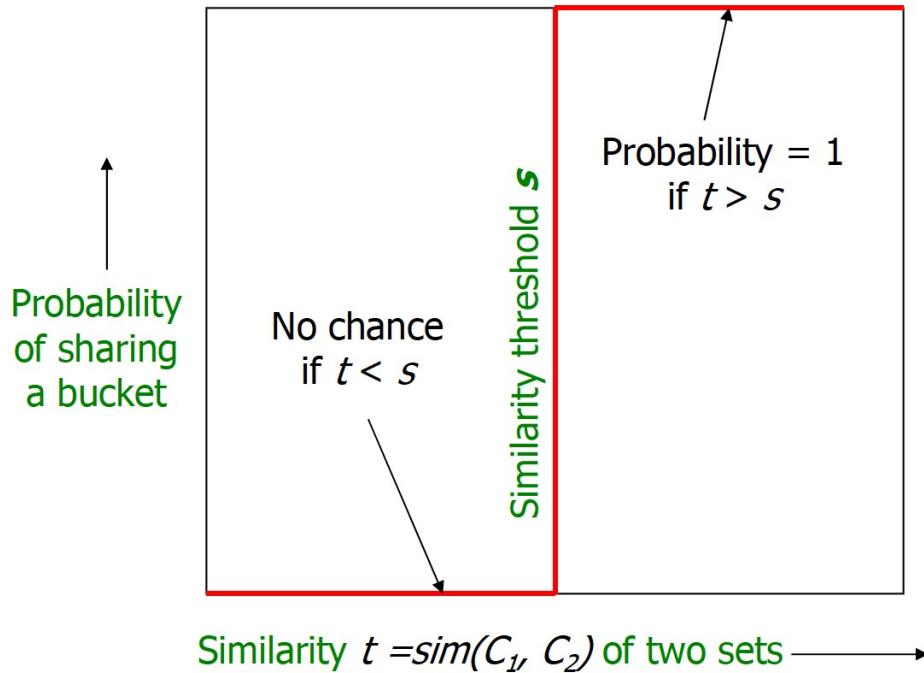


# LSH Involves a Tradeoff

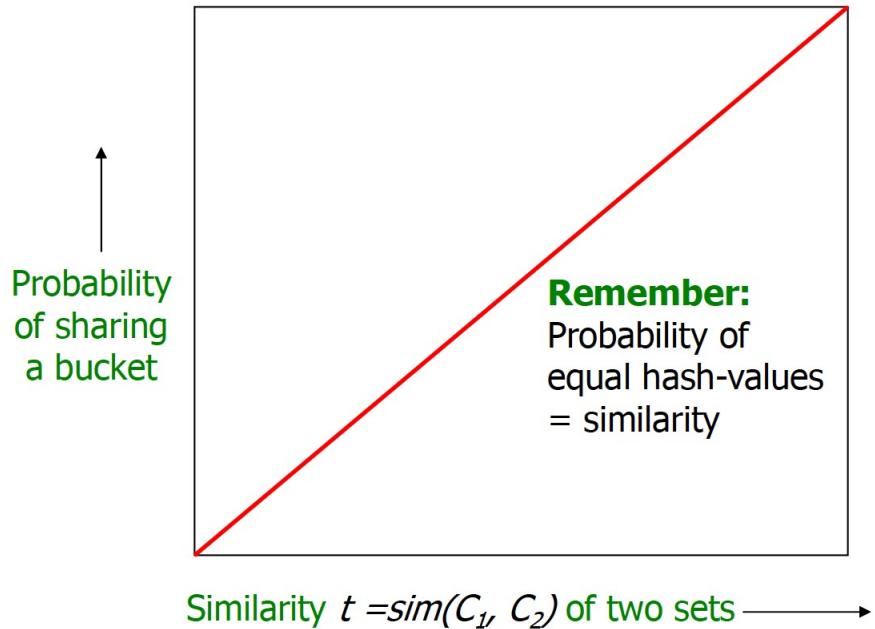
Pick:

- The number of Min-Hashes (signature length)
- 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 (missed candidate pairs) would go up

# Analysis of LSH – What we want



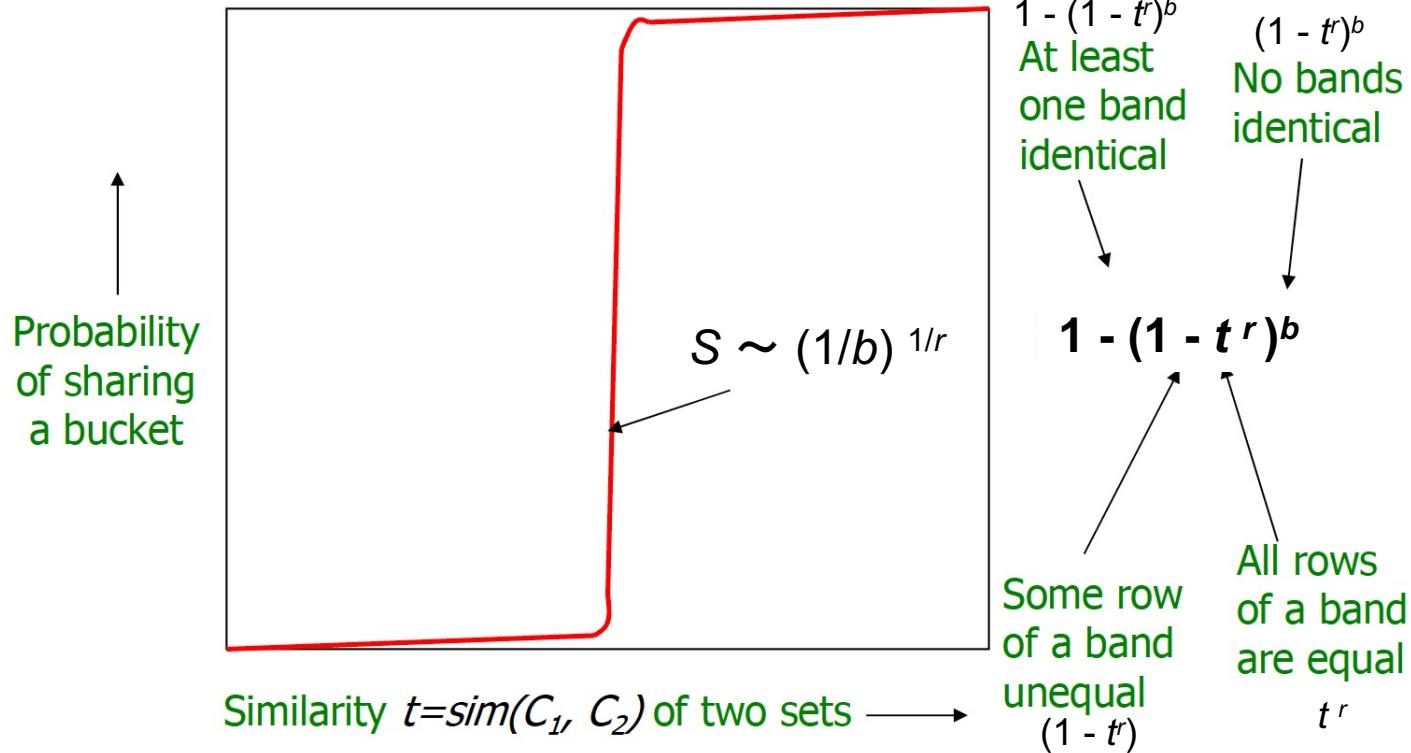
# What 1 Band of 1 Row Gives You



# **$b$ Bands Each of $r$ Rows**

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

# What $b$ Bands of $r$ Rows Gives You



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

Probability that at least 1 band is identical:

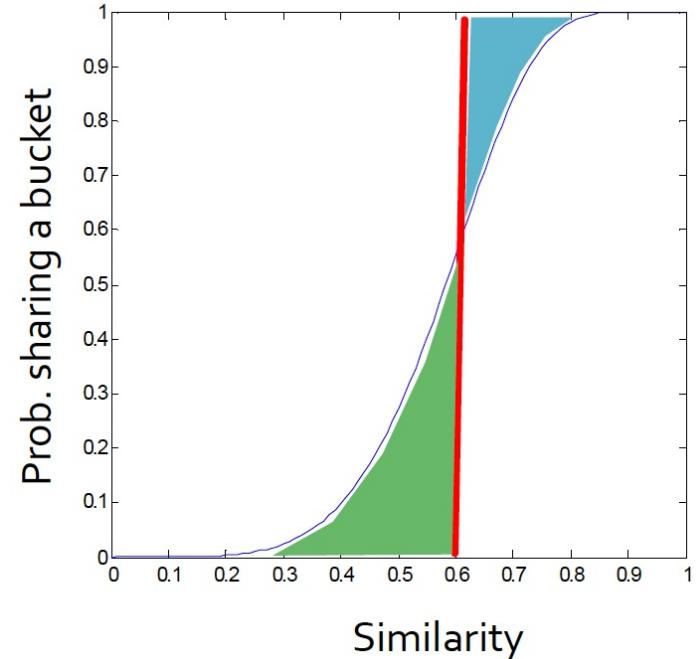
$t$	$1 - (1 - t^r)^b$
0,2	0,006
0,3	0,047
0,4	0,186
0,5	0,470
0,6	0,802
0,7	0,975
0,8	0,9996

$$s \approx (1/b)^{1/r} = 0,549$$

# Picking $r$ and $b$ : The S-curve

- To find  $b$  and  $r$  for given number of hash-functions  $n$  and sim. threshold  $t$ , solve  

$$b \cdot r = n; (1/b)^{1/r} \approx t$$
- Picking  $r$  and  $b$  to get the best **S-curve**
  - e.g. 50 hash-functions ( $b = 10, r = 5$ )
- If avoidance of false negatives is important, select  $b$  and  $r$  to produce a **threshold lower than  $t$** ;
- if speed is important and to **limit false positives**, select  $b$  and  $r$  to produce a **higher threshold**.



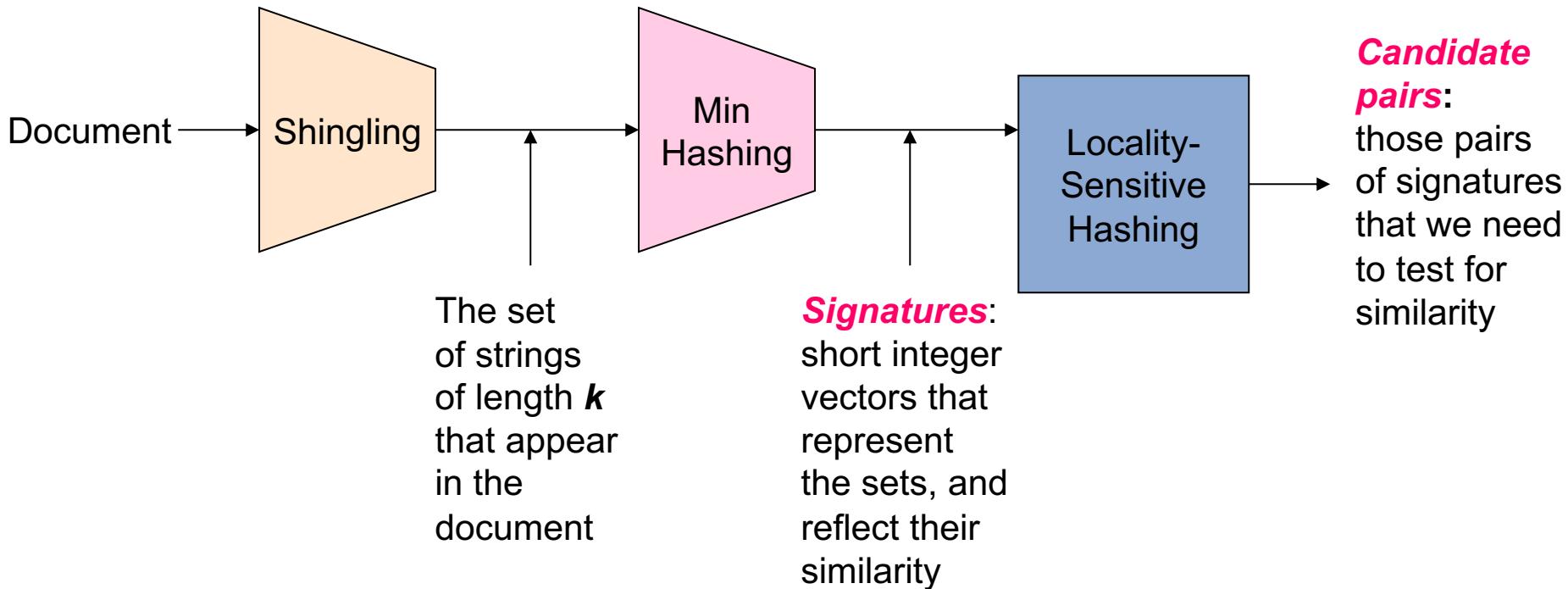
**Blue area:** False Negative rate  
**Green area:** False Positive rate



# LSH Summary

- Tune  $r$  and  $b$  to get almost all pairs with similar signatures but eliminate most pairs that do not have similar signatures.
- Check that candidate pairs really do have similar signatures.
- **Optional:** In another pass through data, check that the remaining candidate pairs really represent similar sets .

# Combining the Techniques



# Combining the Techniques

1. **Shingling:** Pick  $k$  and construct from each document the **set of  $k$ -shingles**. Optionally, hash the  $k$ -shingles to shorter bucket numbers.
2. Sort the document-shingle pairs to order them by shingle.
3. **MinHashing:** Pick a length  $n$  for the minhash signatures. Compute the **minhash signatures for all the documents** (see the algorithm in Section 3.3.5).

# Combining the Techniques (cont)

4. **LSH**: Choose a similarity threshold  $t$  to classify a pair of sets (signatures) as a desired “*similar pair*”  
Pick the number of bands  $b$  and the number of rows  $r$  such that  $br = n$ , and the threshold  $t \simeq (1/b)^{1/r}$ .  
**Tradeoff**: To avoid *false negatives* select  $b$  and  $r$  to produce a threshold lower than  $t$ ; if speed is important and you wish to limit *false positives*, select  $b$  and  $r$  to produce a higher threshold.
5. Find *candidate pairs* by applying LSH (Section 3.4.1).



# Combining the Techniques (cont)

6. Check each candidate pair's signatures if the **fraction of components in which they agree is at least  $t$** , i.e. if they are similar at least  $t$
7. **Optionally**, if the signatures are sufficiently similar, check if the corresponding documents are truly similar.



# Distance Measures

Generalized LSH is based on some kind of “*distance*” between points.

Similar points are “*close*.”



# Axioms of a Distance Measure

$d$  is a distance measure if it is a function from pairs of points to real numbers such that:

1.  $d(x,y) \geq 0$ .
2.  $d(x,y) = 0$  iff  $x = y$ .
3.  $d(x,y) = d(y,x)$ .
4.  $d(x,y) \leq d(x,z) + d(z,y)$  (*triangle inequality*).



# Euclidean Distances

In an  $n$ -dimensional Euclidean space, points are *vectors of  $n$  real numbers*.

The conventional distance  $L_2$ -norm

$$d([x_1, x_2, \dots, x_n], [y_1, y_2, \dots, y_n]) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$



# Jaccard Distance

$$d(x, y) = 1 - \text{SIM}(x, y)$$

The Jaccard distance is 1 minus the ratio of the sizes of the intersection and union of sets  $x$  and  $y$ .



# Cosine Distance

- Used in spaces with dimensions, where points may be thought of as directions.
- The cosine distance between two points is ***the angle*** that the vectors to those points make.

To calculate the cosine distance:

1. compute the cosine of the angle
2. apply the arc-cosine function to translate to an angle in the 0-180 degree range.



# Edit Distance

Edit distance between two strings  $x$  and  $y$  is the smallest ***number of insertions and deletions*** of single characters that will convert  $x$  to  $y$ .

The edit distance  $d(x, y) = \text{length}(x) + \text{length}(y) - 2 * \text{length}(\text{LCS})$

- Here LCS is a longest common subsequence of  $x$  and  $y$ ;
- to find LCS, delete characters from  $x$  and  $y$  until they are identical.



# Hamming Distance

The Hamming distance between two vectors in a vector space is the number of components in which they differ.



# Distance Measures

Generalized LSH is based on some kind of “*distance*” between points.

Similar points are “*close*.”

# Locality-Sensitive Functions – General Definition

Generalized LSH is based on some kind of “**distance**” between points. Similar points are “**close**.”

Let  $d_1 < d_2$  be two distances according to some distance measure  $d$ .

A family  $\mathcal{F}$  of hash functions is said to be **( $d_1, d_2, p_1, p_2$ )-sensitive** if for every  $f$  in  $\mathcal{F}$ :

1. If  $d(x, y) \leq d_1$ , then the probability that  $f(x) = f(y)$  is at least  $p_1$ .
2. If  $d(x, y) \geq d_2$ , then the probability that  $f(x) = f(y)$  is at most  $p_2$ .

