CSE 5243 INTRO. TO DATA MINING

Locality Sensitive Hashing (LSH)
Review, Proof, Examples

Huan Sun, CSE@The Ohio State University

Slides adapted from Prof. Jiawei Han @UIUC, Prof. Srinivasan Parthasarathy @OSU

MMDS Secs. 3.2-3.4.

Slides adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

FINDING SIMILAR ITEMS

Slides also adapted from Prof. Srinivasan Parthasarathy @OSU

Two Essential Steps for Similar Docs

1. ***Shingling:*** Convert documents to sets
2. ***Min-Hashing:*** Convert large sets to short signatures, while preserving similarity

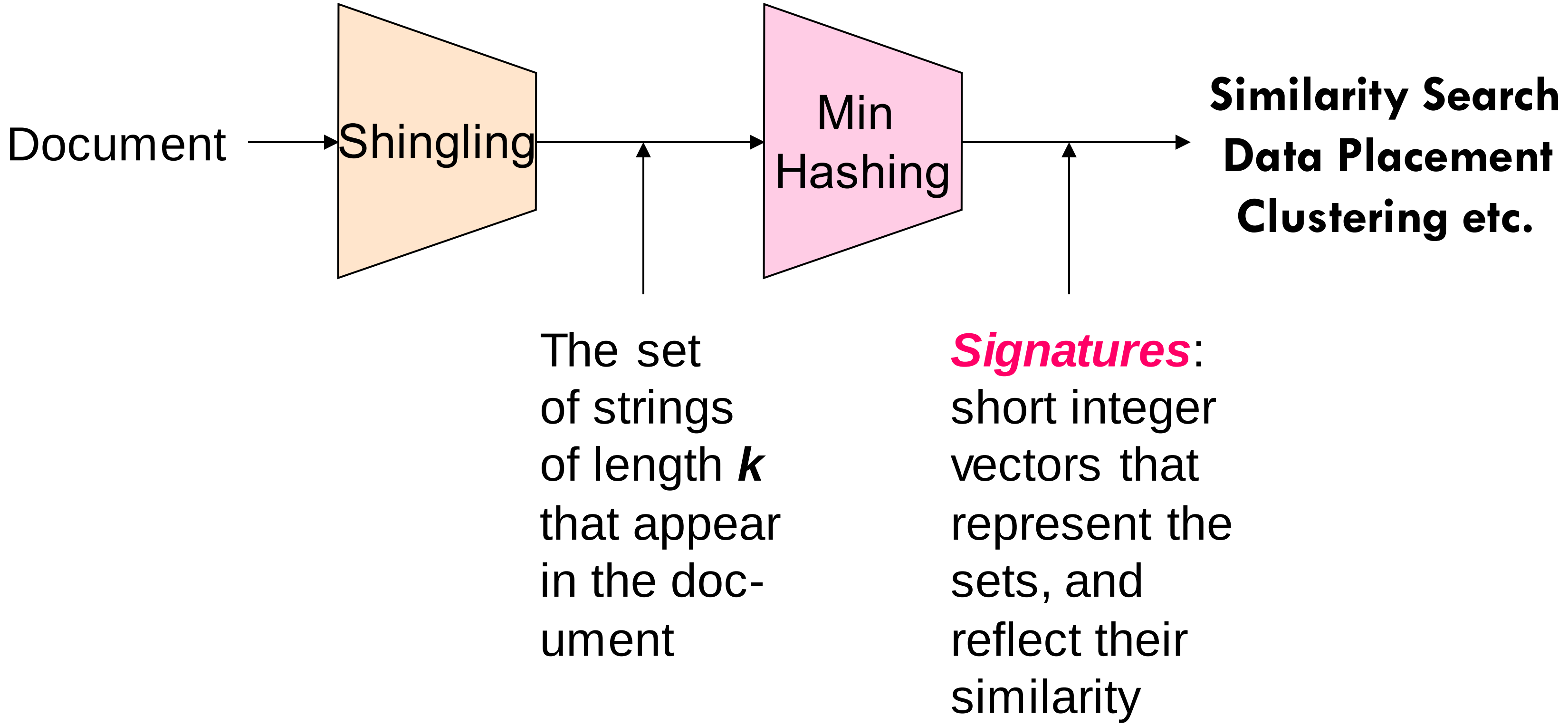
Host of follow up applications

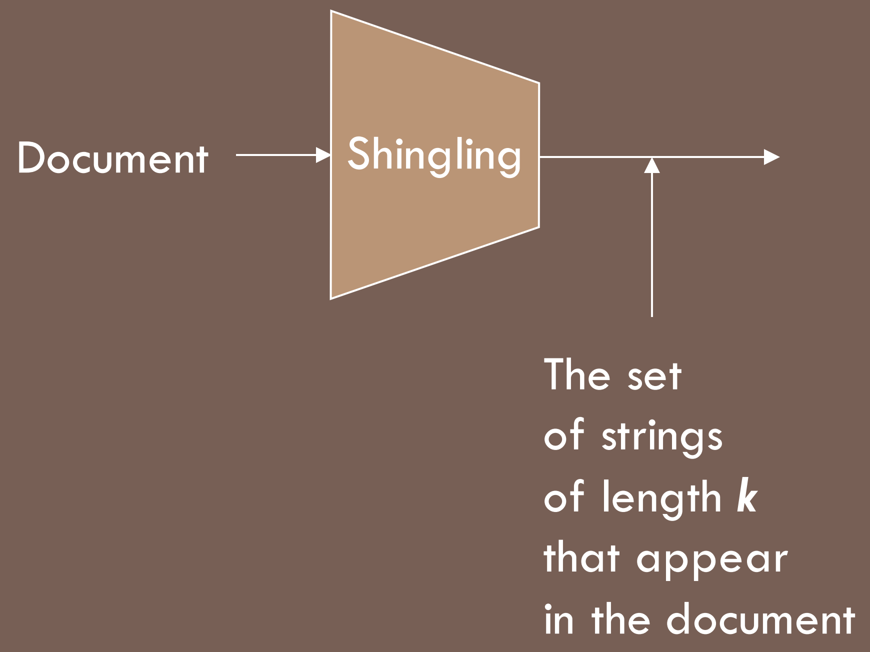
e.g. Similarity Search

Data Placement

Clustering etc.

The Big Picture





SHINGLING

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

Define: Shingles

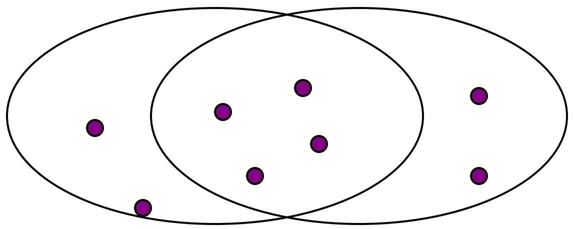
- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples

- **Example: k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}

Similarity Metric for Shingles

- **Document D_1 is a set of its 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:**

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



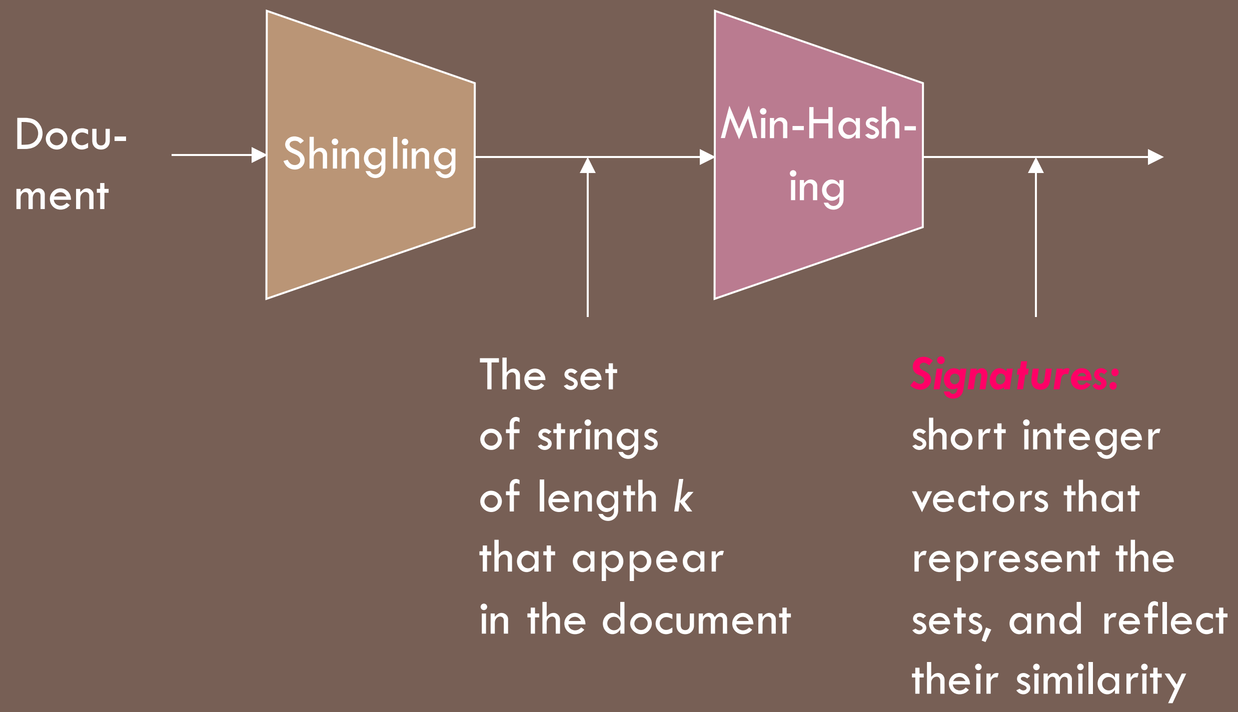
Motivation for Minhash/LSH

- **Suppose we need to find similar documents among $N = 1$ million documents**

- Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**

  - $N(N-1)/2 \approx$ **5*10^{11}** comparisons
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**

- For $N = \mathbf{10}$ million, it takes more than a year…



MINHASHING

Step 2: ***Minhashing:*** Convert **large variable length sets** to **short** <u>**fixed-length**</u> **signatures**, while <u>**preserving similarity**</u>

From Sets to Boolean Matrices

- **Rows** = elements (shingles)

Note: Transposed Document Matrix

- **Columns** = sets (documents)
  - 1 in row **e** and column *s* if and only if **e** is a valid shingle of document represented by s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value *1*)
  - **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

Outline: Finding Similar Columns

- **So far:**
  - A documents → a set of shingles
  - Represent a set as a boolean vector in a matrix

- **Next goal: Find similar columns while computing small signatures**
  - **Similarity of columns == similarity of signatures**

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

Outline: Finding Similar Columns

- **Next Goal: Find similar columns based on small signatures**

- **Naïve approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **Essential:** Similarities of signatures and columns are related
  - **3) Optional:** Check that columns with similar signatures are really similar

Outline: Finding Similar Columns

- **Next Goal: Find similar columns based on small signatures**

- **Naïve approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **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)

Hashing Columns (Signatures) : LSH principle

- **Key idea:** "hash" each column **C** to a small ***signature*** ***h(C)***, such that:
  - (1) ***h(C)*** is small enough that the signature fits in RAM
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ 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. $h(C_1) \neq h(C_2)$

- Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!

Min-Hashing

- **Goal: Find a hash function $h(\cdot)$ 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 the Jaccard similarity:** It is called **Min-Hashing**

Min-Hashing

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

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

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

Min-Hashing

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

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

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

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Min-Hashing Example

2nd element of the permutation is the first to map to a 1

Permutation π

2
3
7
6
1
5
4

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

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Min-Hashing Example

2nd element of the permutation is the first to map to a 1

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

4th element of the permutation is the first to map to a 1

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Min-Hashing Example

Note: Another (equivalent) way is to store row indexes or raw shingles (e.g. mouse, lion):

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

2nd element of the permutation is the first to map to a 1

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

4th element of the permutation is the first to map to a 1

Min-Hash Signatures

- **Pick K=100 random permutations of 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

    $$sig(C)[i] = \min(\pi_i(C))$$

- **Note:** The sketch (signature) of document C is small **~100 bytes!**

- **We achieved our goal! We "compressed" long bit vectors into short signatures**

Key Fact

For two sets A, B, and a min-hash function $mh_i()$:

$$Pr[mh_i(A) = mh_i(B)] = Sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Unbiased estimator for Sim using *K* hashes (notation policy – this is a different K from size of shingle)

$$\boxed{\hat{Sim}(A, B) = \frac{1}{k} \sum_{i=1:k} I[mh_i(A) = mh_i(B)]}$$

Key Fact

For two sets A, B, and a min-hash function $mh_i()$:

$$Pr[mh_i(A) = mh_i(B)] = Sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Unbiased estimator for Sim using *K* hashes (notation policy – this is a different K from size of shingle)

$$\boxed{\hat{Sim}(A, B) = \frac{1}{k} \sum_{i=1:k} I[mh_i(A) = mh_i(B)]}$$

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

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	?	?	?	?

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

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

The Min-Hash Property

- **Choose a random permutation π**
- **<u>Claim:</u>** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**

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

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

The Min-Hash Property

- **Choose a random permutation π**
- **<u>Claim:</u>** **$\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- **Why?**
  - Given a set X, the probability that any one element is the min-hash under π is $1/|X|$ ← (0)
    - It is equally likely that any $y \in X$ is mapped to the ***min*** element
  - Given a set X, the probability that one of any **k** elements is the min-hash under π is $\mathbf{k}/|X|$ ← (1)

The Min-Hash Property

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Given a set X, the probability that any one element is the min-hash under π is $1/|X|$ ← (0)
    - It is equally likely that any $y \in X$ is mapped to the ***min*** element
  - Given a set X, the probability that one of any **k** elements is the min-hash under π is **k**$/|X|$ ← (1)
  - For $C_1 \cup C_2$, the probability that any element is the min-hash under π is $1/|C_1 \cup C_2|$ (from 0) ← (2)
  - For any C_1 and C_2, the probability of choosing the same min-hash under π is $|C_1 \cap C_2|/|C_1 \cup C_2|$ ← from (1) and (2)

Similarity for 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

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

Min-Hash Signatures

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

$$sig(C)[i] = \min(\pi_i(C))$$

Min-Hash Signatures

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

$$sig(C)[i] = \min(\pi_i(C))$$

- **Note:** The sketch (signature) of document C is small **~100 bytes!**

- **We achieved our goal! We "compressed" long bit vectors into short signatures**

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Implementation Trick

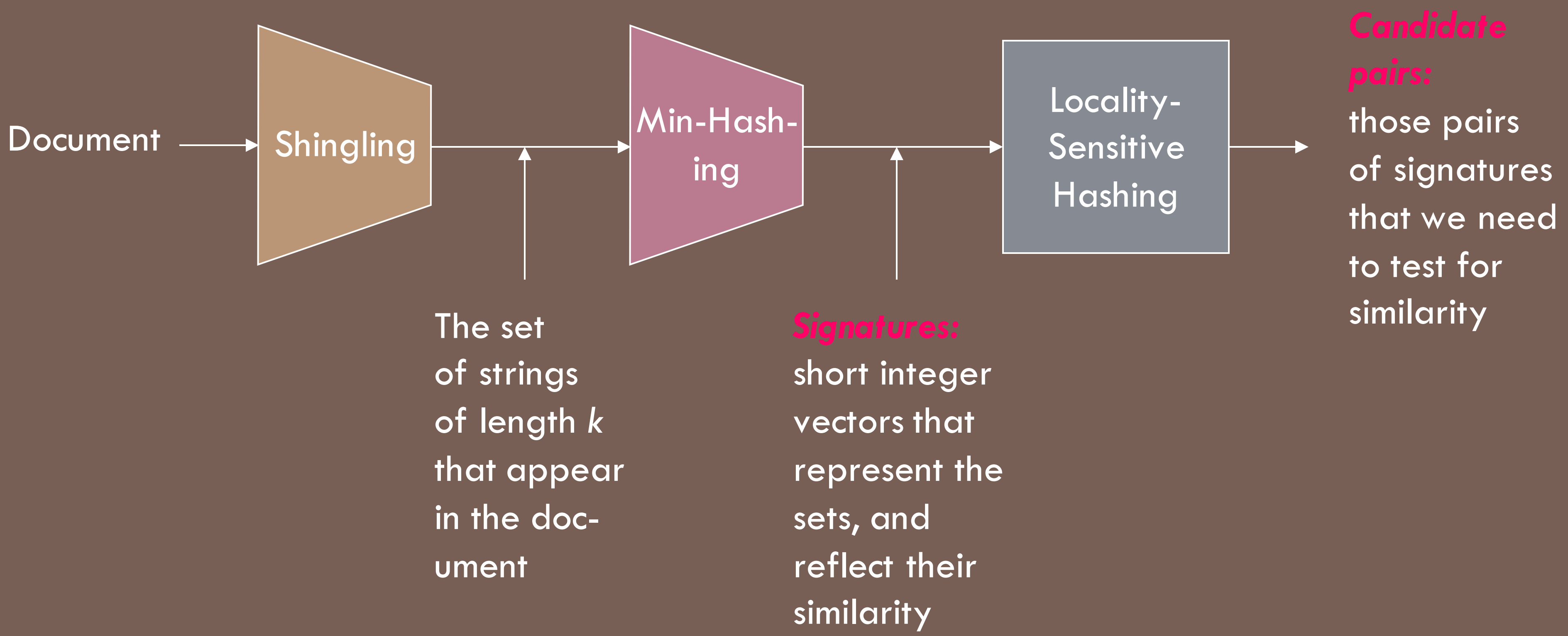
- **Permuting rows even once is prohibitive**
- **Row hashing!**
  - Pick **K = 100** hash functions k_i
  - Ordering under k_i gives a random row permutation!
- **One-pass implementation**
  - For each column **C** and hash-func. k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - **Scan rows looking for 1s**
    - Suppose row J has 1 in column ***C***
    - Then for each k_i :
      - If $k_i(J) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(J)$

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)

Page 13: http://infolab.stanford.edu/~ullman/mmds/ch3.pdf

Summary: Two Key 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 $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - We used hashing to get around generating random permutations



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**

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

Candidates from Min-Hash

- **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:
  $\boldsymbol{M(i, x) = M(i, y)}$ for at least frac. ***s*** values of ***I***
  - We expect documents ***x*** and ***y*** to have the same (Jaccard) similarity as their signatures

LSH for Min-Hash

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

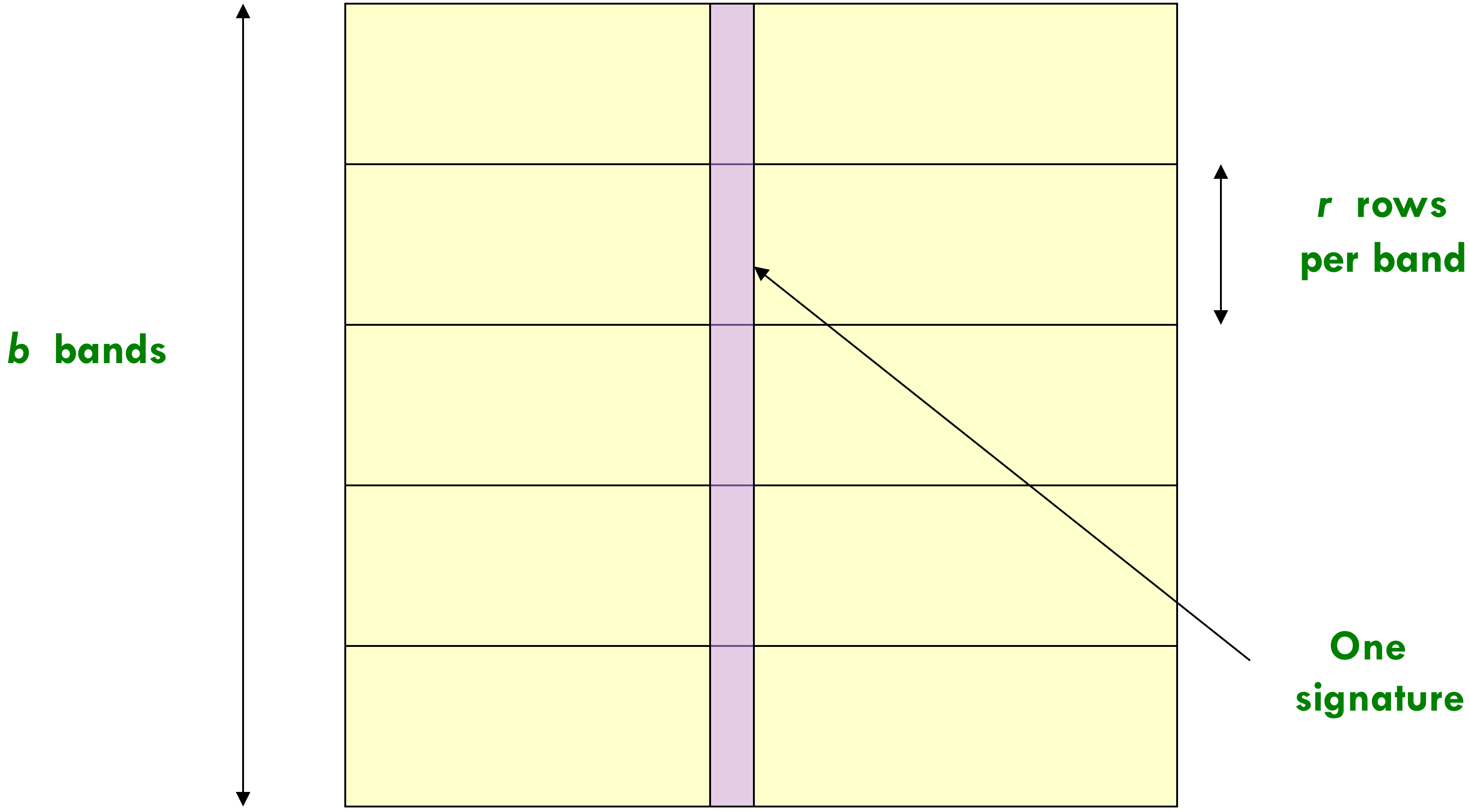
- **Big idea: Hash columns of signature matrix *M* several times**

- 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**

Partition *M* into *b* Bands

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



Signature matrix *M*

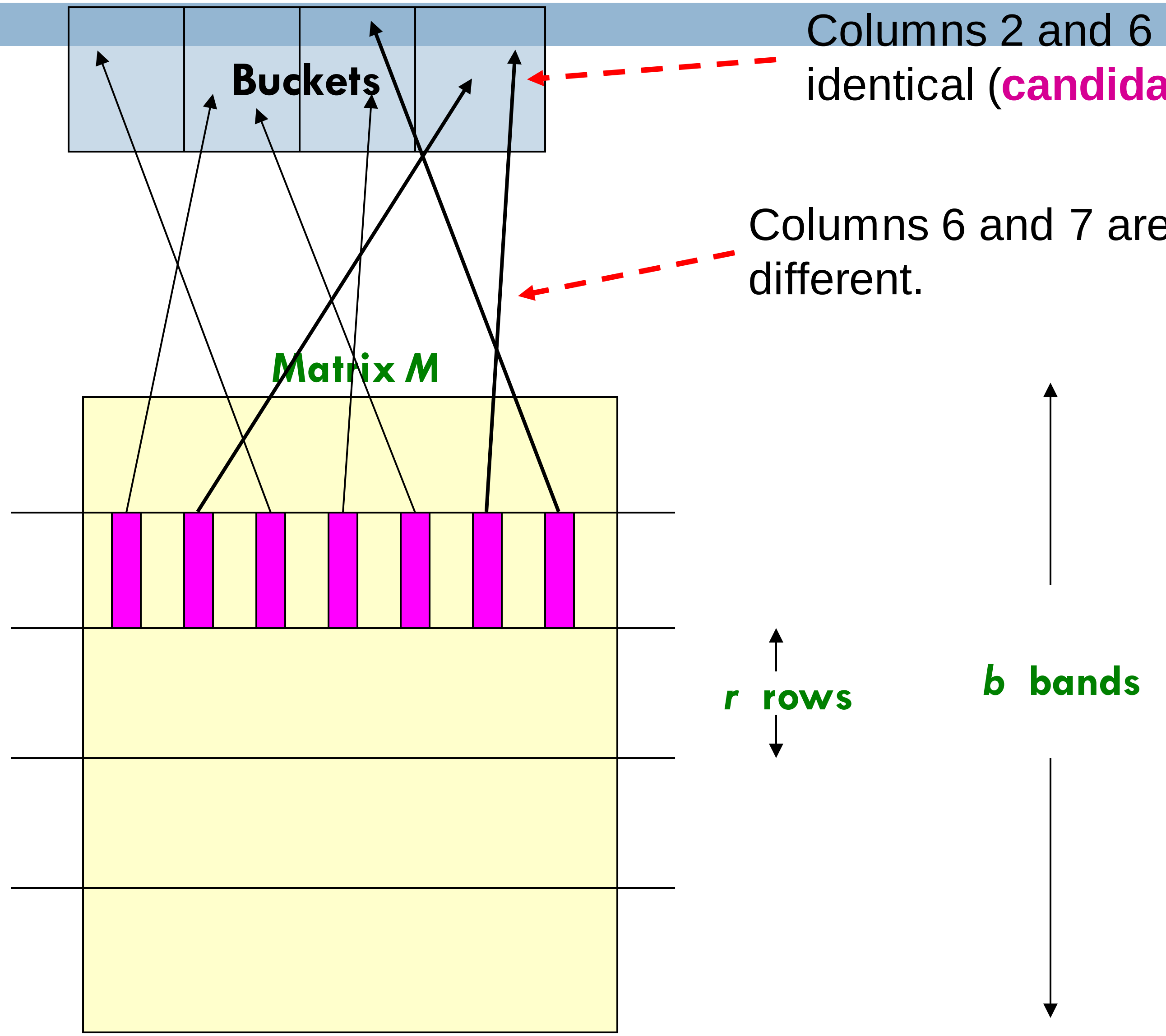
Partition M into Bands

- 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

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

Hashing Bands



Columns 2 and 6 are probably identical (**candidate pair**)

Columns 6 and 7 are surely different.

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 of Bands

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

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

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:** sim(C_1, C_2) = 0.8
  - Since 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)

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:** sim(C_1, C_2) = 0.8
  - Since 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 any 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 $\mathbf{b}=20$, $\mathbf{r}=5$
- **Assume:** $\text{sim}(C_1, C_2) = 0.3$
  - Since $\text{sim}(C_1, C_2) < \mathbf{s}$ we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)

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:** $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < \mathbf{s}$ 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 30% 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

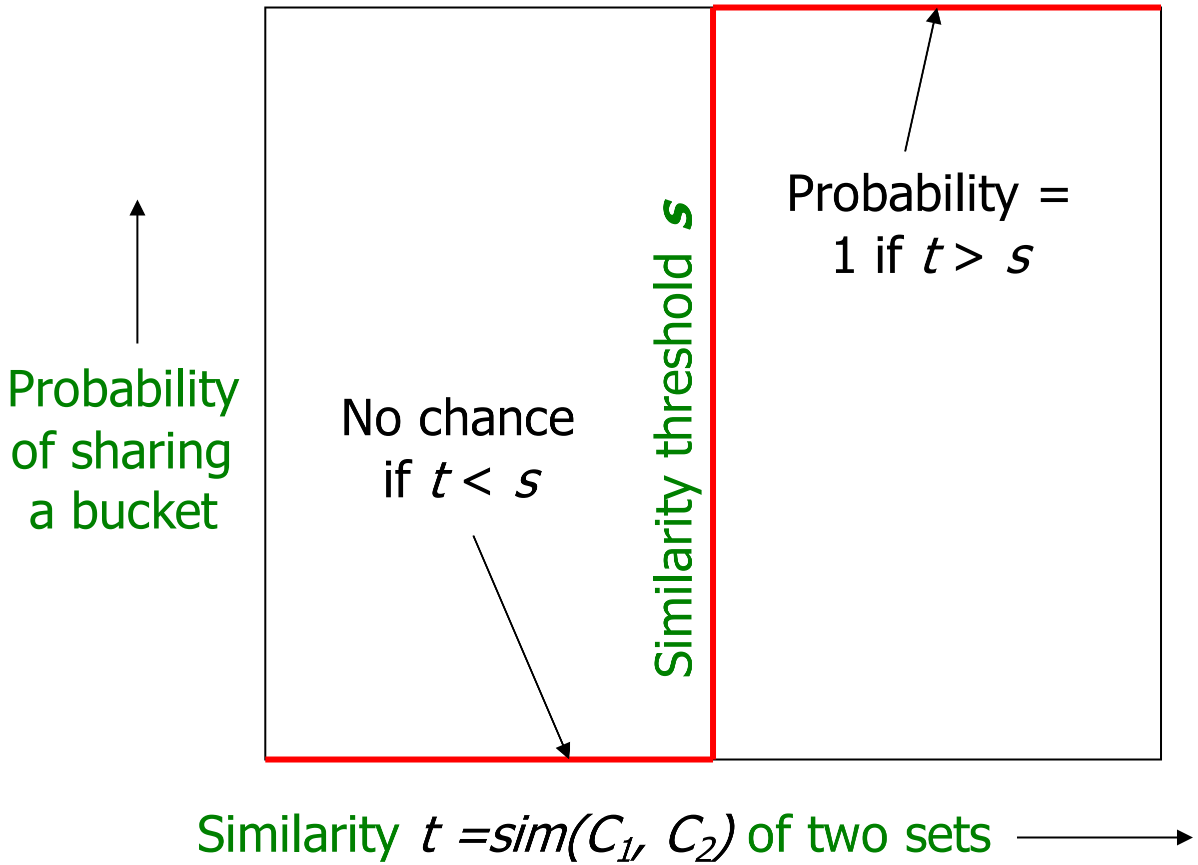
2	1	4	1
1	2	1	2
2	1	2	1

- **Pick:**
  - The number of Min-Hashes (rows of ***M***)
  - The number of bands ***b***, and
  - The number of rows ***r*** per band

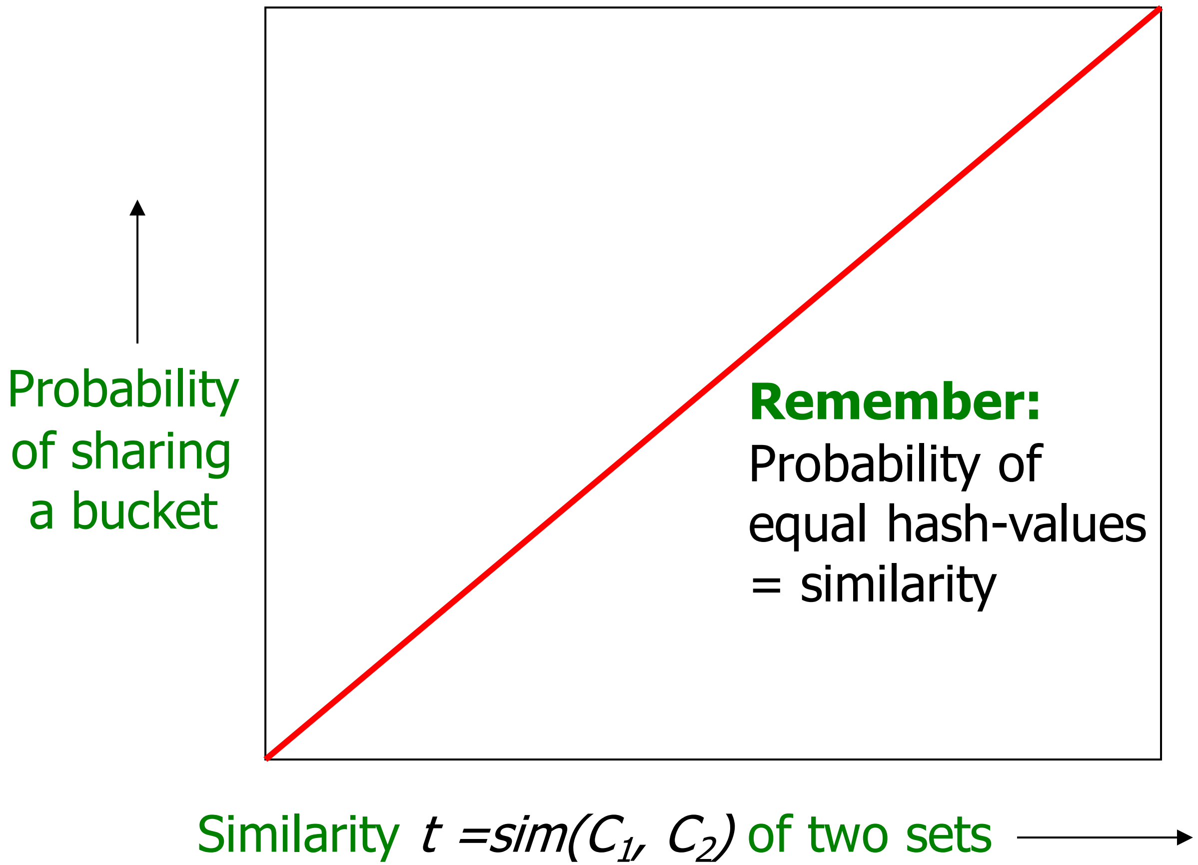
  to balance false positives/negatives

- **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



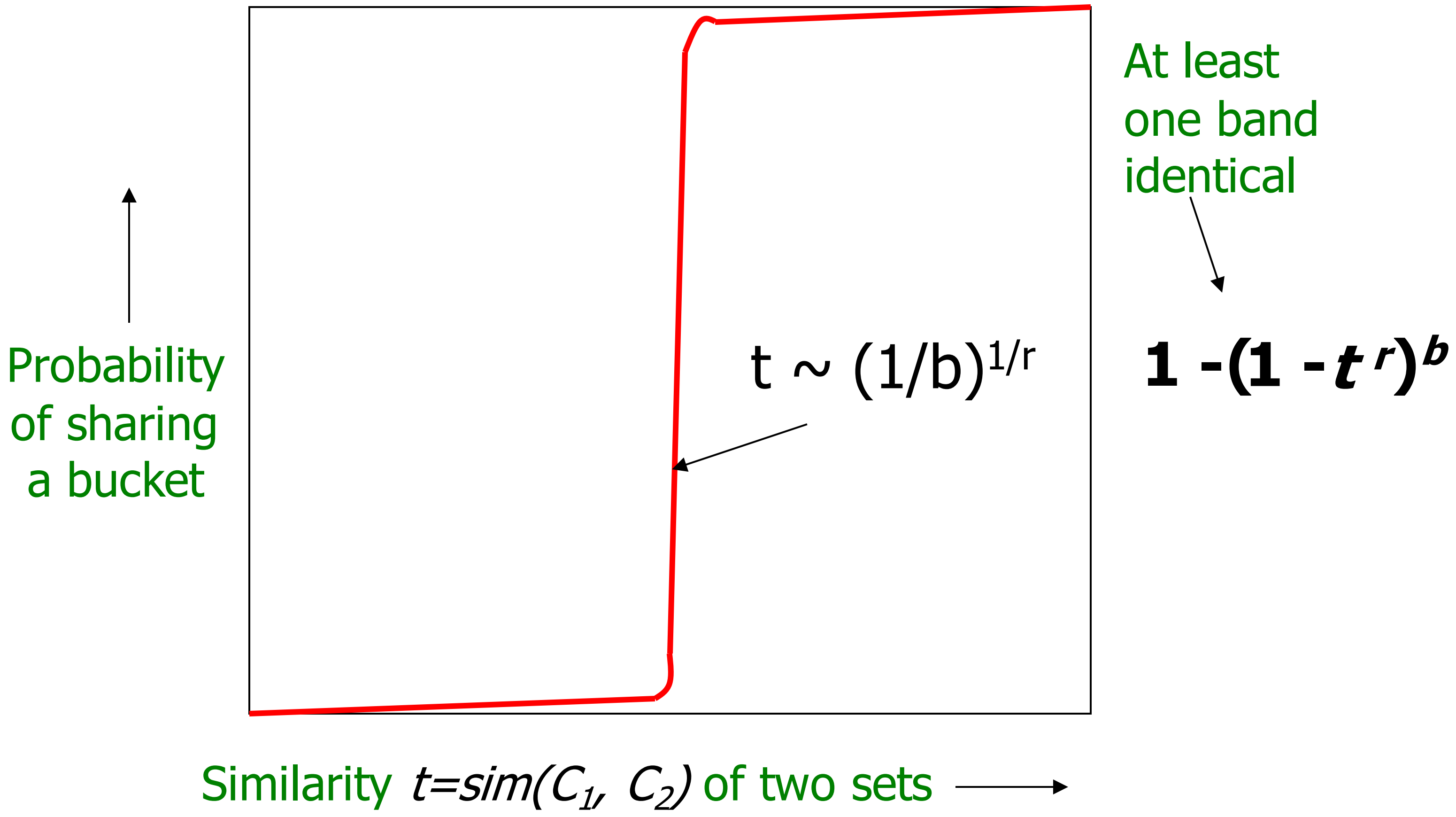
b bands, *r* rows/band

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

What b Bands of r Rows Gives You



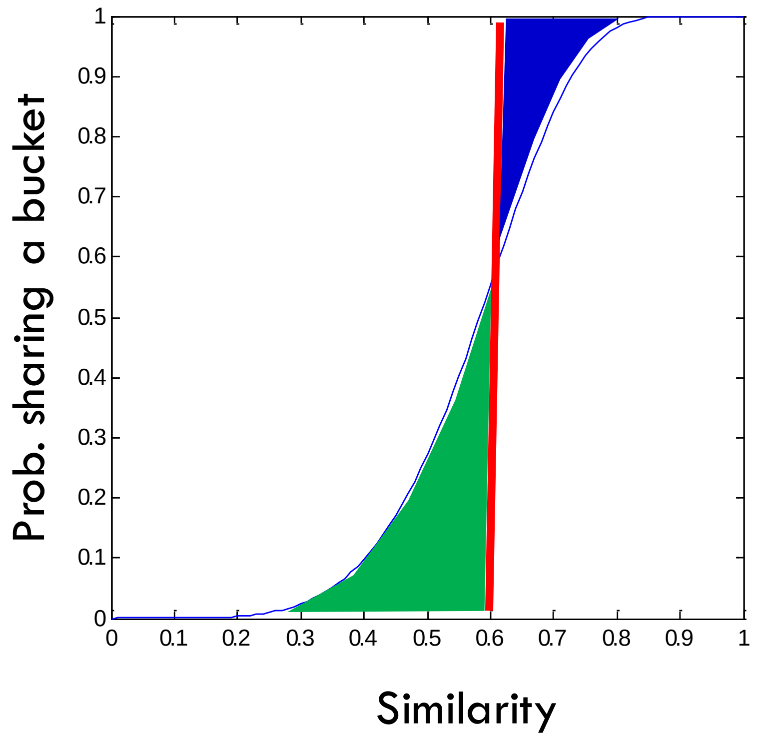
Example: $b = 20; r = 5$

- **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

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

Backup slides