Similarity Search 1

Presentation partially based on slides from:

http://infolab.stanford.edu/~ullman/mining/2009/index.html

Anand Rajaraman & Jeff Ullman

Agenda

- Applications
- Shingling
- Jaccard Similarity
- **Minhashing**
- **Locality-Sensitive Hashing**

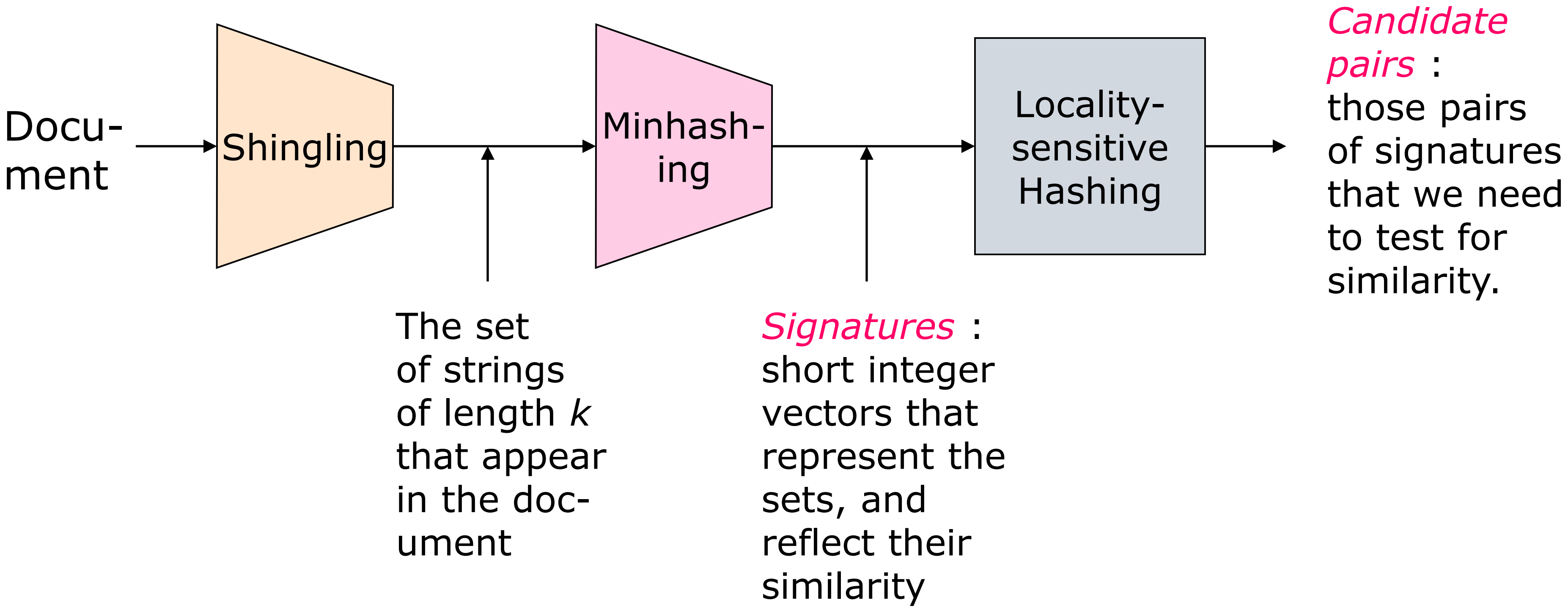
Finding Similar Documents

- Given a huge number (millions) of documents, e.g., the Web, find **QUICKLY** pairs of docs that have a lot of text in common, e.g.:
  - Mirror sites, or approximate mirrors.
    - Don’t want to show both in a search.
  - Plagiarism, including large quotations.
  - Similar news articles at many news sites.
    - Reflects importance of the news item.

Three steps for Similarity Testing

1. *Shingling* : convert documents, emails, etc., 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 similar.

The Big Picture



Comparing Documents

- What makes documents “similar”?
- Special cases are easy, e.g., identical documents, or one document contained character-by-character in another.
- General case, where many small pieces of one doc appear out of order in another, is very hard.

Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k consecutive characters* that appear in the document.

- Example: k=2; doc = abcab.
  Set of 2-shingles = {ab, bc, ca}.
  - Option: regard shingles as a bag, and count ab twice.

- Represent a doc by its set of *k*-shingles.

- Another Option: *k*-shingle for a document is a sequence of *k* consecutive words that occur in the document

Working Assumption

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

- Careful: you must pick k large enough, or most documents will have most shingles.
  - k = 5 is OK for short documents (emails);
  - k = 10 is better for long documents;
  - k = 3 when working with word shingles;
  - For different languages different values

Shingles: Compression Option

- To compress long shingles, we can hash them to (say) 4 bytes => uint32, easy to compare and store [2^32 = 4.294.967.296 = 4 billion values]

- Represent a doc by the set of hash values of its *k*-shingles.

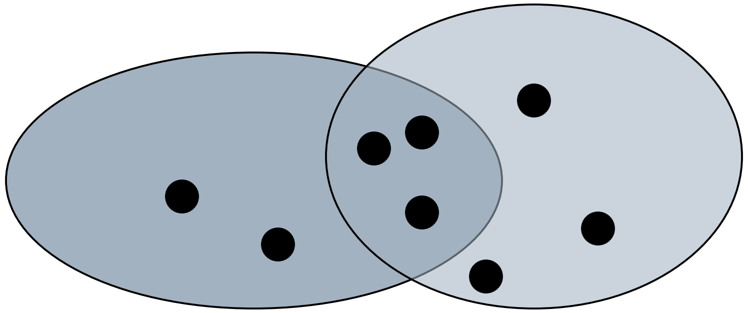
- Probability of “clashes” (two shingles mapped into the same integer) is very small, so we may assume that two documents have *k* shingles in common, when their hashed representations share abut *k* hash-values.

Jaccard Similarity

- Given two sets C_1 and C_2 we define the *Jaccard similarity* of C_1 and C_2 , *Sim* (C_1,C_2), as the ratio of the sizes of the intersection and union of C_1 and C_2:

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

e.g.: Sim (A,B)=3/8



Documents in Matrix Form

- Rows = shingles (or hashes of shingles)
- Columns = documents
- 1 in row *r*, column *c* iff document *c* has shingle *r*
- Expect the matrix to be sparse

- WE USE THE MATRIX FORM ONLY FOR CONCEPTUALIZATION; THE IMPLEMENTATION WILL BE VERY DIFFERENT!!!

Example

	S	T	U	V	W
a	1	1	0	1	0
b	1	0	1	1	0
c	1	0	0	1	0
d	0	1	0	0	1
e	1	0	1	0	1
f	1	1	0	1	1
g	0	1	0	1	1
h	0	1	0	1	0

S = {a,b,c,e,f} T = {a,d,f,g,h} U = {b,e}
V = {a,b,c,f,g,h} W = {d,e,f,g}

Jaccard Similarity

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

$$Sim\ (C_1, C_2) = 2/5 = 0.4$$

Signatures

- Key idea: "hash" each column *C* to a small *signature* *Sig* (C), such that:

1. *Sig* (C) is small enough that we can fit a signature in main memory for each column.
2. *Sim* (C_1, C_2) is "almost" the same as the "similarity" of *Sig* (C_1) and *Sig* (C_2).

An Idea That Doesn't Work

- Pick 100 rows at random, and let the signature of column *C* be the 100 bits of *C* in those rows.

- Because the matrix is sparse, many columns would have 00. . .0 as a signature, yet have Jaccard similarity > 0, because their 1's are "outside" the 100 selected rows.

Four Types of Rows

Given two columns C_1 and C_2, rows may be labeled as:
Both1: (1, 1); Left1: (1, 0); Right1: (0,1); **(0, 0) are irrelevant!**

	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

$$Sim\,(C_1, C_2) = B\,/(B + L + R\,)$$

B, L, R denote counts of rows of type *Both1, Left1, Right1*

Sim (C_1, C_2) is invariant with respect to permutations of rows!

Minhashing

- Imagine the rows permuted randomly
- Define "hash" function $h(C)$ = the position of the first (in the permuted order) row in which column C has 1.
- Use several (e.g., 100) independent hash functions to create a signature.

Minhashing Example

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

Input matrix

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

This is the most important slide!

Surprising Property

The probability (over all permutations of the rows) that h (C_1) = h (C_2) is the same as *Sim* (C_1, C_2).

- Prob($h(C_1) = h(C_2)$) and *Sim* (C_1, C_2) are equal to:

  $B/(B+L+R)$

- Why?
  - Take a random permutation of rows *h.*
  - Look down permuted columns C_1 and C_2 until you see a 1.
  - If it's a type-*B* row, then $h(C_1) = h(C_2)$. If a type-*L* or type-*R* row, then $h(C_1) \neq h(C_2)$. And Prob(type-B)=B/(B+L+R) !

Permutations are long, signatures are short!

Similarity of Signatures

- Thus, the test if h (C_1) = h (C_2) will return "yes" with probability *Sim* (C_1, C_2). Therefore, repeating the test 100 times with various permutations h will lead to a good estimate of *Sim* (C_1, C_2) !

- The *signature of length k for a set C is*

$$<h_1(C), h_2(C),\dots, h_k(C)>$$

  where h_1, h_2,…, h_k are some randomly chosen permutations

- The *similarity of signatures* is the fraction of the positions in which they agree.

Min Hashing – Example

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

Input matrix

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

Minhash signatures

- Pick (say) 100 random permutations of the rows.
- Think of *Sig* (C) as a column vector.
- Let *Sig* (C)[i] =

  the position of the first row that has a 1 in column *C*
  according to the *i* th permutation
- How to "remember" the 100 permutations of "a very long vector of possible positions"??? ***Hash functions?***
- h: a hash function; {w1, …, wk} -> min({h(w1), …, h(wk)})

Implementation – (1)

- Suppose 1 billion rows…
- Hard to pick a random permutation from 1…billion.
- Representing a random permutation requires 1 billion entries.
- Accessing rows in permuted order leads to thrashing (memory swapping).

- *Idea: Instead of a permutation use a hash function (applied to row numbers) and "implicitly" find the minimum element for each column.*

Simulating permutations column by column:

document [w1, …, wk] -> min({h(w1), …, h(wk)})

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

Input matrix

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

First hash function
applied to the first column:

Inf -> 3 ->(4?) ->**2** ->(5)?

Second hash function
applied to the first column:

Inf -> 4 ->**2**

Third hash function
applied to the first column:

Inf -> 1 ->**1**

Observations (example related):

- Hash functions can take bigger values than 1-7
- Potential clashes (h(a)==h(b)) are rare and not harmful
- Documents can be represented as *lists of items* (words? shingles? shingle id's?), so instead of scanning the whole column we can scan a list of items, avoiding "0's"!
- Unfortunately, scanning the table column by column will require multiple applications of the same hash function to different columns (e.g. the first hash function will be applied to the first row of the first and the third column).
- *Therefore it's cheaper to scan the table "row by row"!*
- *But that requires a change of representation of documents:* *item -> list of documents that contain the item* [that's easy to find!]

Implementation – (2)

- Consider a family of, say, 100 hash functions h_i
- For each column c and each hash function h_i, keep a "slot" $M(i, c)$ for that minhash value
  (M is of size 100*#documents)
- Initialize each slot to Infinity
- **Scan your "big document table", row by row, updating the matrix M…**

Implementation – (3)

```
for each row r
    for each column c
        if c has 1 in row r
            for each hash function h_i do
                if h_i(r) is a smaller value than M(i, c) then
                    M(i, c) := h_i(r);
```

Implementation – (4)

- If data is stored row-by-row, then only one pass is needed
- If data is stored column-by-column
  - E.g., data is a sequence of documents

  represent it by (row-column) pairs and sort once by row.
  - *Saves cost of computing $h_i(r)$ many times.*
- O(N) time complexity (N = total size of docs (?))

Checking All Pairs is Hard

- While the signatures of all columns may fit in main memory, comparing the signatures of all pairs of columns is quadratic in the number of columns.
- Example: 10^6 columns implies $5*10^{11}$ comparisons.
- At 1 microsecond/comparison: 6 days.
- **Solution: Locality Sensitive Hashing!**

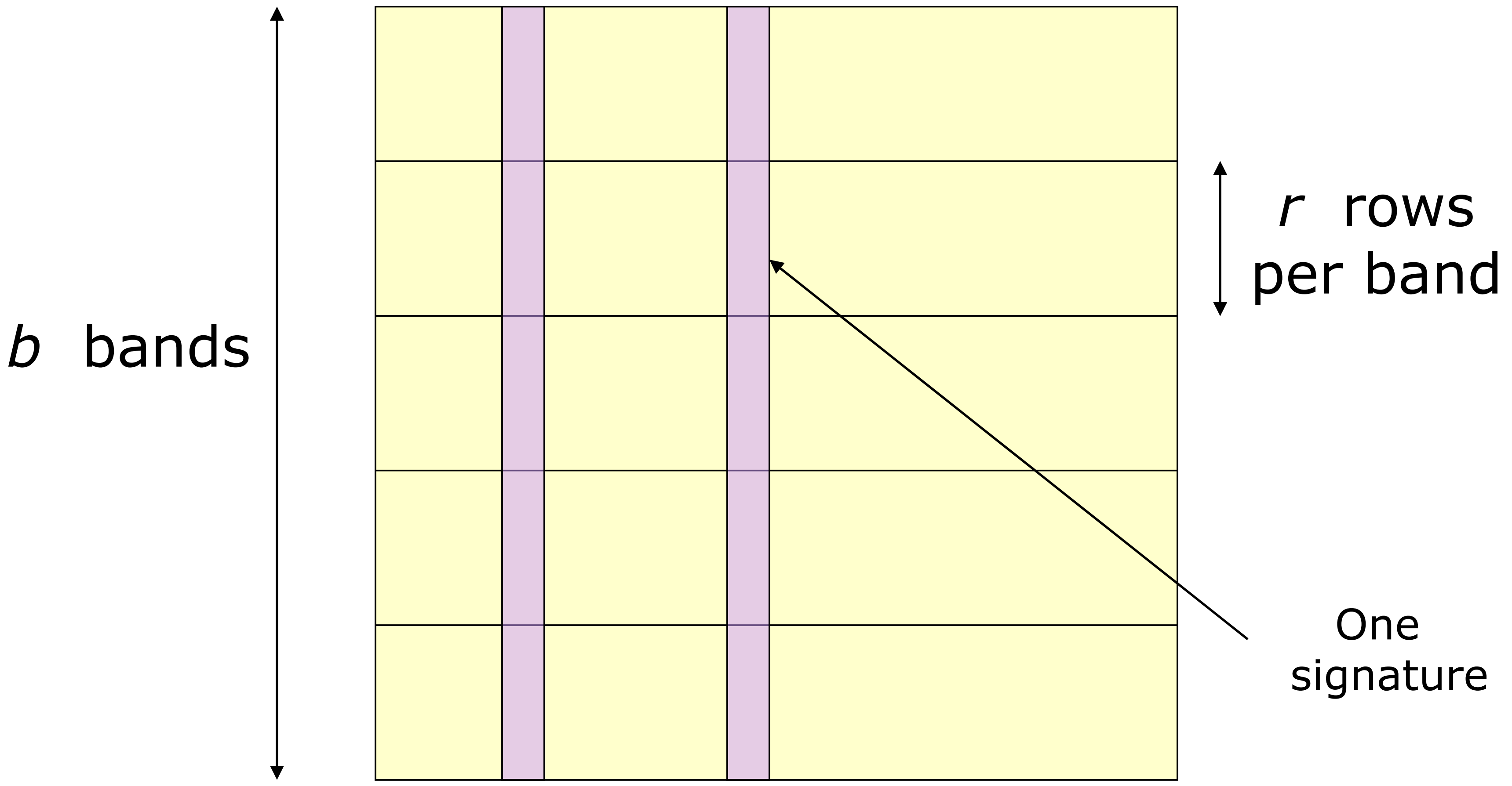
LSH: Key Ideas

- Split columns of signature matrix *M* into **several blocks** of the same size.

- If **two columns are very similar** then it is very likely that at least **at one block they will be identical**. Instead of comparing blocks, send them to buckets with help of hash functions!

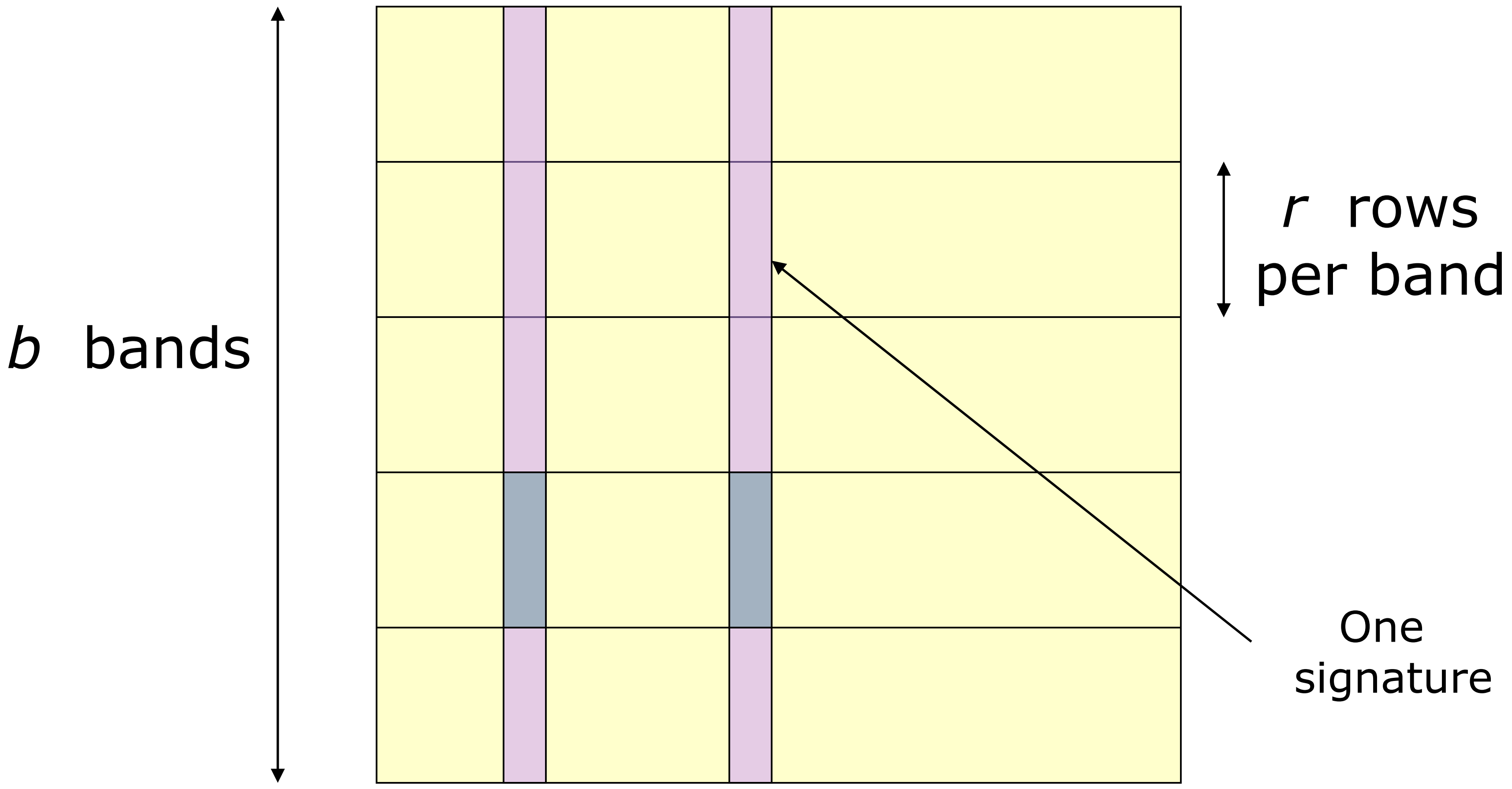
- **Candidate pairs** are those that hash **at least once** to the same bucket. Such pairs are often similar to each other, but rarely they might be not similar (“false positives”). Therefore, check if candidate pairs are really similar!

Partition Into Bands (1)

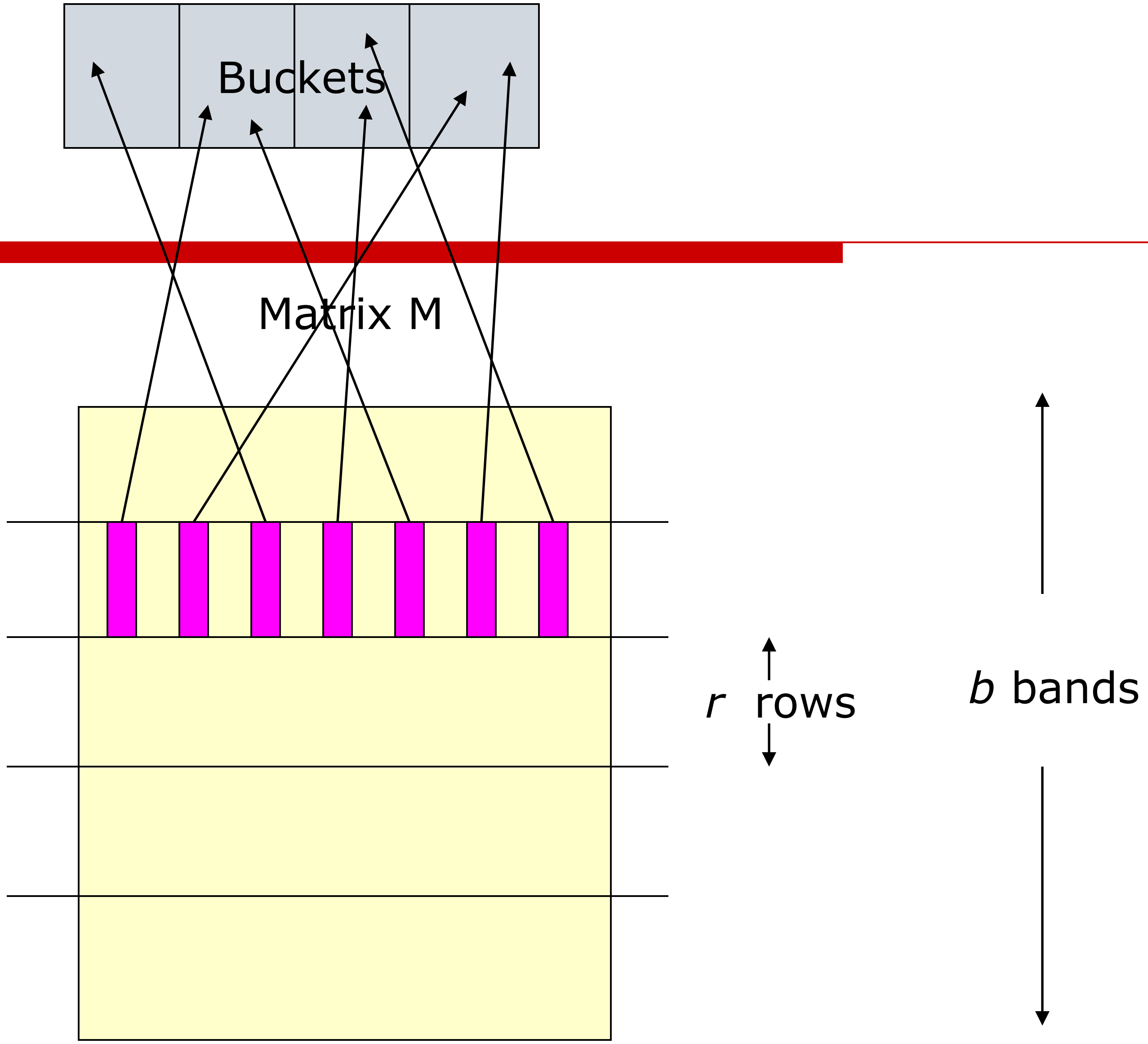


Matrix M

Partition Into Bands (1)



Matrix M



Buckets
Matrix M
r rows
b bands

Partition into Bands – (2)

- 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 (k big!).
- *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.

Example: Effect of Bands

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

Suppose C_1, C_2 are 80% Similar

- Probability C_1, C_2 identical in one particular band:
  $(0.8)^5 = 0.328$.

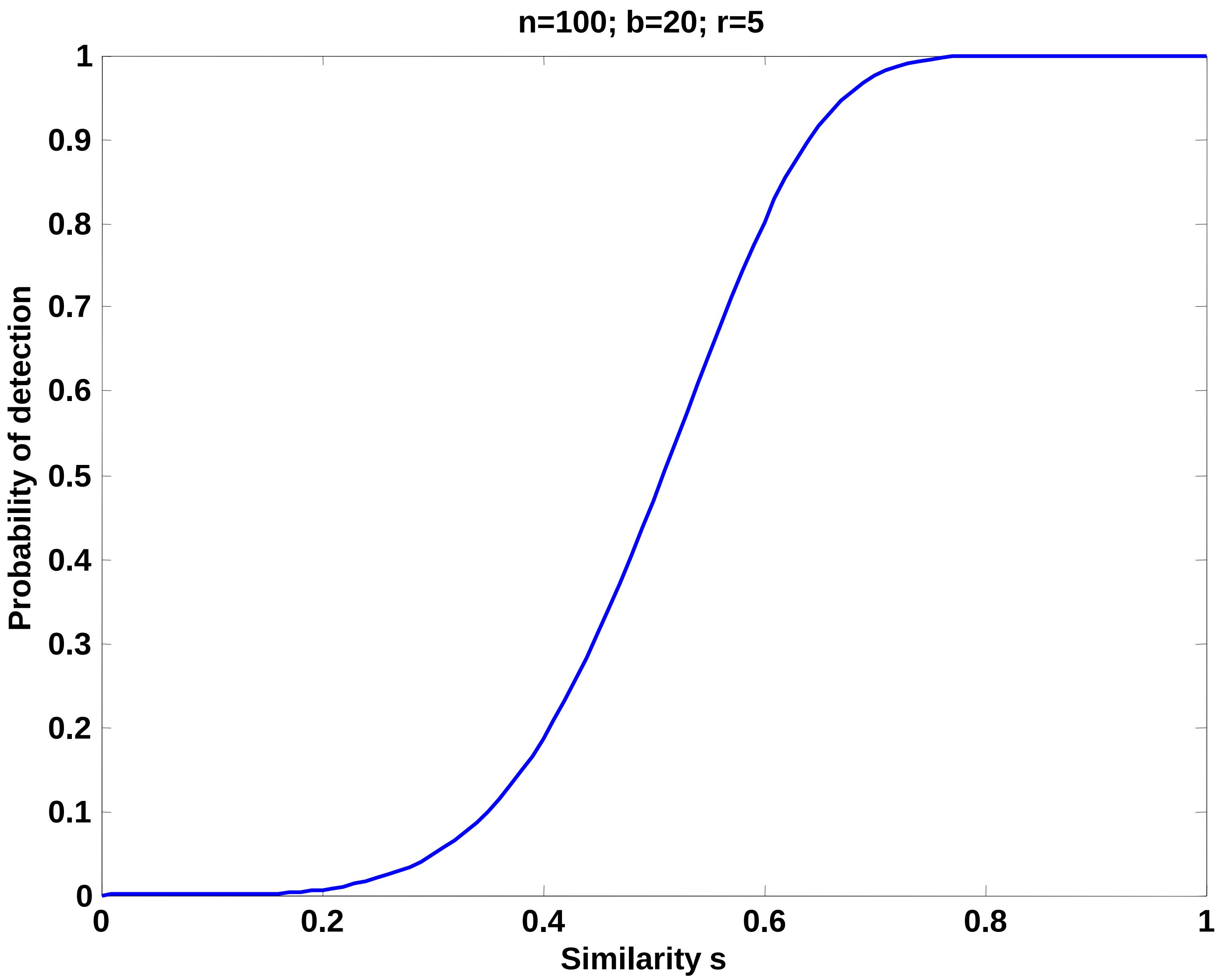
- Probability C_1, C_2 are *not identical* in any of the 20 bands:
  $(1\text{-}0.328)^{20} = .00035$.
  - i.e., we miss about 1/3000th of the 80%-similar column pairs.

- Probability C_1, C_2 identical in at least one band:
  $1 - (1\text{-}0.328)^{20} = 0.99965$.

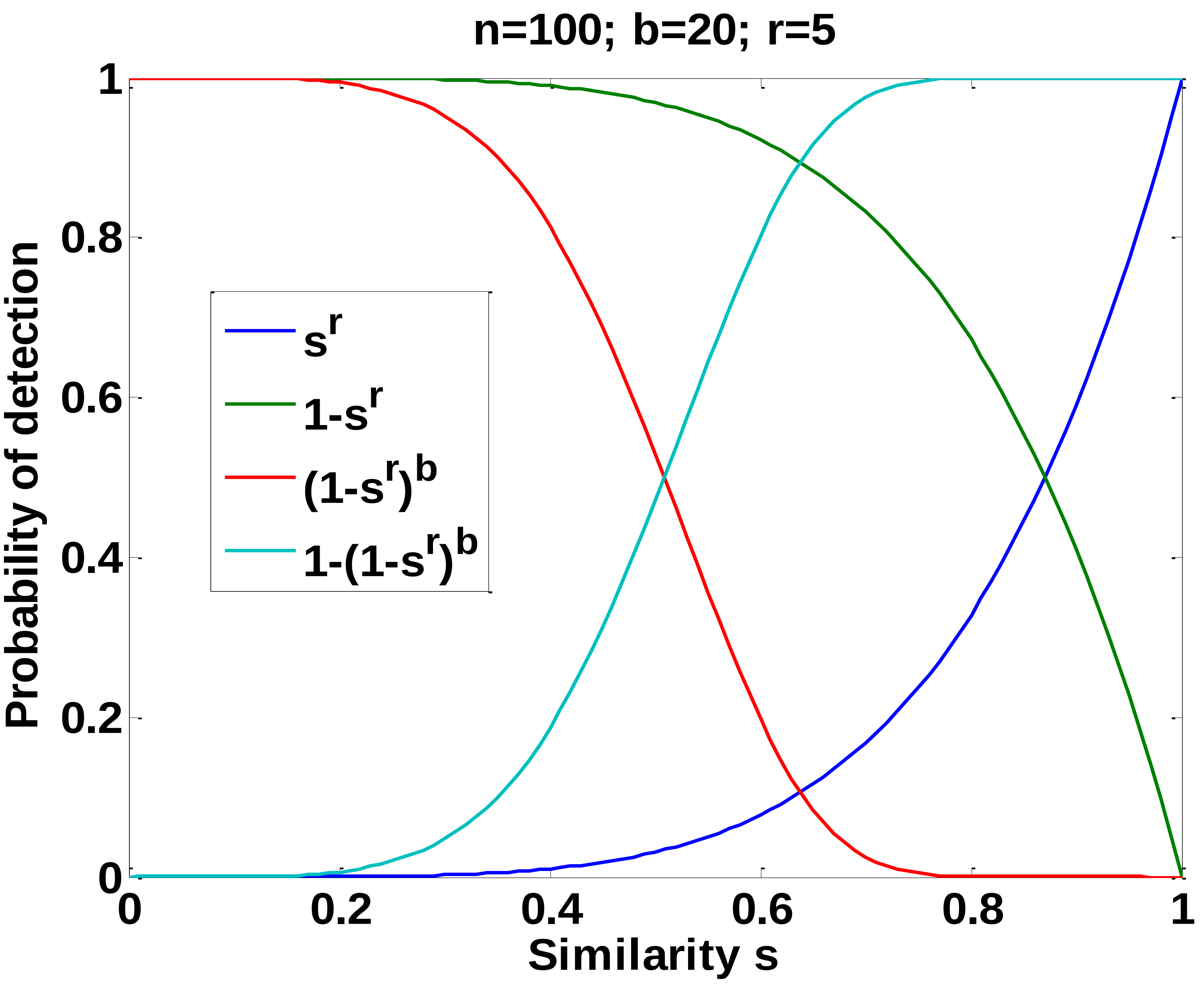
- *Probability of "being detected" for "less similar" pairs?*

General Case

- Probability that the signatures agree on one row is:
  s (Jaccard similarity)
- Probability that they agree on all r rows of a given band is:
  s^r
- Probability that they do not agree on any of the rows of a band is:
  $1 - s^r$
- Probability that for none of the b bands they agree in all rows of that band is:
  $(1 - s^r)^b$ *[false reject!]*
- Probability that the signatures will agree in all rows of at least one band is:
  $1 - (1 - s^r)^b$ *[probability of acceptance as a function of s]*
- This function is the probability that the signatures will be compared for similarity.



n=100; b=20; r=5
Probability of detection
Similarity s
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1



n=100; b=20; r=5
Probability of detection
Similarity s
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
s^r
$1-s^r$
$(1-s^r)^b$
$1-(1-s^r)^b$

Complexity

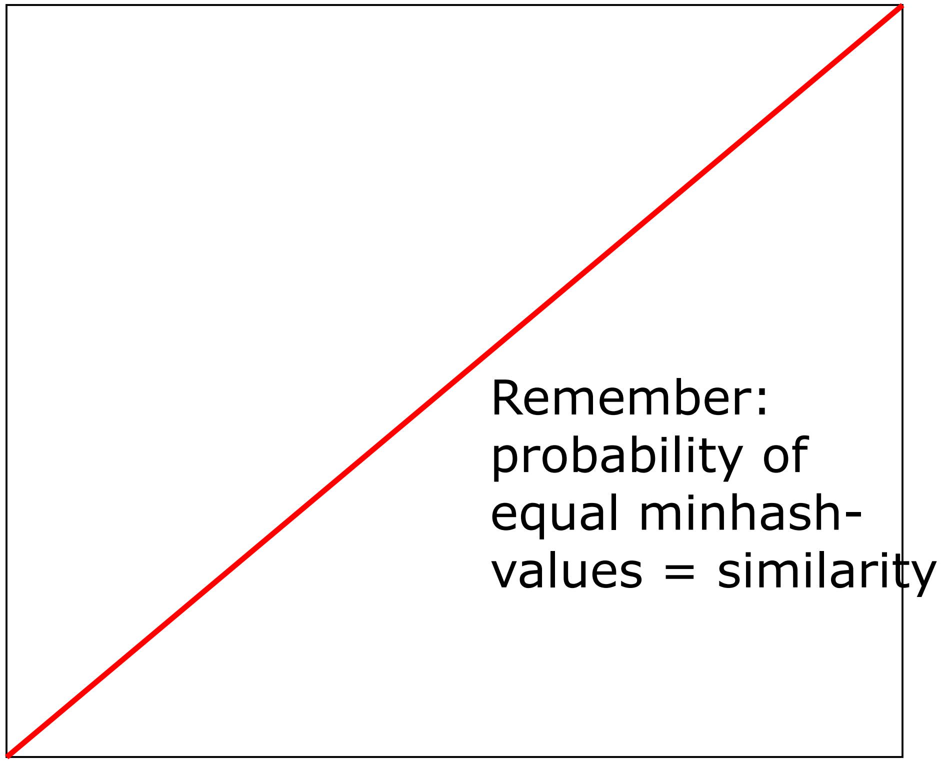
- Time needed to create bands and buckets is linear in the size of M (*O(#documents)*)

- Additionally, we have to *check if "candidate pairs" are really similar*. Two options:
  - compare signatures
  - compare original documents

- The number of detected pairs depends on:
  - the shape of the "detection curve" (the steeper the better)
  - the "true distribution" of similar pairs (do we know it?)

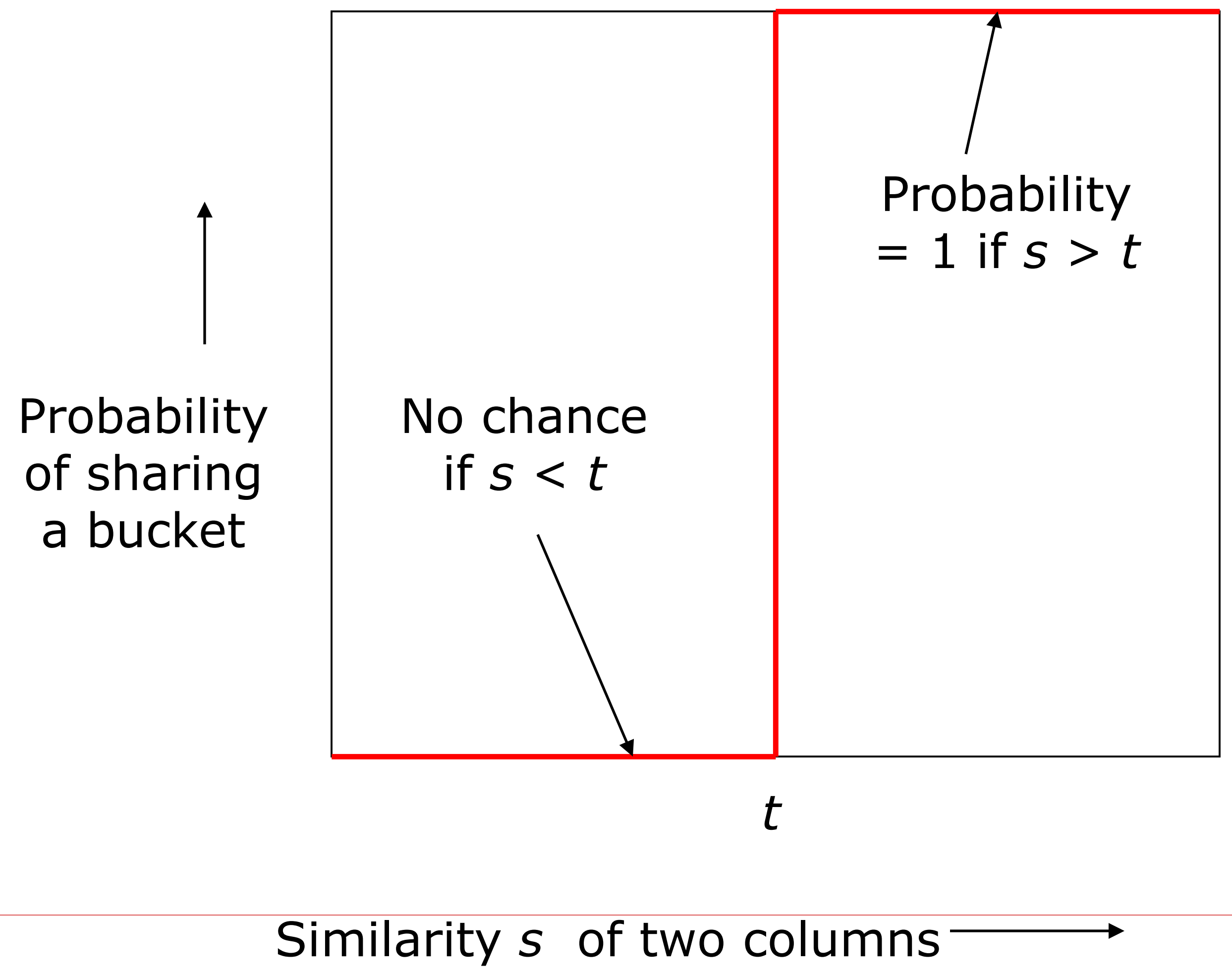
What One Row Gives You

Probability of sharing a bucket

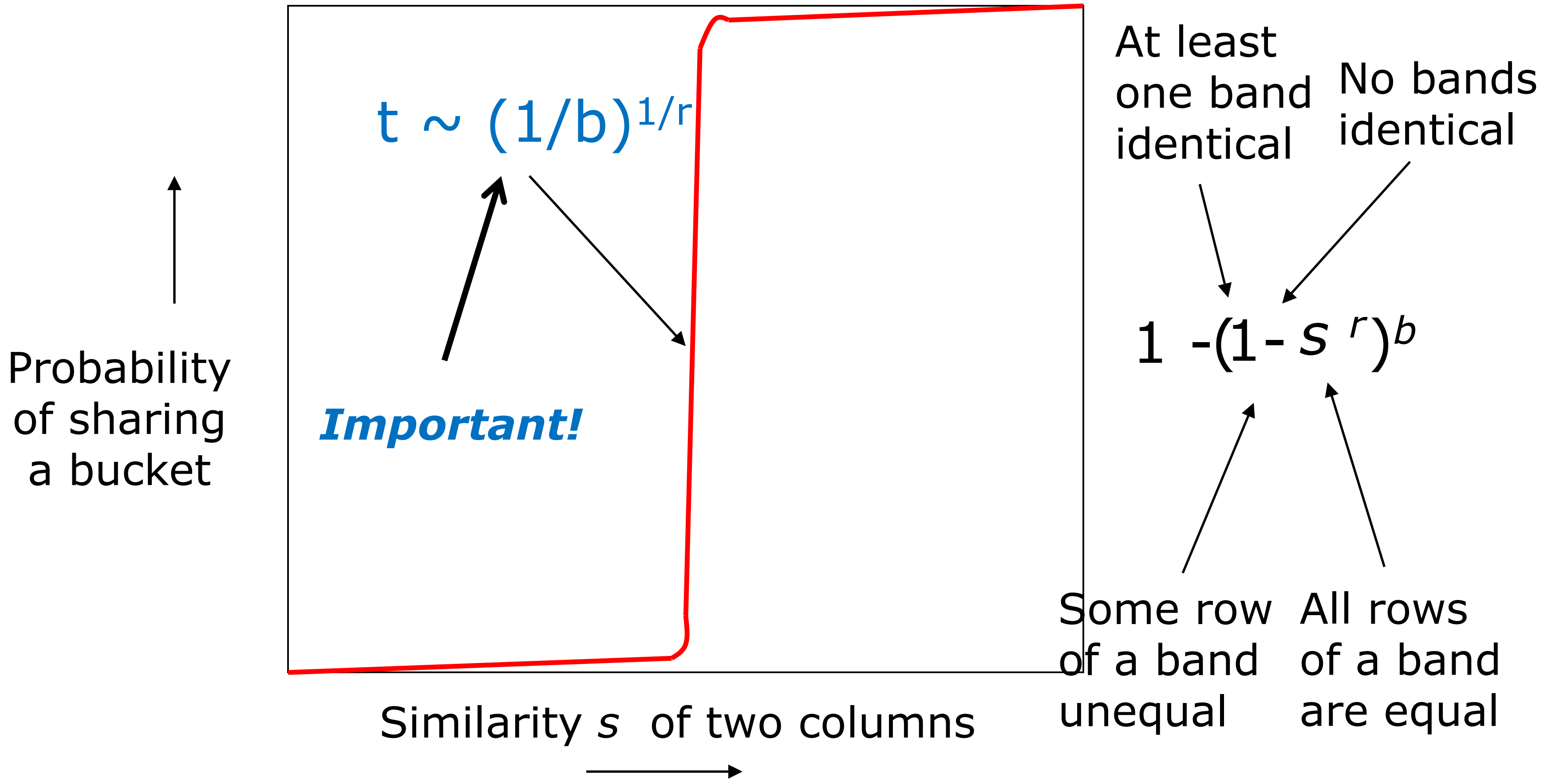


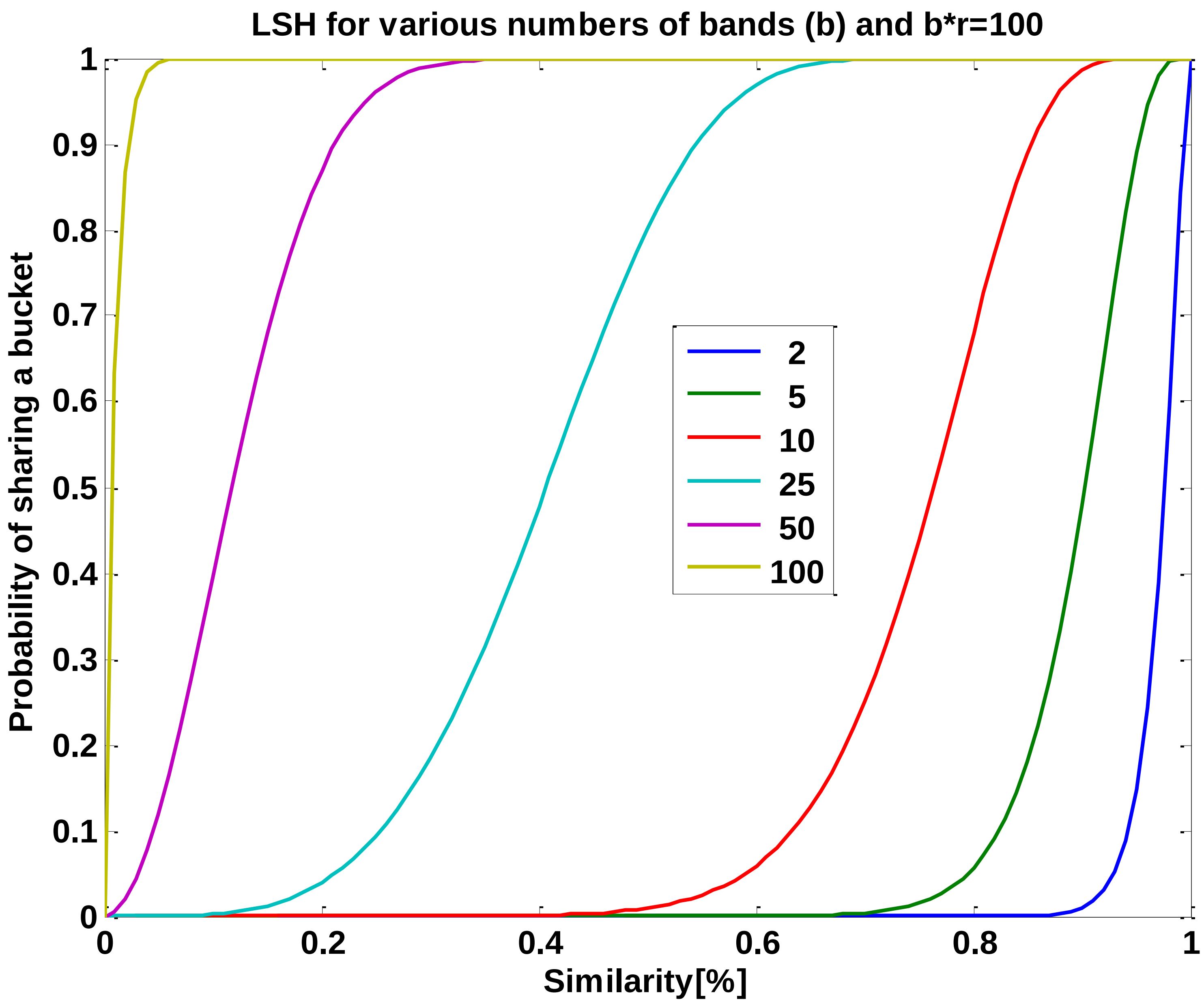
Similarity *s* of two columns

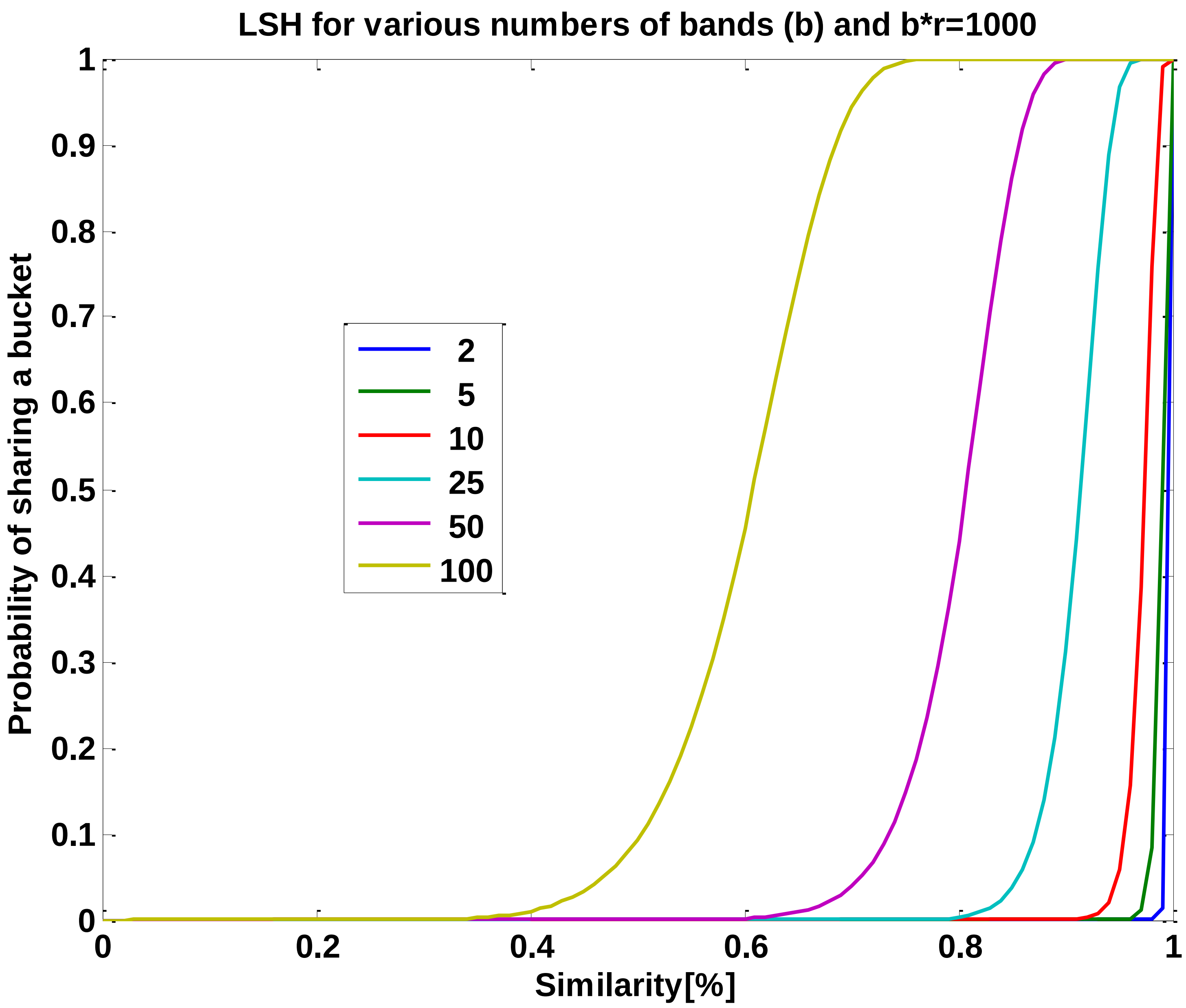
Analysis of LSH – What We Want



What b Bands of r Rows Gives You




LSH for various numbers of bands (b) and b*r=100
Probability of sharing a bucket
Similarity[%]
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1
2
5
10
25
50
100



LSH for various numbers of bands (b) and b*r=1000
Probability of sharing a bucket
Similarity[%]
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1
2
5
10
25
50
100

LSH in practice

- Get an idea of the “true” distribution of similarities of your collection of documents.
- Tune the n, r, b parameters 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.[why?]
- Optional: In another pass through data, check that the remaining candidate pairs really represent similar *sets*. [why?]