Locality-Sensitive Hashing

Finding Similar Sets
Application to Document Similarity
Shingling
Minhashing

Cloud and Big Data Summer School, Stockholm, Aug., 2015
Jeffrey D. Ullman

Free Book

- You can download a free copy of *Mining of Massive Datasets*, by Jure Leskovec, Anand Rajaraman, and U. at www.mmds.org
- Relevant readings:
  - LSH: 3.1-3.4, 3.8.
  - Stream algorithms: 4.1-4.6.
  - PageRank: 5.1, 5.3-5.5.
  - Clustering: 7.1-7.4.
  - Graph algorithms: 10.2.4-10.2.5, 10.7, 10.8.7.
  - MapReduce theory: 2.5-2.6.

Automated Gradiance Homework

- Go to www.gradiance.com/services
- Create an account for yourself.
  - Passwords are $\geq$10 letters and digits, at least one of each.
- Register for class 3E5A44A9
- You can try homeworks as many times as you like.
- When you submit, you get advice for wrong answers and you can repeat the same problem, but with a different choice of answers.

My Biggest Point

- Machine learning is cool, but it is not all you need to know about mining "big data."
- I'm going to cover some of the other ideas that are worth knowing.

A Fundamental Idea of CS

- How do we find "similar" items in a very large collection of items without looking at every pair?
  - A quadratic process.
- *Locality-sensitive hashing* (LSH) is the general idea of hashing items into bins many times, and looking only at those items that fall into the same bin at least once.
- Hard part: arranging that only high-similarity items are likely to fall into the same bucket.
- Starting point: "similar documents."

Applications of Set-Similarity

Many data-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. Entity resolution.

Similar Documents

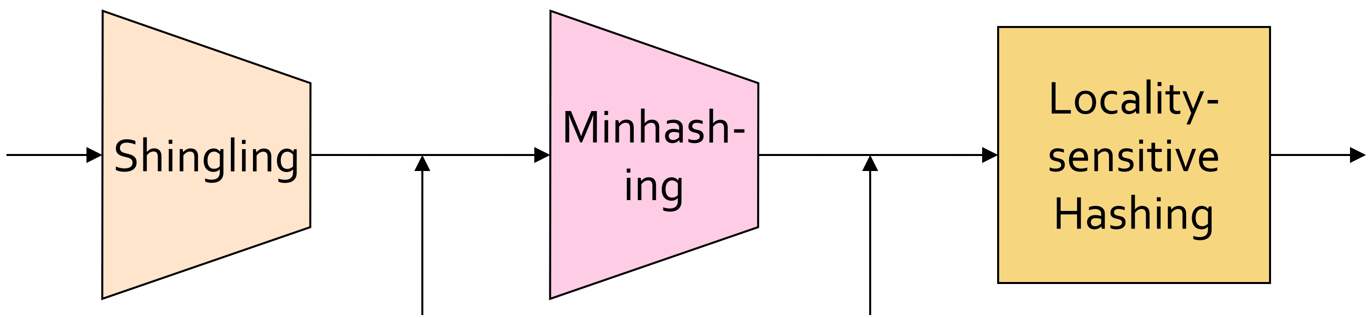
- Given a body of documents, e.g., the Web, find pairs of documents with a lot of text in common, such as:
  - Mirror sites, or approximate mirrors.
    - Application: Don’t want to show both in a search.
  - Plagiarism, including large quotations.
  - Similar news articles at many news sites.
    -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

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

Docu-
ment



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

Shingles and Similarity

- Documents that are intuitively similar will have many shingles in common.
- 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_, and h_c.

Shingles: Compression Option

- To compress long shingles, 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.

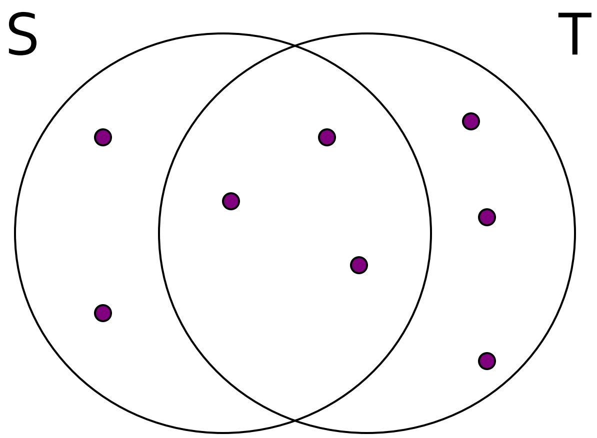
Minhashing

Jaccard Similarity Measure
Definition of Signatures
Constructing Signatures in Practice

Jaccard Similarity

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

Example: Jaccard Similarity



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

From Sets to Boolean Matrices

- Rows = elements of the universal set.
  - Example: the set of all k-shingles.
- 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: Column Similarity

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

$Sim(C_1, C_2) =$

$2/5 = 0.4$

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 *minhash function* $h(C)$ = the first row (in the permuted order) in which column C has 1.
- Use several (e.g., 100) independent hash functions to create a signature for each column.
- The signatures can be displayed in another matrix – the *signature matrix* – whose columns represent the sets and the rows represent the minhash values, in order for that column.

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

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 minhash functions in which they agree.
  - Thinking of signatures as columns of integers, the similarity of signatures is the fraction of 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.

Min Hashing – Example

Permutations:

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

Input matrix

1	2	3	4
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*

1	2	3	4
2	1	2	1
2	1	4	1
1	2	1	2

	1-3	2-4	1-2
Col/Col	0.75	0.75	0
Sig/Sig	0.67	1.00	0

Implementation of Minhashing

- Suppose 1 billion rows.
- 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.

Implementation – (2)

- A good approximation to permuting rows: pick, say, 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 i^{th} permutation.

Implementation – (3)

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;

Example

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

	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

$h(x) = x \bmod 5$, i.e., permutation [5,1,2,3,4]

$g(x) = (2x+1) \bmod 5$, i.e., permutation [2,5,3,1,4]

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.

Locality-Sensitive Hashing

Focusing on Similar Minhash Signatures
Other Applications Will Follow

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.

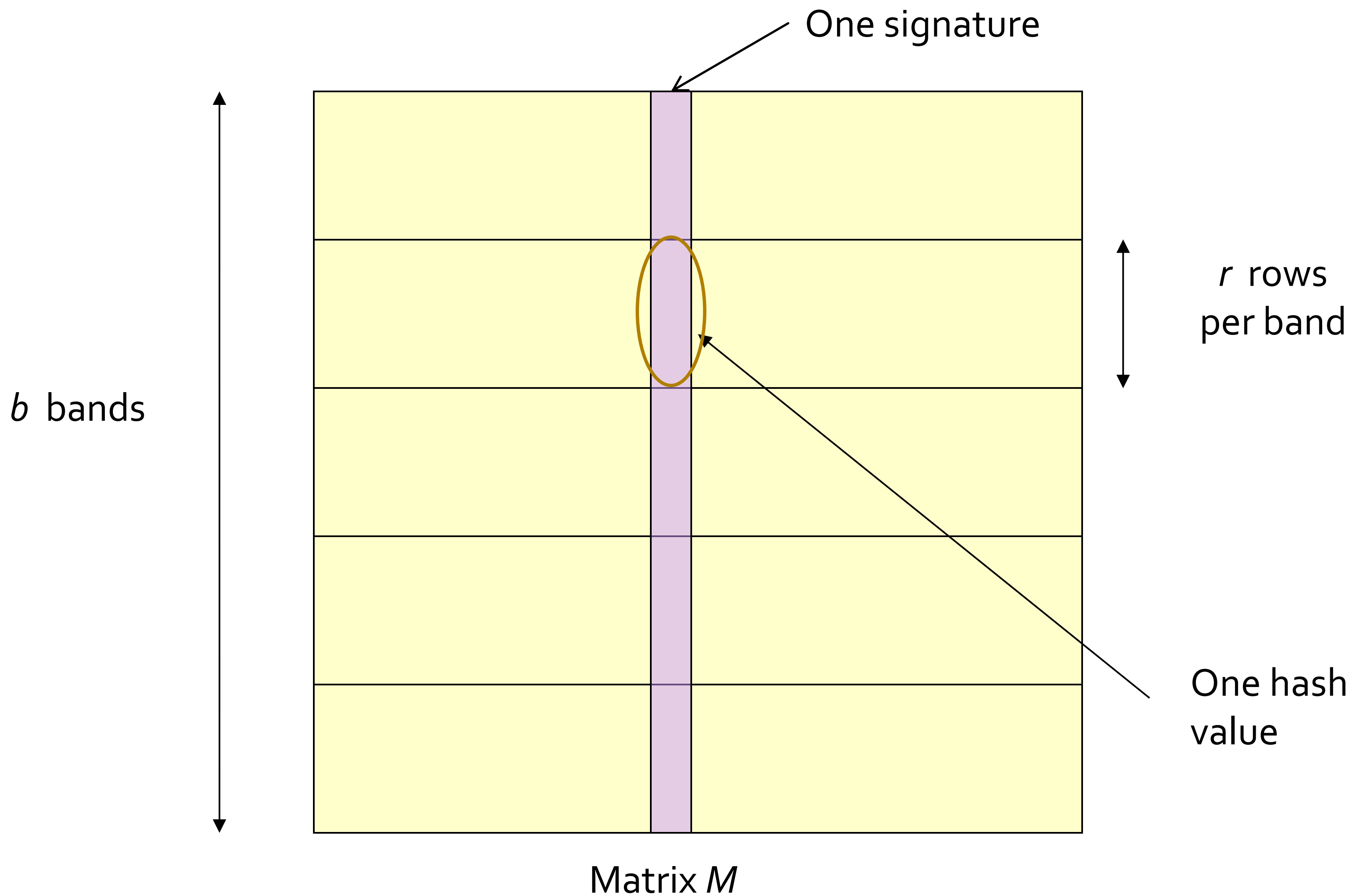
Candidate Generation From Minhash Signatures

- Pick a similarity threshold t, a fraction < 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 t of the rows.
  - I.e., $M(i, c) = M(i, d)$ for at least fraction t 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.

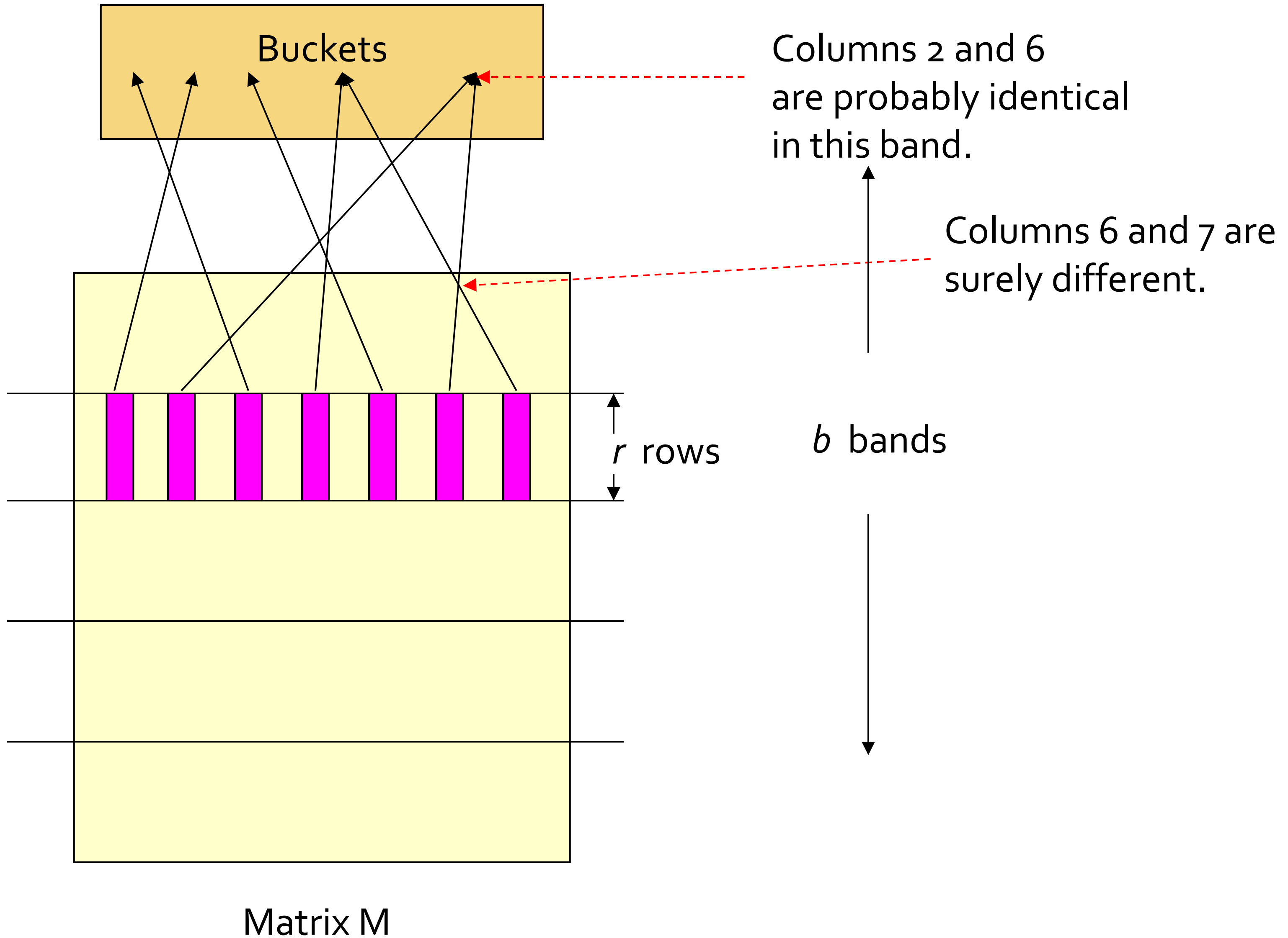
Partition Into Bands



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.

Hash Function for One Bucket



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 integers/band.

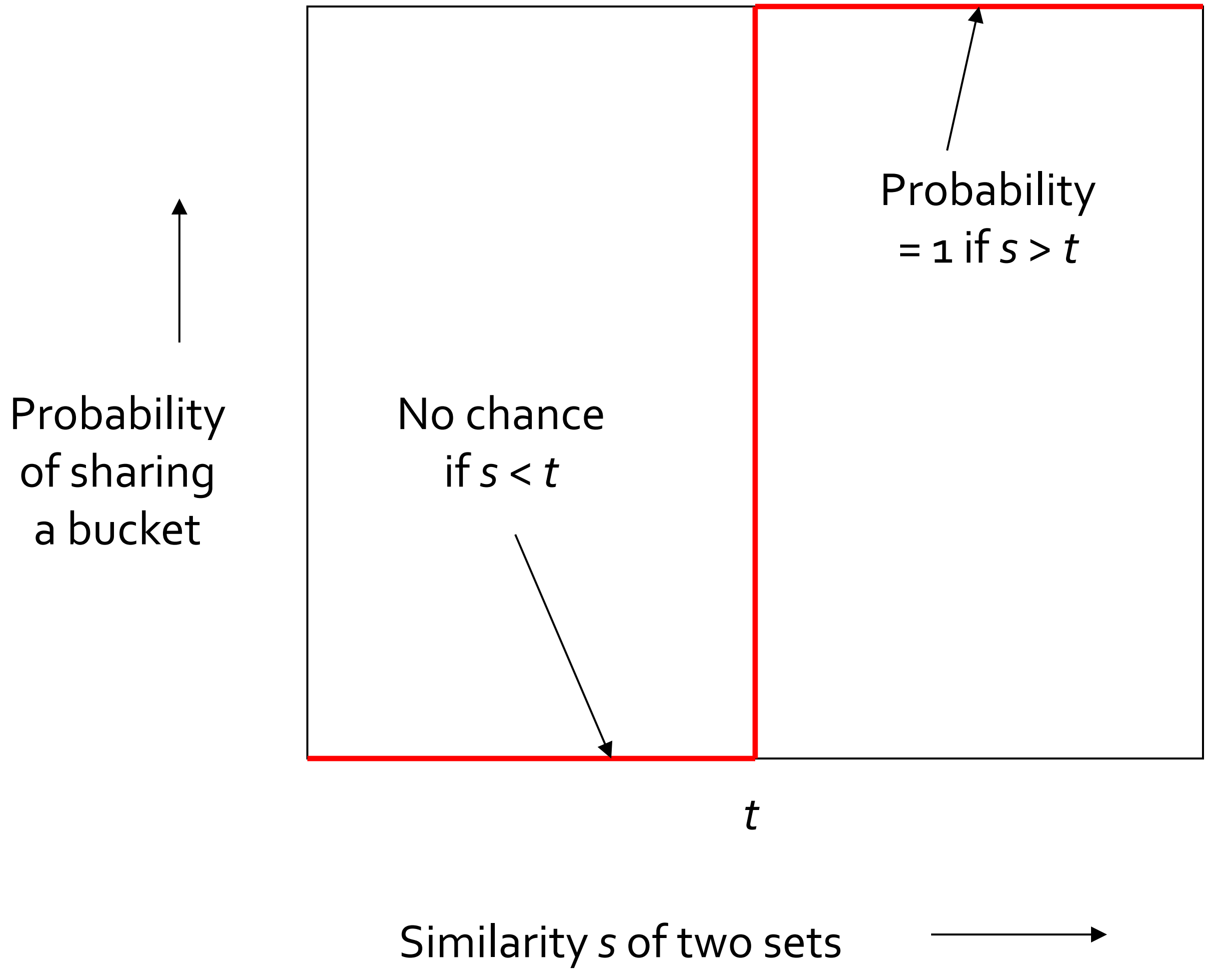
Suppose C_1, C_2 are 80% Similar

- 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 underlying sets are false negatives.

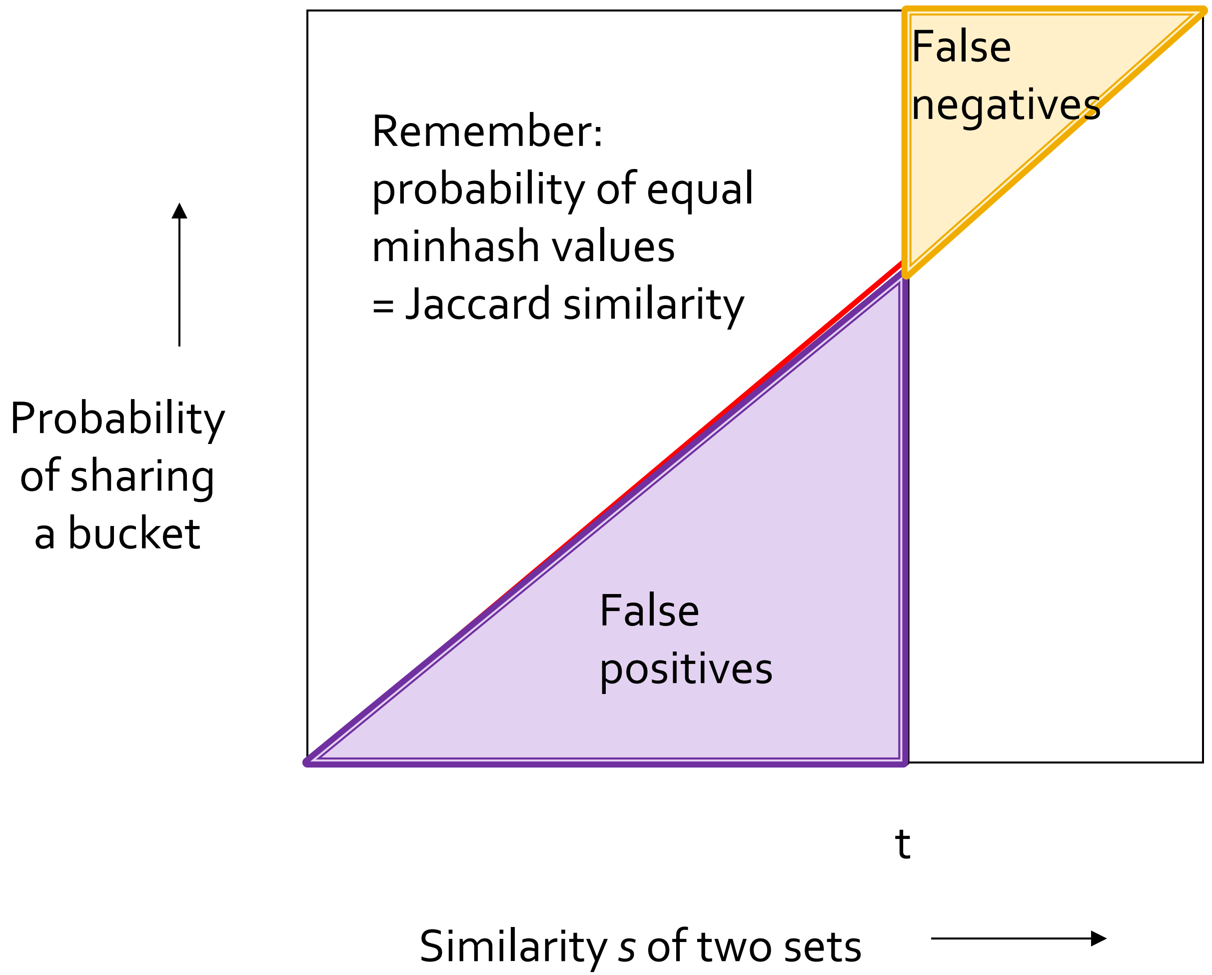
Suppose C_1, C_2 Only 40% Similar

- 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: $\leq 20 * 0.01 = 0.2$.
- But false positives much lower for similarities << 40%.

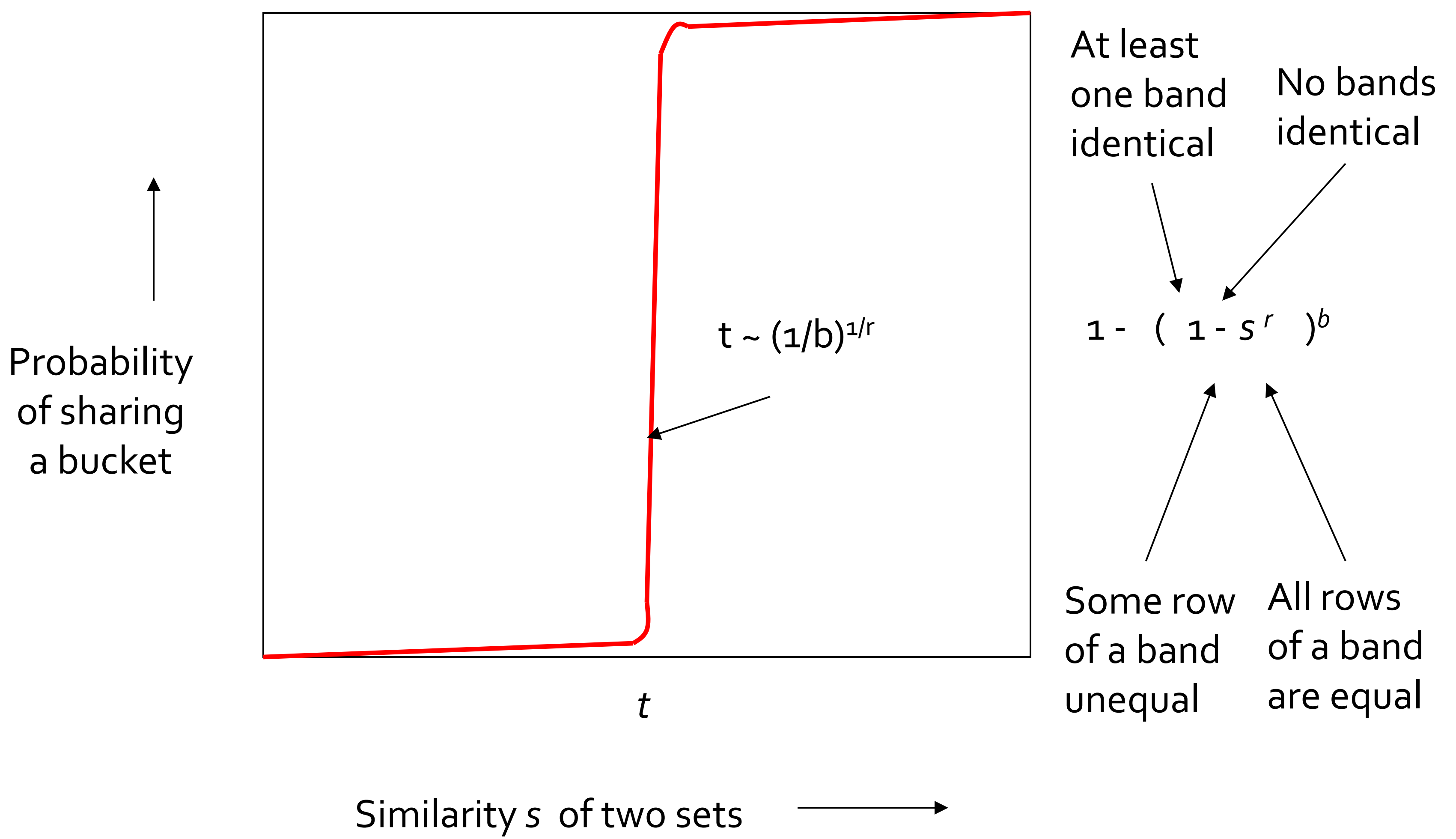
Analysis of LSH – What We Want



What One Band of One Row Gives You



What b Bands of r Rows Gives You



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 *r* and *c* 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* .