Finding Similar Sets

Applications

Shingling

Minhashing

Locality-Sensitive Hashing

Goals

- Many Web-mining problems can be expressed as finding "similar" sets:
  1. Pages with similar words, e.g., for classification by topic.
  2. NetFlix users with similar tastes in movies, for recommendation systems.
  3. Dual: movies with similar sets of fans.
  4. Images of related things.

Similarity Algorithms

- The best techniques depend on whether you are looking for items that are very similar or only somewhat similar.
- We'll cover the "somewhat" case first, then talk about "very."

Example Problem: Comparing Documents

- Goal: common text, not common topic.
- 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.

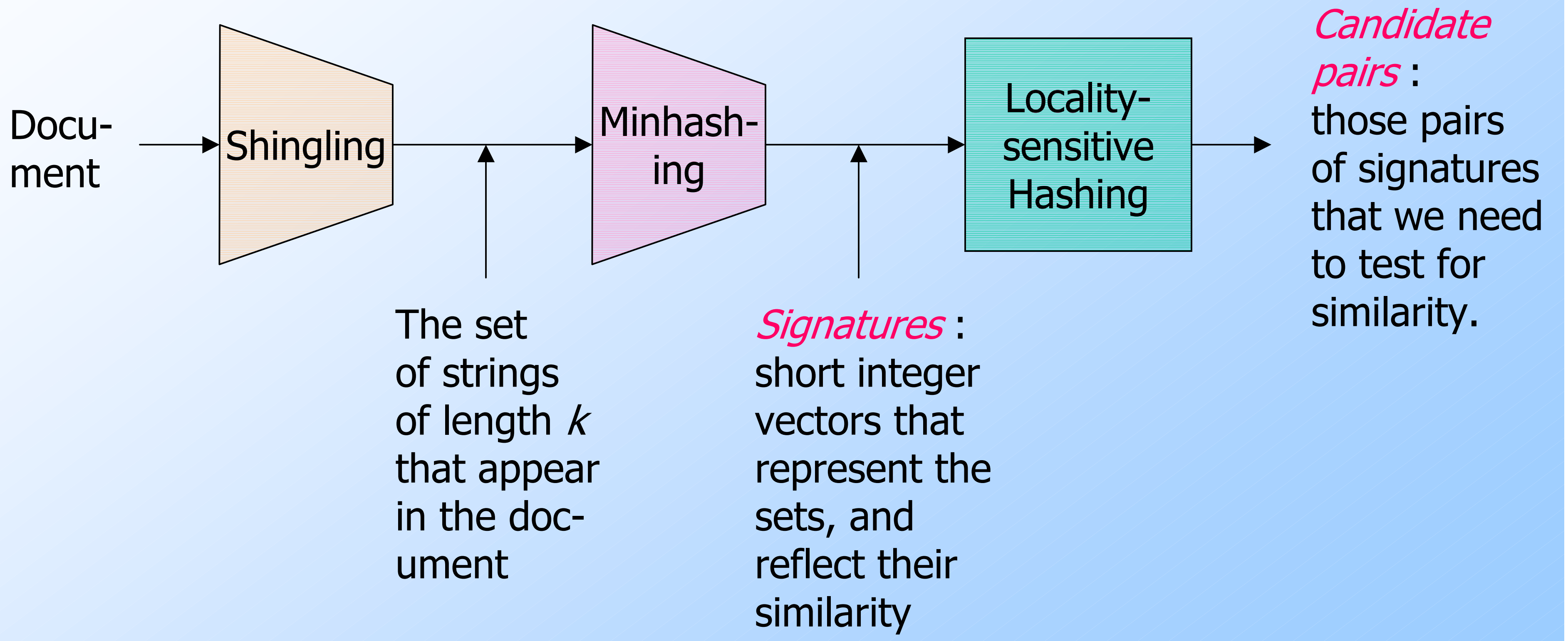
Similar Documents – (2)

- Given a body of documents, e.g., the Web, find pairs of documents with a lot of text in common, e.g.:
  - Mirror sites, or approximate mirrors.
    - Application: Don't want to show both in a search.
  - Plagiarism, including large quotations.
  - Similar news articles at many news sites.
    - Application: Cluster articles by "same story."

Three Essential Techniques for Similar Documents

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



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

Shingles

- A *k* -shingle (or *k* -gram) for a document is a sequence of *k* characters that appears 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.

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; $k = 10$ is better for long documents.

Shingles: Compression Option

- To compress long shingles, we can hash them to (say) 4 bytes.
- Represent a doc by the set of hash values of its *k*-shingles.
- Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.

Thought Question

- Why is it better to hash 9-shingles (say) to 4 bytes than to use 4-shingles?
- Hint: How random are the 32-bit sequences that result from 4-shingling?

MinHashing

Data as Sparse Matrices

Jaccard Similarity Measure

Constructing Signatures

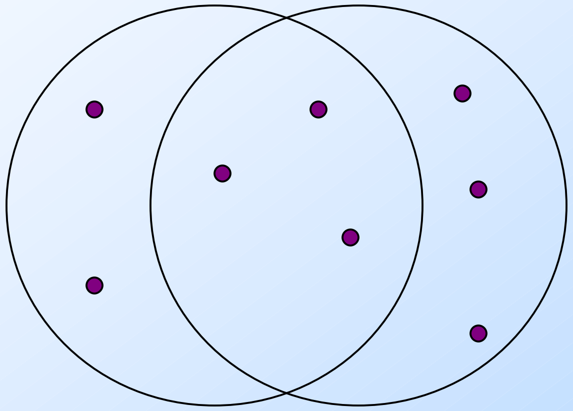
Basic Data Model: Sets

- Many similarity problems can be couched 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.

Jaccard Similarity of Sets

- The *Jaccard similarity* of two sets is the size of their intersection divided by the size of their union.
    - *Sim* $(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$.

Example: Jaccard Similarity



3 in intersection.
8 in union.
Jaccard similarity
= 3/8

From Sets to Boolean Matrices

- Rows = elements of the universal set.
- Columns = sets.
- 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 sets of their rows with 1.
- Typical matrix is sparse.

Example: Jaccard Similarity of Columns

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

Sim $(C_1, C_2) =$

$2/5 = 0.4$

Aside

- We might not really represent the data by a boolean matrix.
- Sparse matrices are usually better represented by the list of places where there is a non-zero value.
- But the matrix picture is conceptually useful.

When Is Similarity Interesting?

1. When the sets are so large or so many that they cannot fit in main memory.
2. Or, when there are so many sets that comparing all pairs of sets takes too much time.
3. Or both.

Outline: Finding Similar Columns

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

1. Comparing all pairs of signatures may take too much time, even if not too much space.
   - A job for Locality-Sensitive Hashing.
2. These methods can produce false negatives, and even false positives (if the optional check is not made).

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 the same as the "similarity" of *Sig* (C_1) and *Sig* (C_2).

Four Types of Rows

- 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

- Also, a = # rows of type a, etc.
- Note $Sim(C_1, C_2) = a/(a+b+c)$.

Minhashing

- Imagine the rows permuted randomly.
- Define "hash" function $h(C)$ = the number 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

Input matrix

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

$\Rightarrow$

Signature matrix *M*

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

Surprising Property

- The probability (over all permutations of the rows) that $h(C_1) = h(C_2)$ is the same as $Sim(C_1, C_2)$.
- Both are $a/(a+b+c)$!
- Why?
  - Look down the permuted columns C_1 and C_2 until we see a 1.
  - If it's a type-a row, then $h(C_1) = h(C_2)$. If a type-b or type-c row, then not.

Similarity for Signatures

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

Min Hashing – Example

Input matrix

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

  according to the *i* th permutation, the number of the first row that has a 1 in column *C*.

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.

Implementation – (2)

- A good approximation to permuting rows: pick 100 (?) hash functions.
- For each column c and each hash function h_i, keep a "slot" $M(i, c)$.
- Intent: $M(i, c)$ will become the smallest value of $h_i(r)$ for which column c has 1 in row r.
  - I.e., $h_i(r)$ gives order of rows for ith permuation.

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
                    $M(i, c) := h_i(r)$;
```

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$
$g(x) = 2x+1 \bmod 5$

	Sig1	Sig2
$h(1) = 1$	1	-
$g(1) = 3$	3	-
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

Implementation – (4)

- Often, data is given by column, not row.
  - E.g., columns = documents, rows = shingles.
- If so, sort matrix once so it is by row.
- And *always* compute $h_i(r)$ only once for each row.

Locality-Sensitive Hashing

Focusing on Similar Minhash Signatures

Other Applications Will Follow

Finding Similar Pairs

- Suppose we have, in main memory, data representing a large number of objects.
  - May be the objects themselves .
  - May be signatures as in minhashing.
- We want to compare each to each, finding those pairs that are sufficiently similar.

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 $5*10^{11}$ column-comparisons.
- At 1 microsecond/comparison: 6 days.

Locality-Sensitive Hashing

- General idea: Use a function f(x,y) that tells whether or not *x* and *y* is a *candidate pair* : a pair of elements whose similarity must be evaluated.
- For minhash matrices: Hash columns to many buckets, and make elements of the same bucket candidate pairs.

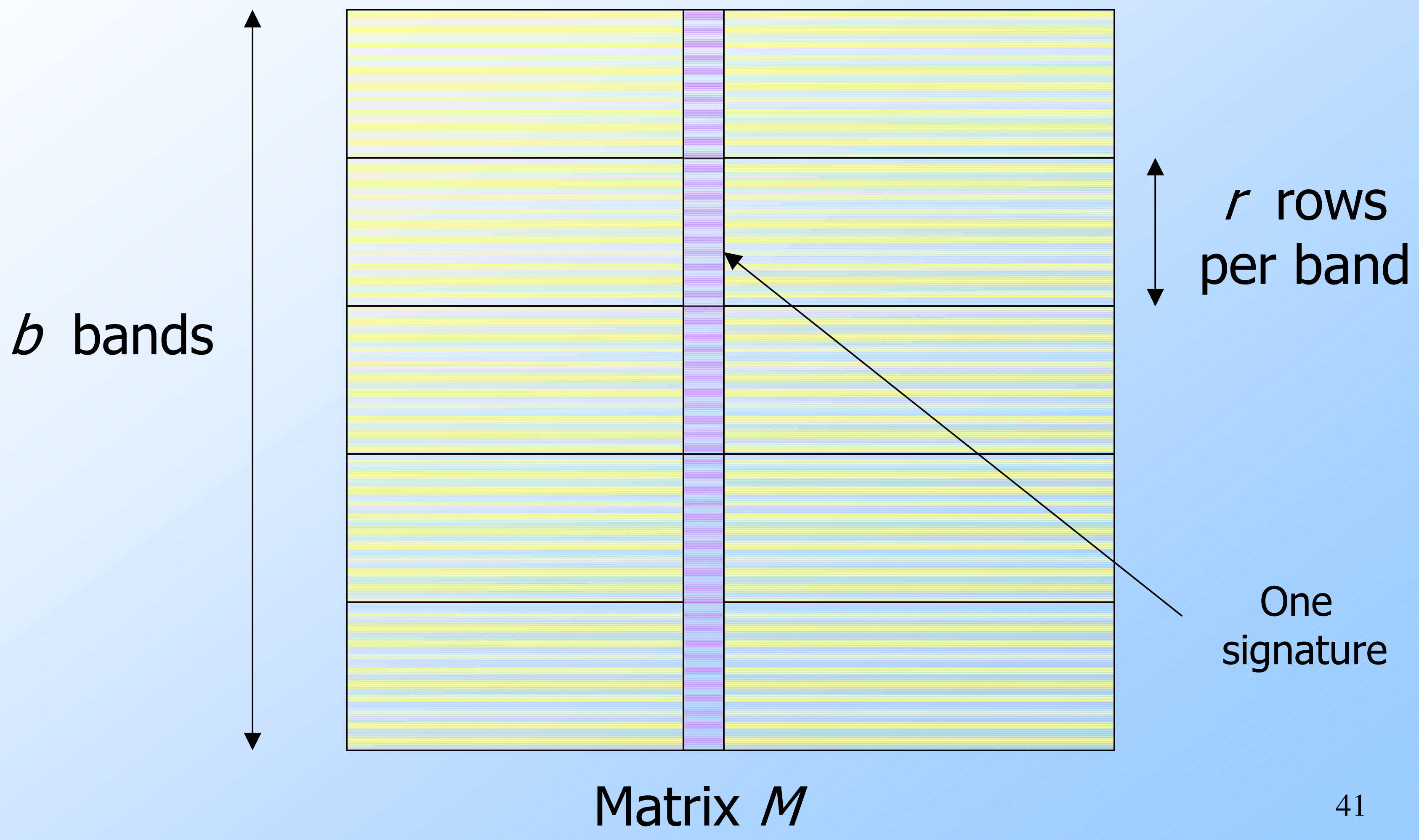
Candidate Generation From Minhash Signatures

- Pick a similarity threshold s, a fraction < 1.
- A pair of columns c and d is a *candidate pair* 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

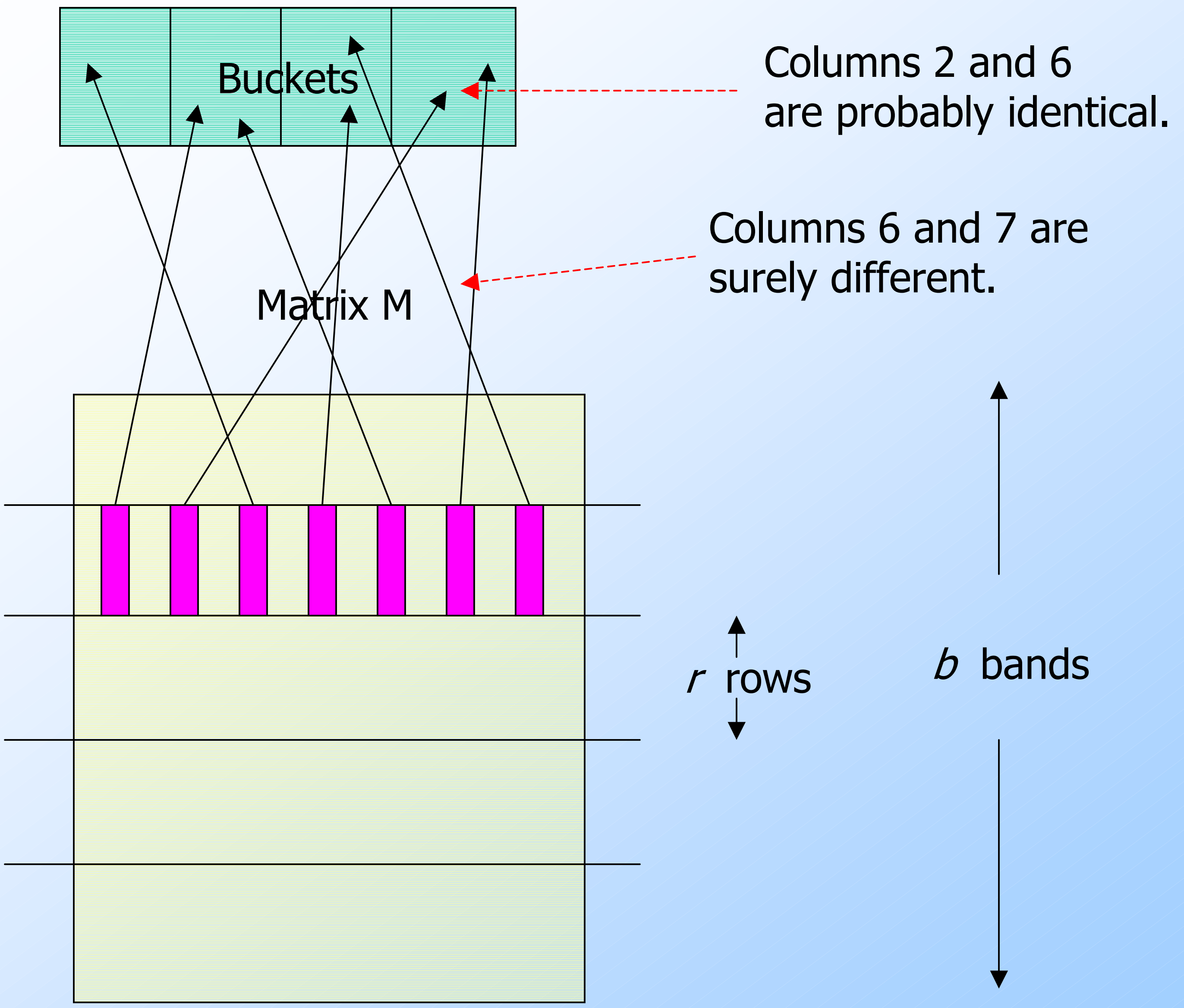
- Big idea: hash columns of signature matrix *M* several times.
- Arrange that (only) similar columns are likely to hash to the same bucket.
- Candidate pairs are those that hash at least once to the same bucket.

Partition Into 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.
  - 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 nonsimilar pairs.



Columns 2 and 6 are probably identical.

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."

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* similar in any of the 20 bands: $(1-0.328)^{20} = .00035$.
    - i.e., about 1/3000th of the 80%-similar column pairs are false negatives.

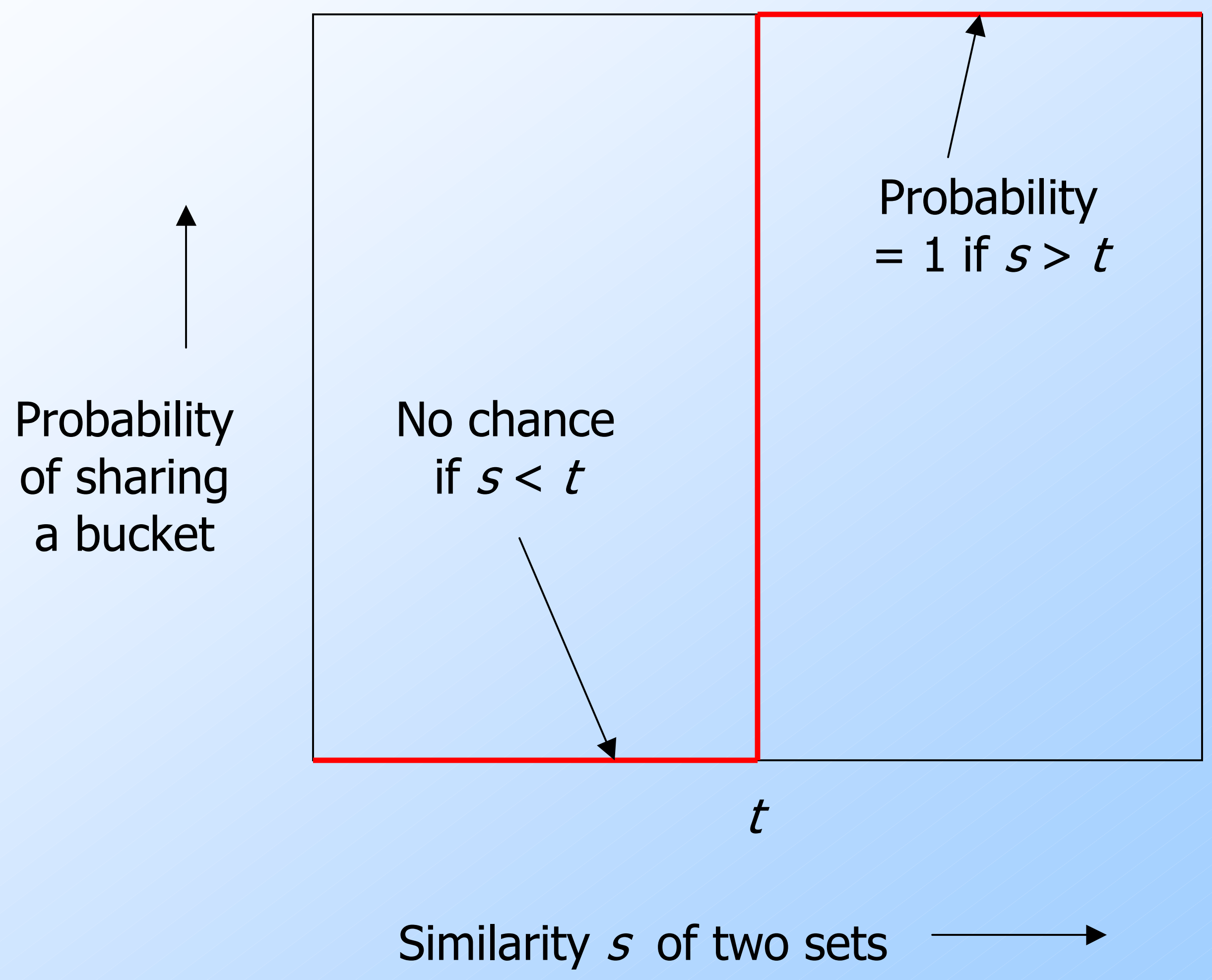
Suppose C_1, C_2 Only 40% Similar

- Probability C_1, C_2 identical in any one particular band: $(0.4)^5 = 0.01$.
- Probability C_1, C_2 identical in $\geq$ 1 of 20 bands: $\leq 20 * 0.01 = 0.2$.
- But false positives much lower for similarities << 40%.

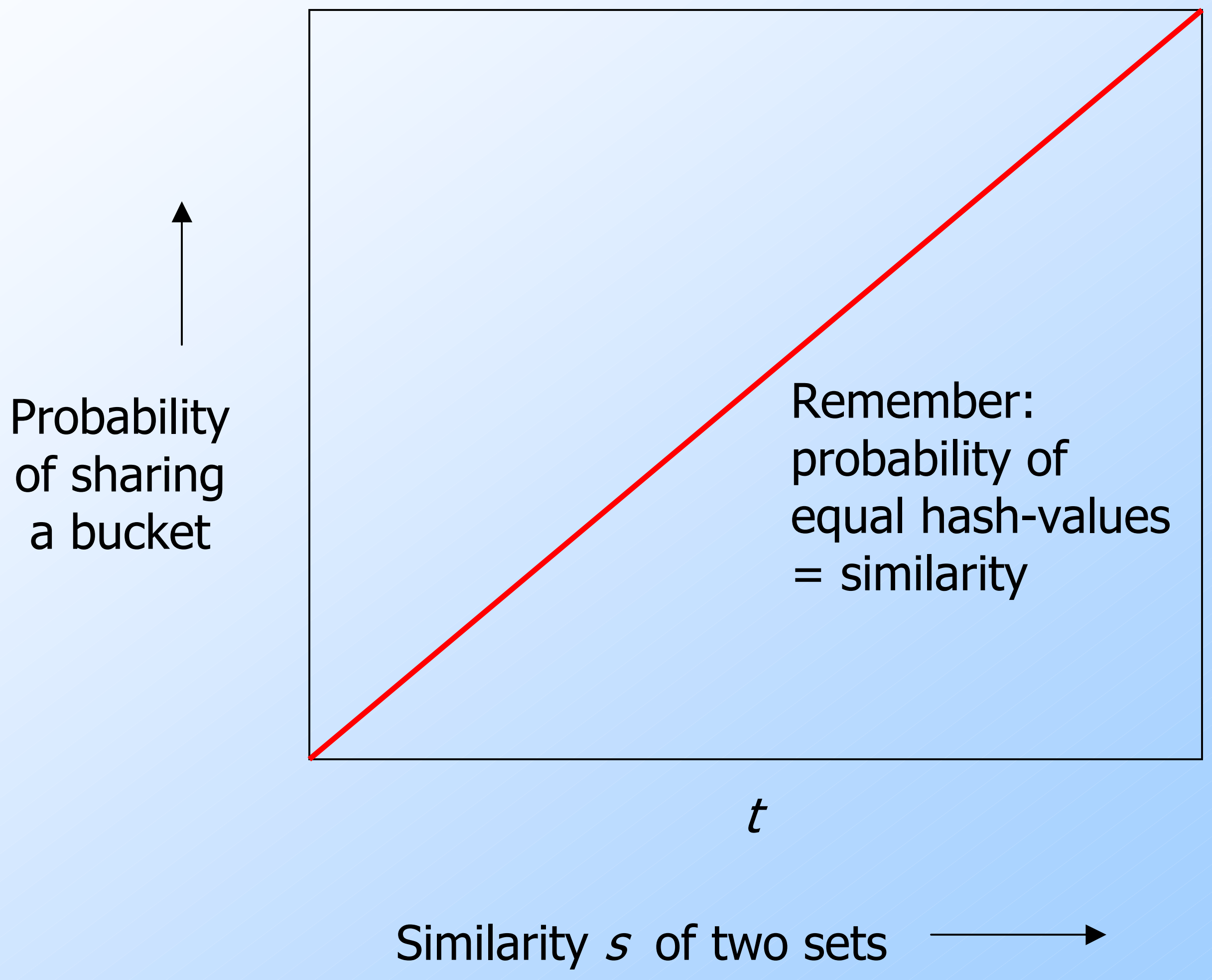
LSH Involves a Tradeoff

- Pick the number of minhashes, the number of bands, and the number of rows 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.

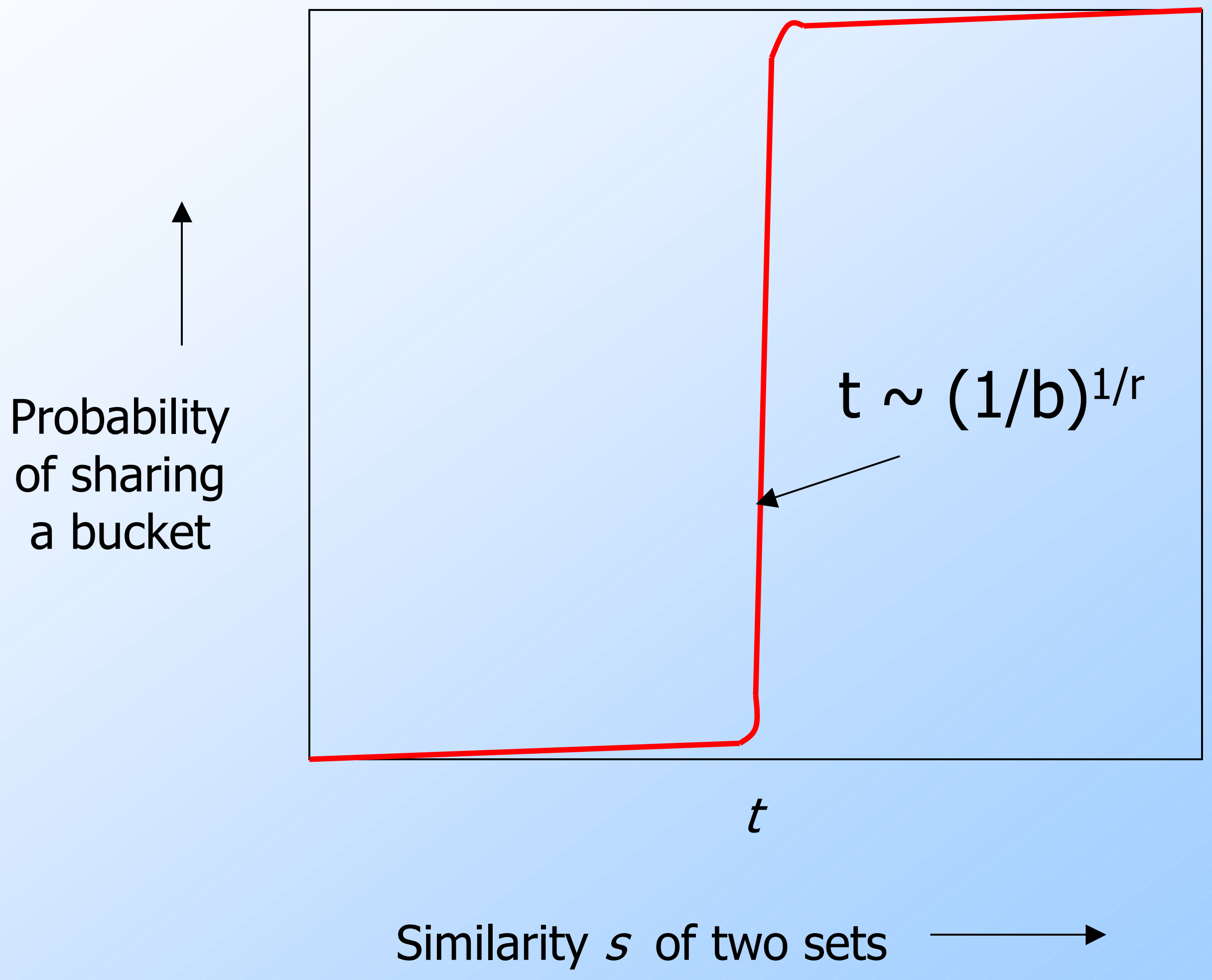
Analysis of LSH – What We Want



What One Band of One Row Gives You



What *b* Bands of *r* Rows Gives You



At least one band identical

No bands identical

$$1 - (1 - s^{r})^{b}$$

Some row of a band unequal

All rows of a band are equal

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

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

LSH Summary

- Tune 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 *sets* .