Finding Similar Items

Application: Similar Documents
Shingling
Minhashing
Locality-Sensitive Hashing

Jeffrey D. Ullman
Stanford University

The Big Picture

- It has been said that the mark of a computer scientist is that they believe hashing is real.
  - I.e., it is possible to insert, delete, and lookup items in a large set in O(1) time per operation.
- *Locality-Sensitive Hashing* (LSH) is another type of magic that, like Bigfoot, is hard to believe is real, until you've seen it.
- It lets you find pairs of similar items in a large set, without the quadratic cost of examining each pair.

LSH: The Bigfoot of CS

- LSH is really a family of related techniques.
- In general, one throws items into buckets using several different "hash functions."
- You examine only those pairs of items that share a bucket for at least one of these hashings.
- Upside: designed correctly, only a small fraction of pairs are ever examined.
- Downside: there are *false negatives* – pairs of similar items that never even get considered.

Some Applications

- We shall first study in detail the problem of finding (lexically) similar documents.
- Later, two other problems:
  - *Entity resolution* (records that refer to the same person or other entity).
  - News-article similarity.

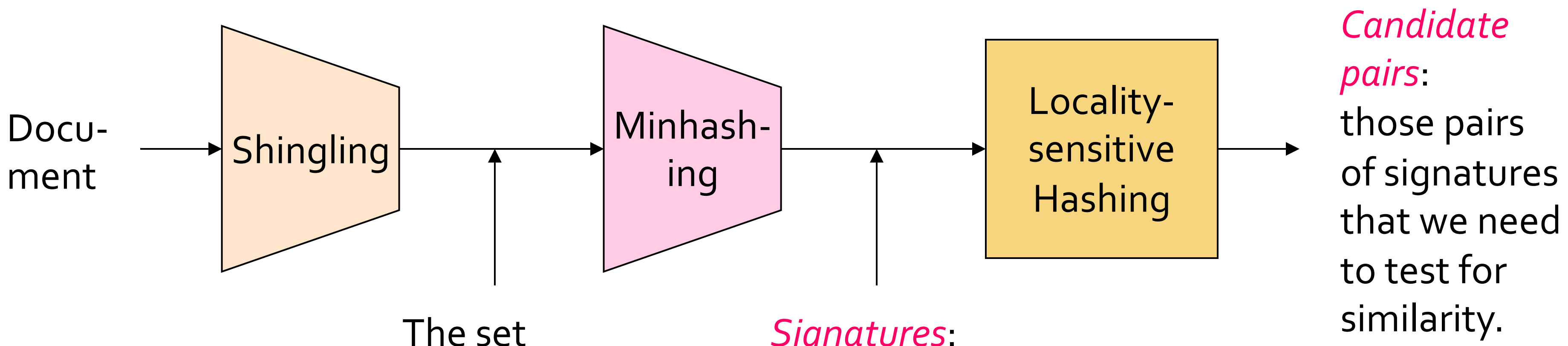
Similar Documents

- Given a body of documents, e.g., the Web, find pairs of documents with a lot of text in common, such as:
  - 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.”
- Warning: LSH not designed for “same topic.”

Three Essential Techniques for Similar Documents

1. *Shingling*: convert documents, emails, etc., to sets.
2. *Minhashing*: convert large sets to short signatures (lists of integers), 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

Note: Similarity of sets = "Jaccard similarity" (defined later) = a measure of how many elements are shared.

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

Shingles and Similarity

- Documents that are textually similar will have many shingles in common.
- Changing a word only affects k-shingles within distance k-1 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_, and h_c.

Compression Option

- Intuition: want enough possible shingles that most docs do not contain most shingles.
  - k = 8-, 9-, or 10-character shingles is often used in practice.
- Character strings are not “random” bit strings, so they take more space than needed.

Tokens

- To save space but still make each shingle rare, we can hash them to (say) 4 bytes.
    - Called *tokens*.
- Represent a doc by its tokens, that is, 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.

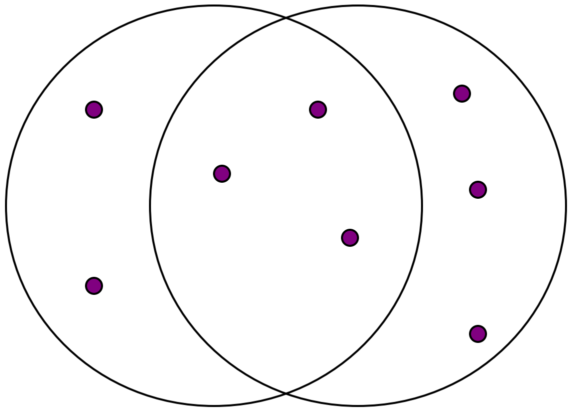
Minhashing

Jaccard Similarity Measure
Constructing Signatures

Jaccard Similarity

- 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.
  - Examples: the set of all k-shingles or all tokens.
- Columns = sets.
- 1 in row *e* and column *S* if and only if *e* is a member of *S*; else 0.
- *Column similarity* is the Jaccard similarity of the sets of their rows with 1.
- Typical matrix is sparse.
- Warning: We don't really construct the matrix; just imagine it exists.

Example: Column Similarity

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

$\mathrm{Sim}(C_1, C_2) =$

$2/5 = 0.4$

Four Types of Rows

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

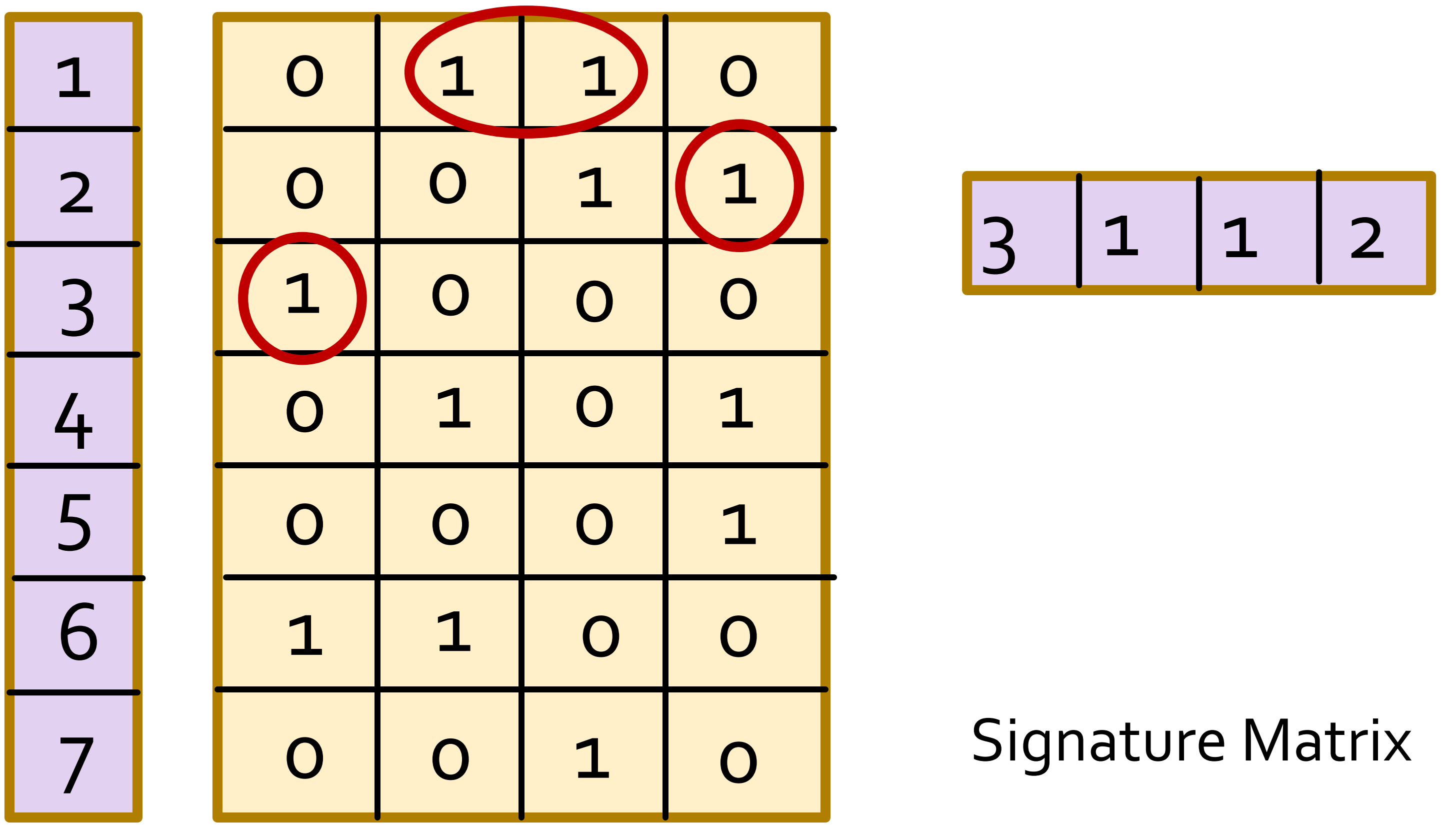
	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

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

Minhashing

- Permute the rows.
  - Thought experiment – not real.
- 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, rows = minhash values, in order for that column.

Example: Minhashing



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

Input Matrix

3	1	1	2

Signature Matrix

Example: Minhashing

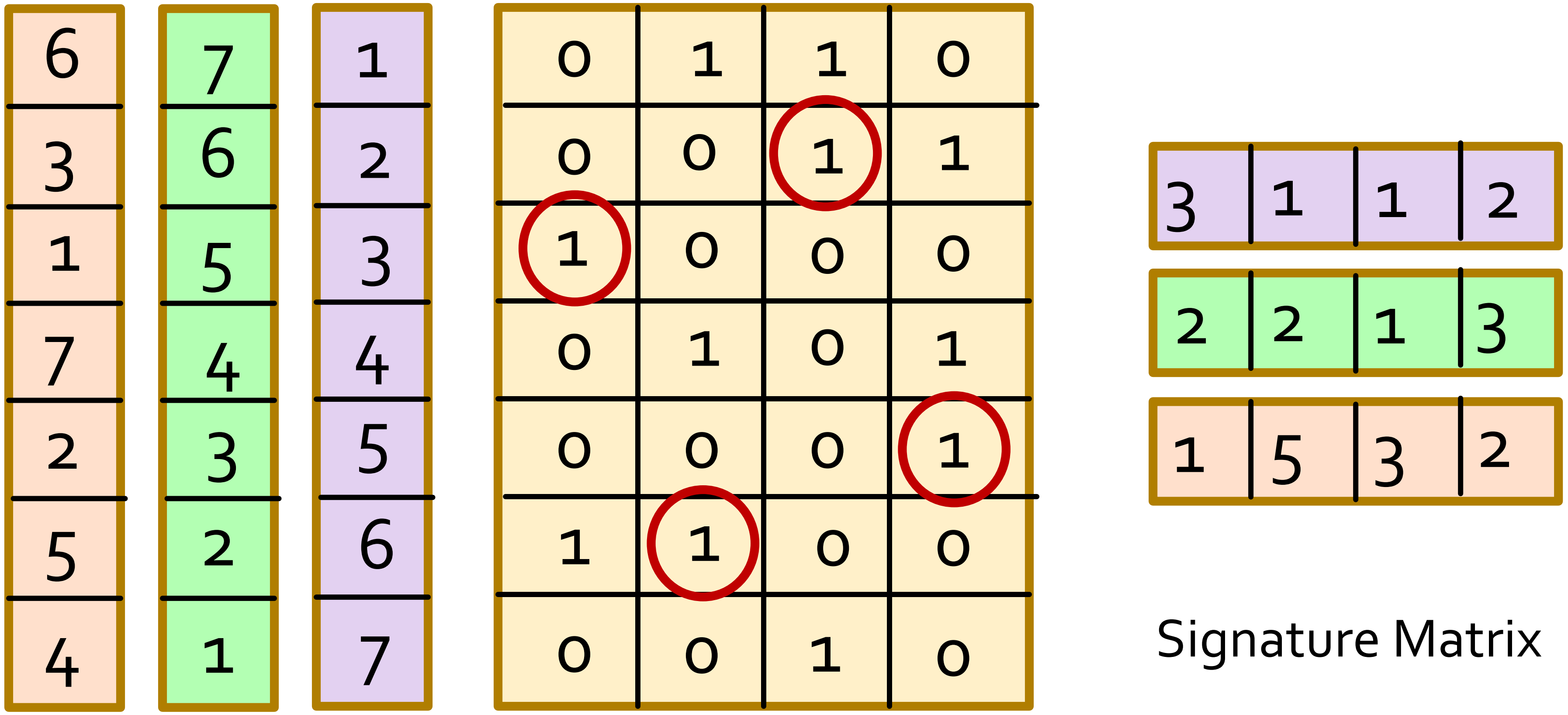
7	1	0	1	1	0
6	2	0	0	1	1
5	3	1	0	0	0
4	4	0	1	0	1
3	5	0	0	0	1
2	6	1	1	0	0
1	7	0	0	1	0

Input Matrix

3	1	1	2
2	2	1	3

Signature Matrix

Example: Minhashing



6	7	1	0	1	1	0
3	6	2	0	0	$\textcircled{1}$	1
1	5	3	$\textcircled{1}$	0	0	0
7	4	4	0	1	0	1
2	3	5	0	0	0	$\textcircled{1}$
5	2	6	1	$\textcircled{1}$	0	0
4	1	7	0	0	1	0

Input Matrix

3	1	1	2
2	2	1	3
1	5	3	2

Signature Matrix

A Subtle Point

- People sometimes ask whether the minhash value should be the original number of the row, or the number in the permuted order (as we did in our example).
- Answer: it doesn’t matter.
- You only need to be consistent, and assure that two columns get the same value if and only if their first 1’s in the permuted order are in the same row.

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?
  - Already know $Sim(C_1, C_2) = a/(a+b+c)$.
  - 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 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.

Example: Similarity

Columns 1 & 2:
Jaccard similarity 1/4.
Signature similarity 1/3

Columns 2 & 3:
Jaccard similarity 1/5.
Signature similarity 1/3

Columns 3 & 4:
Jaccard similarity 1/5.
Signature similarity 0

0	1	1	0
0	0	1	1
1	0	0	0
0	1	0	1
0	0	0	1
1	1	0	0
0	0	1	0

Input Matrix

3	1	1	2
2	2	1	3
1	5	3	2

Signature Matrix

Implementation of Minhashing

- Suppose 1 billion rows.
- Hard to pick a random permutation of 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, say, 100 hash functions.
- Intuition: the resulting permutation is what you get by sorting rows in order of their hash values.
- 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*.

Implementation – (3)

```
for (all $i$ and $c$) $M(i, c) := \infty$;
for (each row $r$) begin
    for (each hash function $h_i$ )
        compute $h_i(r)$;
for (each column $c$)
        if (c has 1 in row $r$)
            for (each hash function $h_i$)
                if ($h_i(r)$ is smaller than $M(i, c)$) then
                    $M(i, c) := h_i(r)$;
 end;
```

Important: so you hash r only once per hash function, not once per 1 in row 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.
  - Example: columns = documents, rows = shingles.
- If so, sort matrix once so it is by row.
  - I.e., generate shingle-docID pairs from the documents and then sort by shingle.

Interesting Improvement

- From Li-Owen-Zhang, Stanford Statistics Dept.
- Cost of minhashing is proportional to the number of rows.
- Suppose we only went a small way down the list of rows, e.g., hashed only the first 1000 rows.
- Advantage: Saves a lot of time.
- Disadvantage: if all 1000 rows have 0 in a column, you get no minhash value.
    - It is a mistake to assume two columns hashing to no value are likely to be similar.

Improvement – (2)

- Divide the rows into k bands.
- As you go down the rows, start a new minhash competition for each band.
- Thus, to get a desired number of minhash values, you need to compute only (1/k)th of the number of hash values per row that you would using the original scheme.
  - But don’t make k so large that you often get “no value” for a minhash.

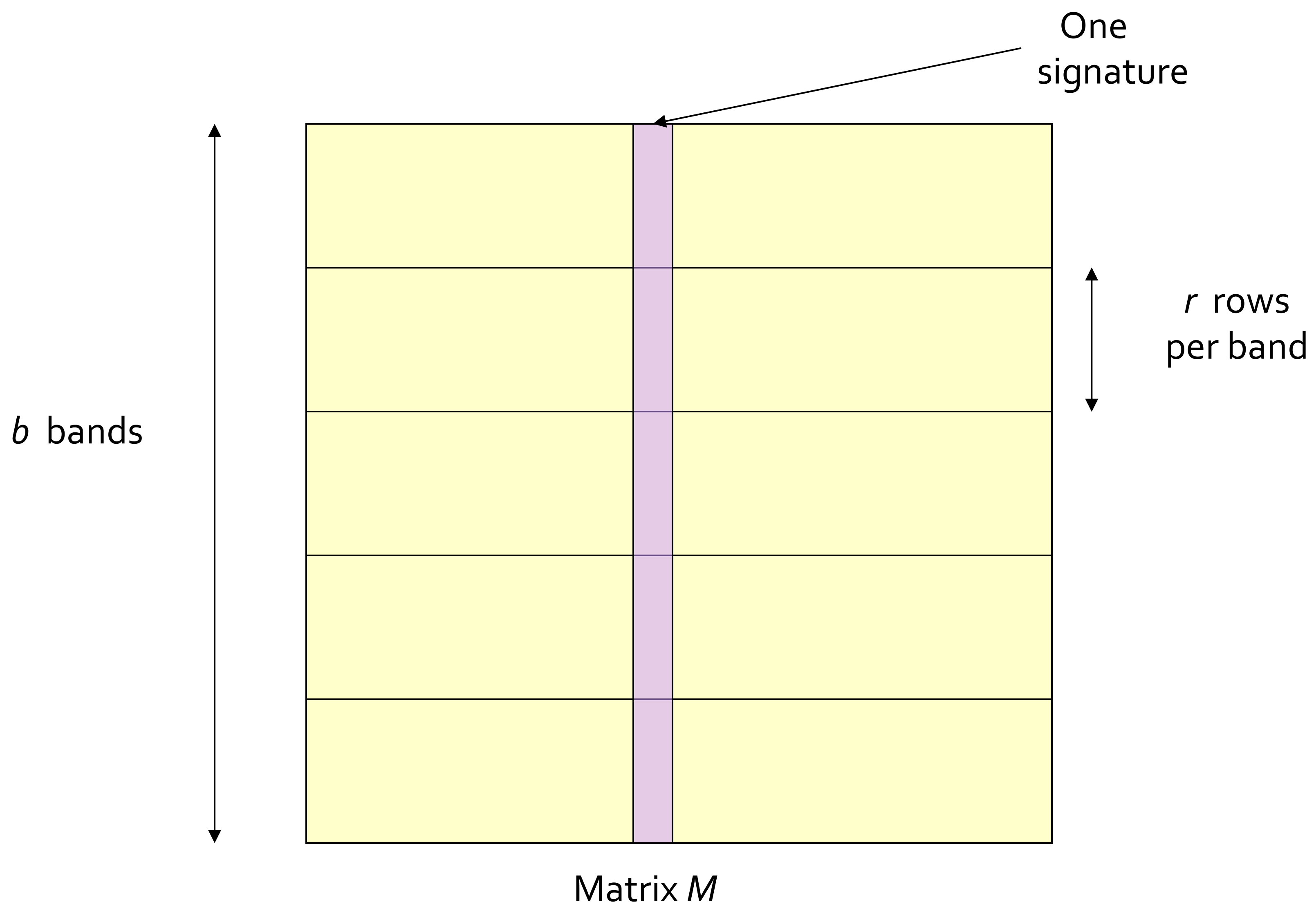
Locality-Sensitive Hashing

Focusing on Similar Minhash Signatures
Other Applications Will Follow

From Signatures to Buckets

- Remember: we want to hash objects such as signatures many times, so that "similar" objects wind up in the same bucket at least once, while other pairs rarely do.
  - *Candidate pairs* are those that share a bucket.
- Define "similar" by a similarity *threshold* t = fraction of rows in which signatures must agree.
- Trick: divide signature rows into bands.
  - Each hash function based on one band.

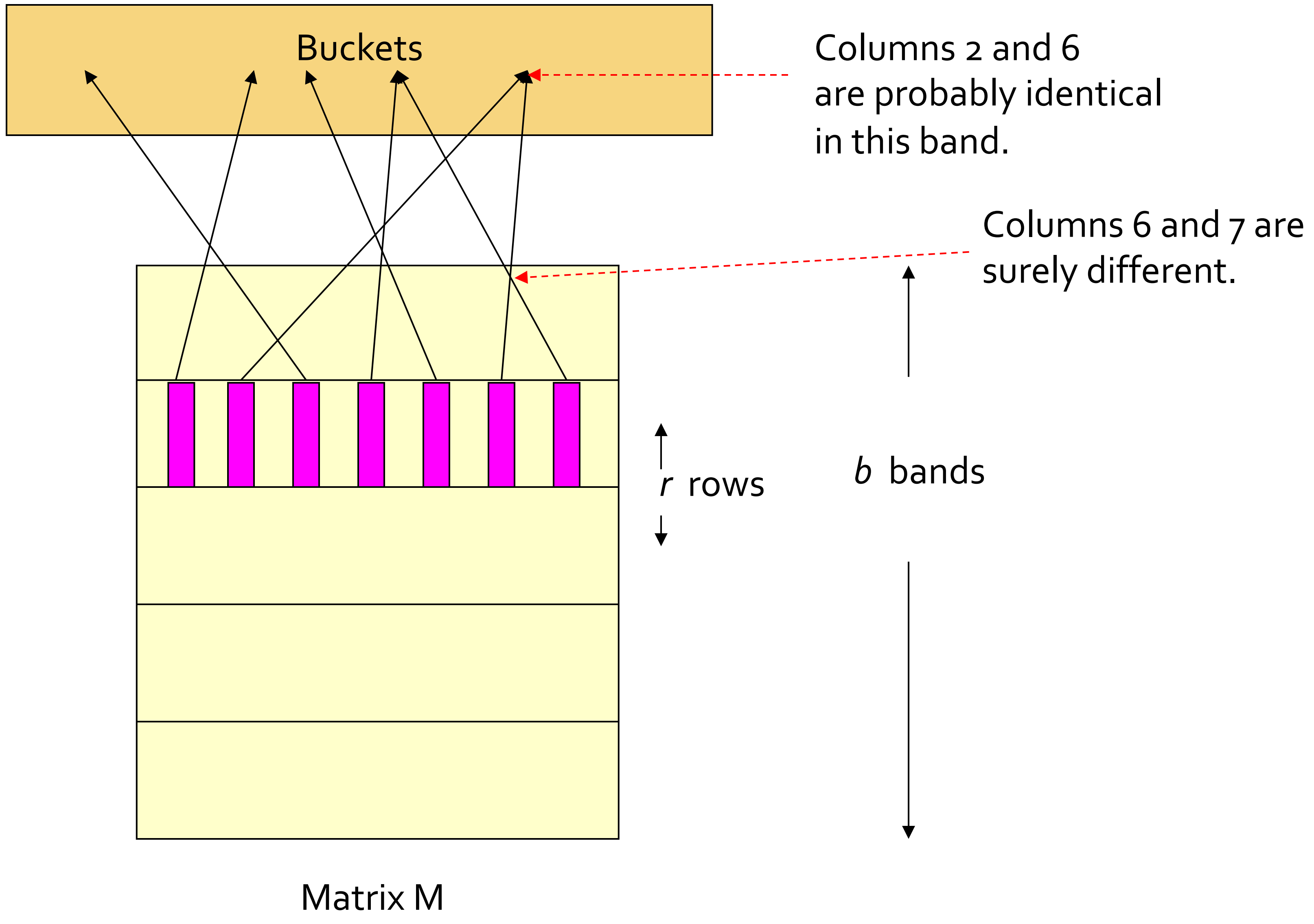
Partition Into Bands



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.
  - Use a different hash table for each band.
- *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.

Hash Function for One Bucket


Buckets
Columns 2 and 6 are probably identical in this band.
Columns 6 and 7 are surely different.
r rows
b bands
Matrix M

Example: 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.

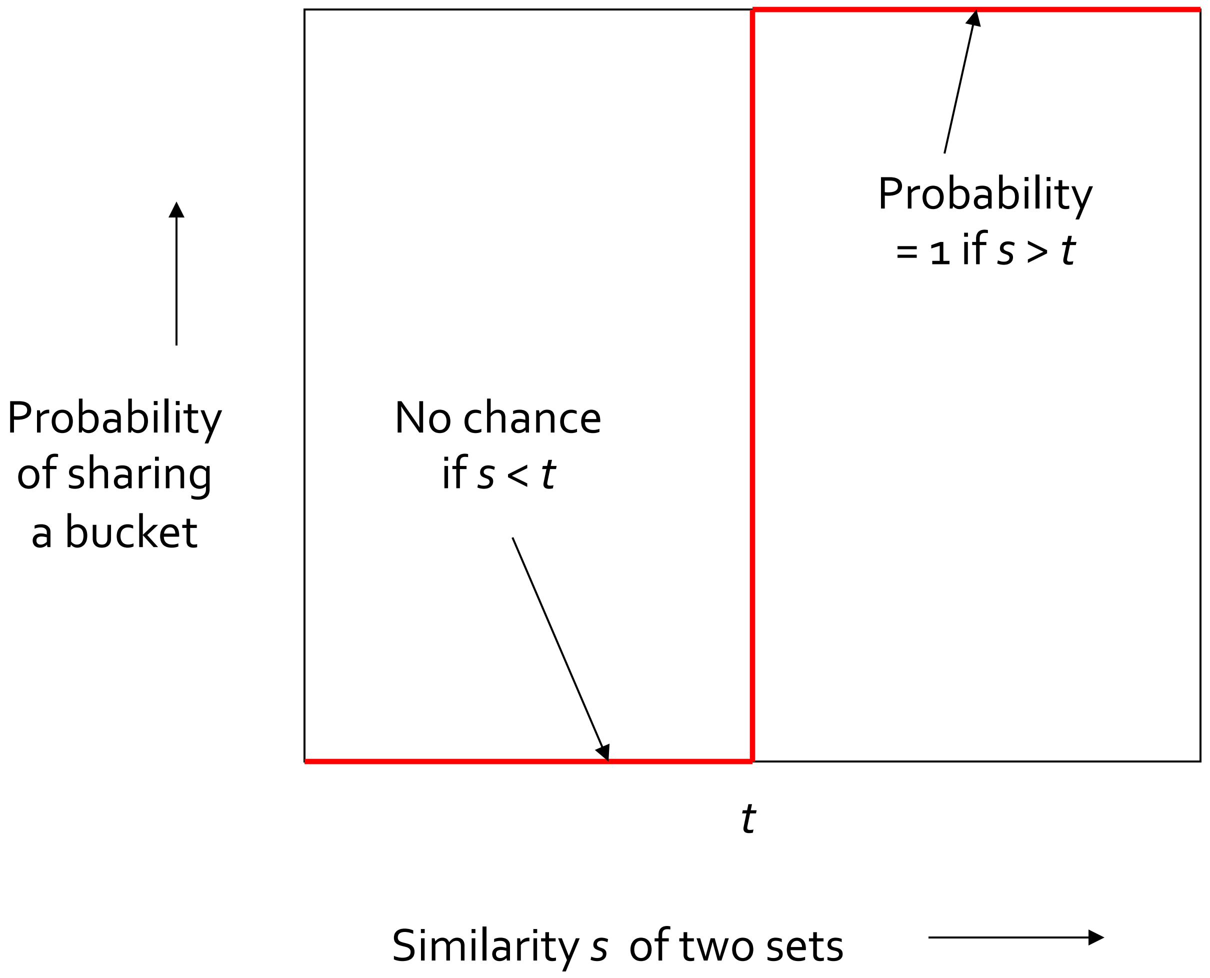
Suppose C_1, C_2 are 80% Similar

- 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: $(1\text{-}0.328)^{20} = .00035$.
  - i.e., about 1/3000th of the 80%-similar underlying sets are false negatives.

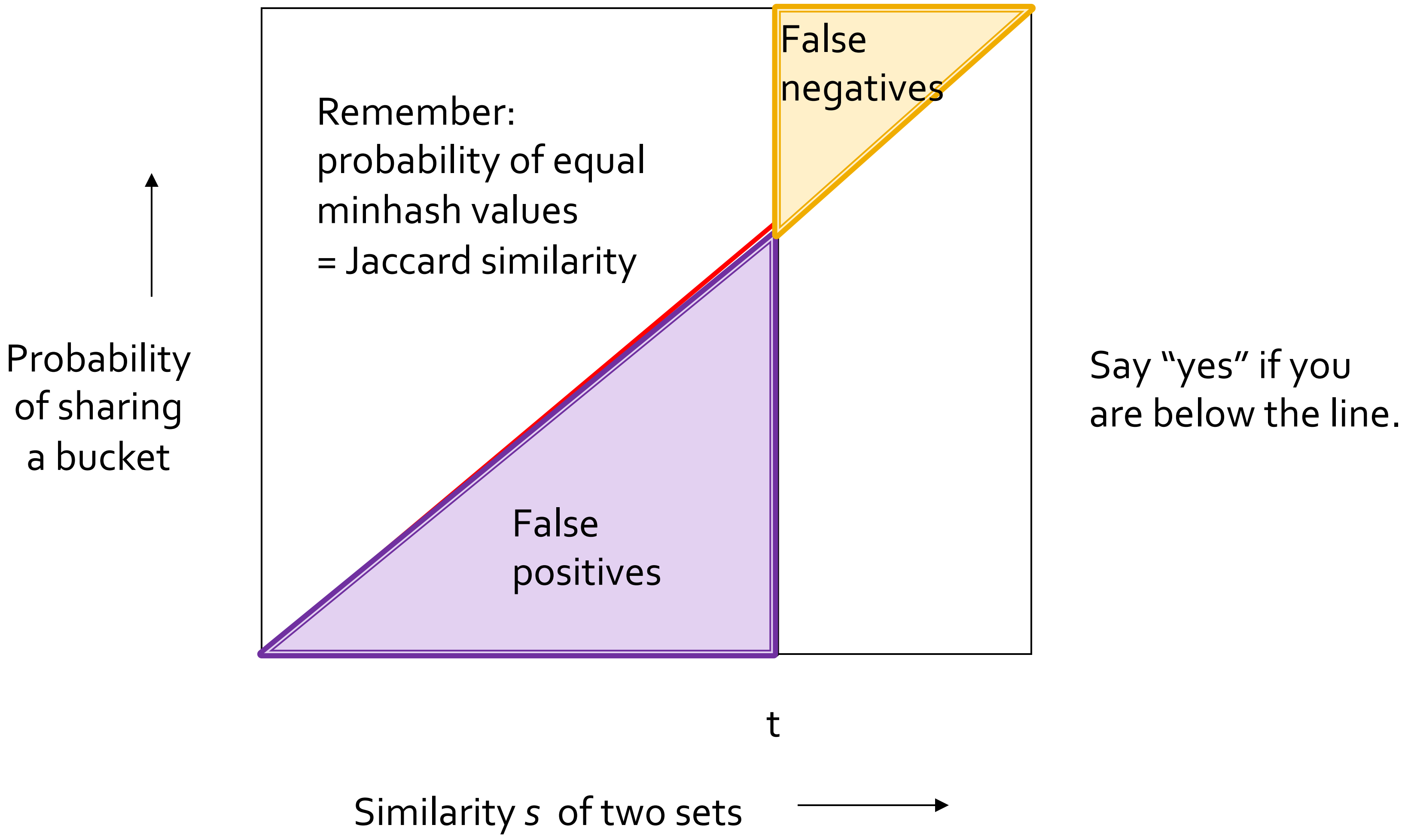
Suppose C_1, C_2 Only 40% Similar

- 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: $1 - (0.99)^{20} < 0.2$.
- But false positives much lower for similarities << 40%.

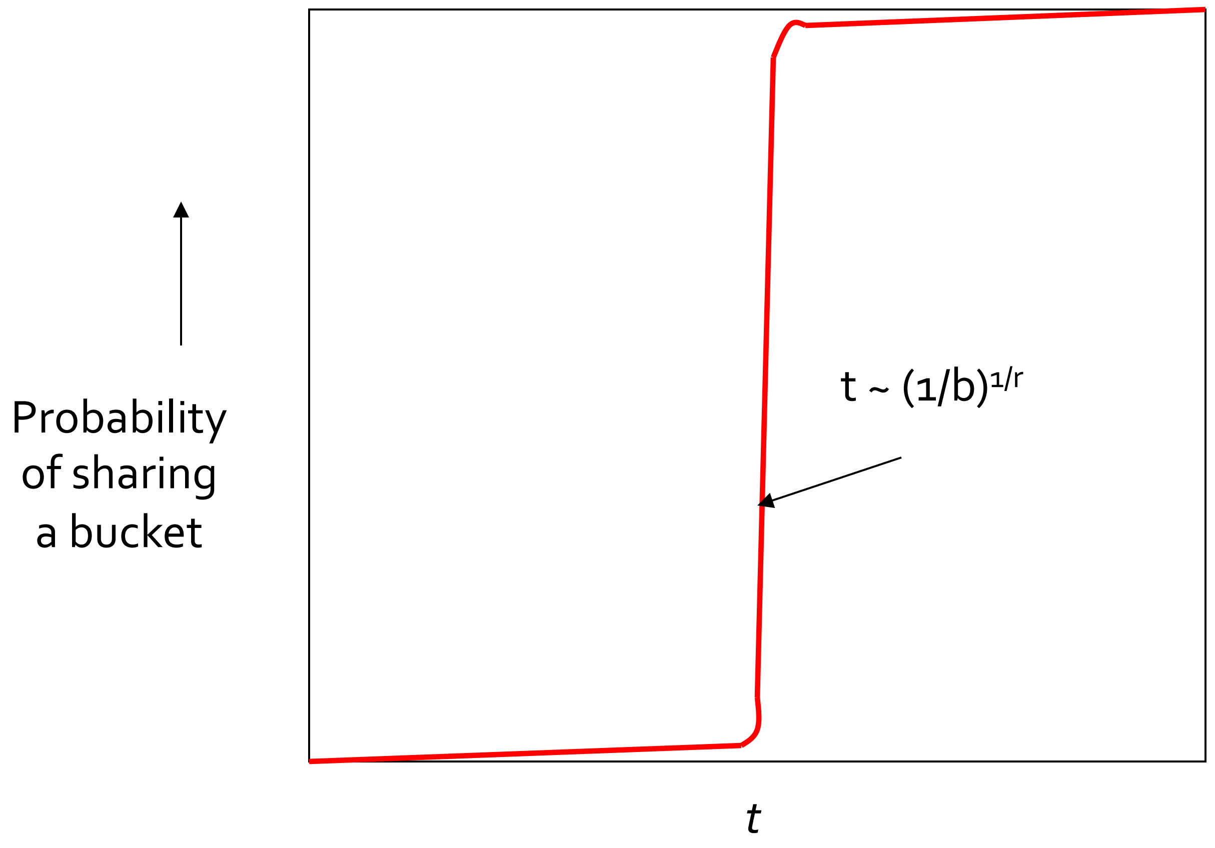
Analysis of LSH – What We Want



What One Band of One Row Gives You



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



Similarity *s* of two sets →

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	$1-(1-s^5)^{20}$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

$\updownarrow$

Slope here about 2.3

LSH for Documents: Summary

- Tune b and r 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*.

Question For Thought

- A student posed the following question, apparently based on a real problem he was having at work.
- He had about a million sets of which he wanted to find the most similar pairs.
- But the universal set had only 52 elements.
- He asked whether he could use the method just outlined to find the similar sets.
- Do you see any problems?