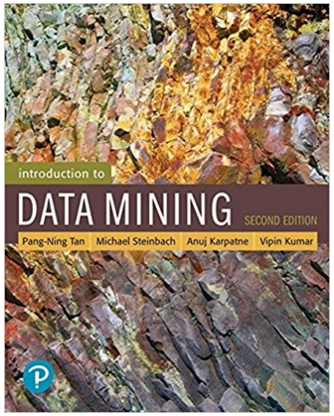


CIS 530—Advanced Data Mining

8- Min Hashing and Locality Sensitive Hashing

Thomas W Gyeera, Assistant Professor
Computer and Information Science
University of Massachusetts Dartmouth

Courtesy to Prof. Panayiotis Tsaparas

Why is similarity important?

- We saw many definitions of similarity and distance
- How do we make use of similarity in practice?
- What issues do we have to deal with?

An important problem

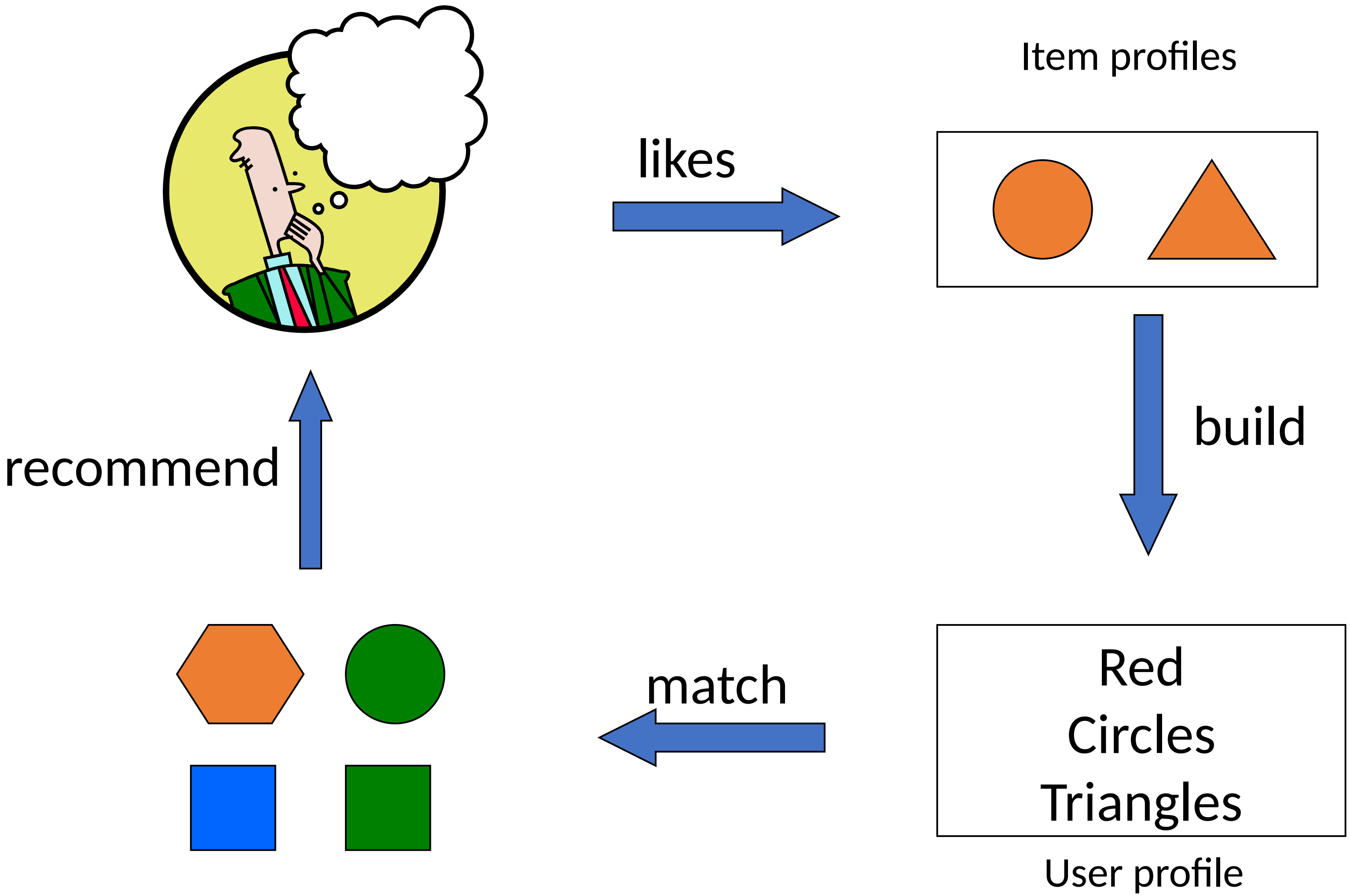
- Recommendation systems
  - When a user buys an item (initially books) we want to recommend other items that the user may like
  - When a user rates a movie, we want to recommend movies that the user may like
  - When a user likes a song, we want to recommend other songs that they may like

- A big success of data mining
- Exploits the long tail

Recommendation systems

- Content-based:
  - Represent the items into a feature space and recommend items to customer C similar to previous items rated highly by C
  - Movie recommendations: recommend movies with same actor(s), director, genre, …
  - Websites, blogs, news: recommend other sites with “similar” content

Plan of action



Limitations of content-based approach

- Finding the appropriate features
  - e.g., images, movies, music
- Overspecialization
  - Never recommends items outside user's content profile
  - People might have multiple interests
- Recommendations for new users
  - How to build a profile?

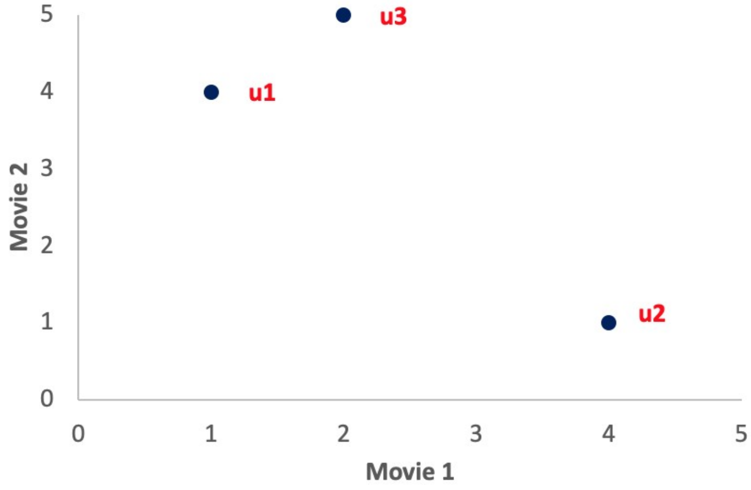
Recommendation Systems (II)

- Collaborative Filtering (user-user)
  - Consider user c
  - Find set D of other users whose ratings are "similar" to c's ratings
  - Estimate user's ratings based on ratings of users in D

Recommendation Systems (II)

- Collaborative Filtering (user-user)

	m1	m2
u1	1	4
u2	4	1
u3	2	5



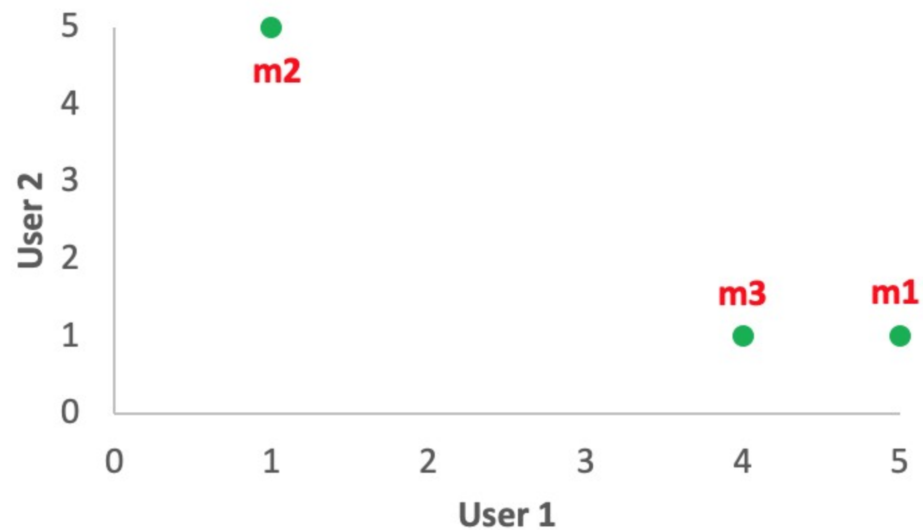
Recommendation Systems (III)

- Collaborative Filtering (item-item)
  - For item s, find other similar items
  - Estimate rating for item based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model
- In practice, it has been observed that item-item often works better than user-user

Recommendation Systems (III)

- Collaborative Filtering (item-item)

	u1	u2
m1	5	1
m2	1	5
m3	4	1



$$\cos(\theta) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{|\boldsymbol{A}||\boldsymbol{B}|}$$

	m1	m2	m3
m1	1.000000	0.384615	0.998868
m2	0.384615	1.000000	0.428086
m3	0.998868	0.428086	1.000000

Pros and cons of collaborative filtering

- Works for any kind of item
  - No feature selection needed
- New user problem
- New item problem
- Sparsity of rating matrix
  - Cluster-based smoothing?

Another important problem

- Find duplicate and near-duplicate documents from a web crawl.
- Why is it important:
  - Identify mirrored web pages, and avoid indexing them, or serving them multiple times
  - Find replicated news stories and cluster them under a single story.
  - Identify plagiarism
- What if we wanted exact duplicates?

Finding similar items

- Both the problems we described have a common component
  - We need a quick way to find highly similar items to a query item
  - OR, we need a method for finding all pairs of items that are highly similar.
- Also known as the Nearest Neighbor problem, or the All Nearest Neighbors problem
- We will examine it for the case of near-duplicate web documents.

Main issues

- What is the right representation of the document when we check for similarity?
  - E.g., representing a document as a set of characters will not do (why?)
- When we have billions of documents, keeping the full text in memory is not an option.
  - We need to find a shorter representation
- How do we do pairwise comparisons of billions of documents?
  - If exact match was the issue, it would be ok, can we replicate this idea?

Three Essential Techniques

1. Shingling: convert documents, emails, etc., to sets.

2. Minhashing: convert large sets to short signatures, while preserving similarity.

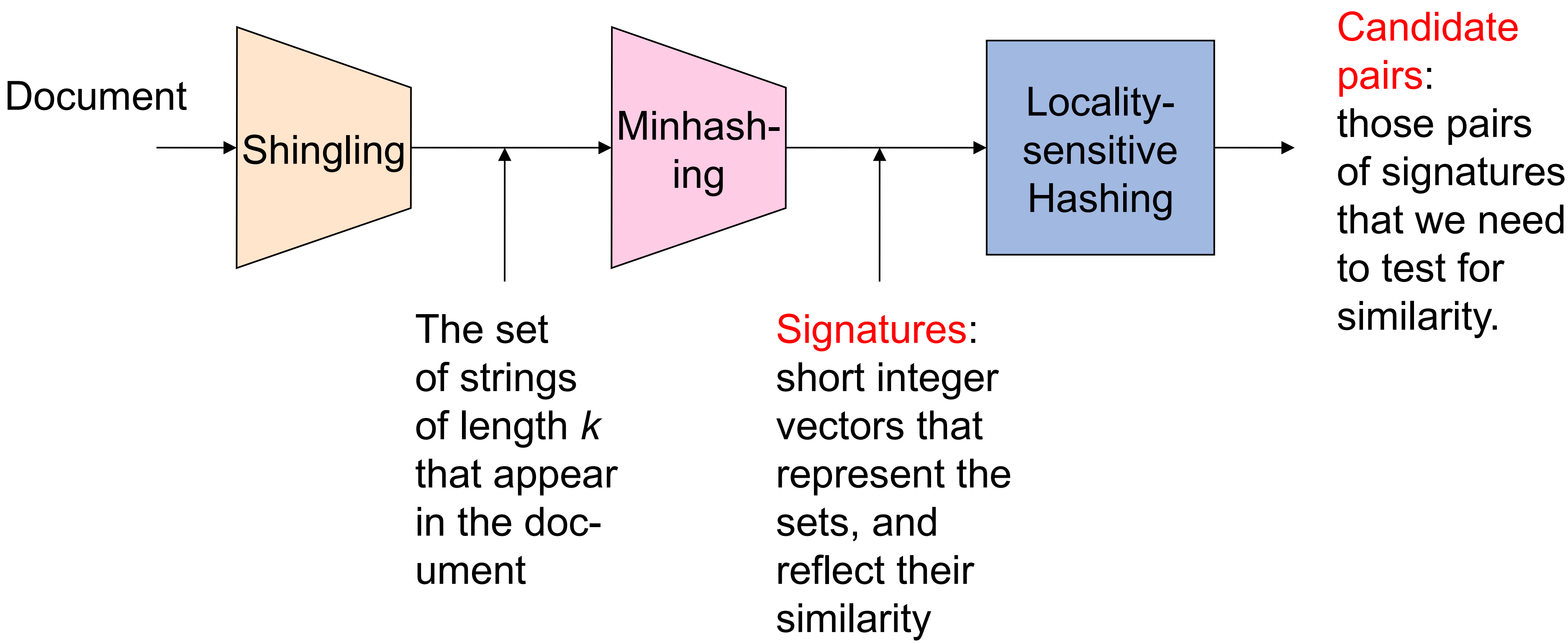
3. Locality-Sensitive Hashing (LSH): focus on pairs of signatures likely to be similar.

Motivating problem

- Find duplicate and near-duplicate documents from a web crawl.

- If we wanted exact duplicates, we could do this by hashing
  - We will see how to adapt this technique for near duplicate documents

The Big Picture



Shingles

- A k-shingle (or k-gram) for a document is a sequence of k characters that appears in the document.
- Example: document = abcab. k=2
  - Set of 2-shingles = {ab, bc, ca}.
  - Option: regard shingles as a bag, and count ab twice.

- Represent a document by its set of k-shingles.

Shingling

- Shingle: a sequence of k contiguous characters

```
a rose is a rose is a rose
a rose is 
 rose is a
  rose is a 
   ose is a r
    se is a ro
     e is a ros
      is a rose
       is a rose 
        s a rose i
         a rose is
          a rose is 
```

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.
  - Extreme case $k = 1$: all documents are the same
  - $k = 5$ is OK for short documents; $k = 10$ is better for long documents.
- Alternative ways to define shingles:
  - Use words instead of characters
  - Anchor on stop words (to avoid templates)

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.
- From now on we will assume that shingles are integers
  - Collisions are possible, but very rare

Fingerprinting

- Hash shingles to 64-bit integers

Set of Shingles	Hash function (Rabin's fingerprints)	Set of 64-bit integers
`a rose is `	→	1111
` rose is a`	→	2222
`rose is a `	→	3333
`ose is a r`	→	4444
`se is a ro`	→	5555
`e is a ros`	→	6666
` is a rose`	→	7777
`is a rose `	→	8888
`s a rose i`	→	9999
` a rose is`	→	0000

Basic Data Model: Sets

- Document: A document is represented as a set shingles (more accurately, hashes of shingles)

- Document similarity: Jaccard similarity of the sets of shingles.
  - Common shingles over the union of shingles
  - $Sim\ (C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$.

- Although we use the documents as our driving example the techniques we will describe apply to any kind of sets.
  - E.g., similar customers or items.

Signatures

- Problem: shingle sets are too large to be kept in memory.

- Key idea: "hash" each set S to a small signature Sig (S), such that:

  1. Sig (S) is small enough that we can fit a signature in main memory for each set.

  2. Sim (S_1, S_2) is (almost) the same as the "similarity" of Sig (S_1) and Sig (S_2). (signature preserves similarity).

- Warning: This method can produce false negatives, and false positives (if an additional check is not made).
  - False negatives: Similar items deemed as non-similar
  - False positives: Non-similar items deemed as similar

From Sets to Boolean Matrices

- Represent the data as a boolean matrix M
  - Rows = the universe of all possible set elements
    - In our case, shingle fingerprints take values in $[0 \ldots 2^{64}-1]$
  - Columns = the sets
    - In our case, documents, sets of shingle fingerprints
  - $M(r,S) = 1$ in row r and column S, if and only if r is a member of S.

- Typical matrix is sparse.
  - We do not really materialize the matrix

Example

- Universe: **U = {A,B,C,D,E,F,G}**

- X = {A,B,F,G}
- Y = {A,E,F,G}

- Sim(X,Y) =

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

Example

- Universe: **U = {A,B,C,D,E,F,G}**

- X = {A,B,F,G}
- Y = {A,E,F,G}

- Sim(X,Y) =

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

At least one of the columns has value 1

Example

- Universe: **U = {A,B,C,D,E,F,G}**

- X = {A,B,F,G}
- Y = {A,E,F,G}

- Sim(X,Y) =

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

Both columns have value 1

Minhashing

- Pick a random permutation of the rows (the universe U).
- Define "hash" function for set S
  - h(S) = the index of the first row (in the permuted order) in which column S has 1.
  - OR
  - h(S) = the index of the first element of S in the permuted order.
- Use k (e.g., k = 100) independent random permutations to create a signature.

Example of minhash signatures

- Input matrix

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

A
C
G
F
B
E
D

		S_1	S_2	S_3	S_4
1	**A**	1	0	1	0
2	**C**	0	1	0	1
3	**G**	1	0	1	0
4	**F**	1	0	1	0
5	**B**	1	0	0	1
6	**E**	0	1	0	1
7	**D**	0	1	0	1
		1	2	1	2

Example of minhash signatures

- Input matrix

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

D
B
A
C
F
G
E

		S_1	S_2	S_3	S_4
1	**D**	0	1	0	1
2	**B**	1	0	0	1
3	**A**	1	0	1	0
4	**C**	0	1	0	1
5	**F**	1	0	1	0
6	**G**	1	0	1	0
7	**E**	0	1	0	1
		2	1	3	1

Example of minhash signatures

- Input matrix

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

C
D
G
F
A
B
E

		S_1	S_2	S_3	S_4
1	**C**	0	1	0	1
2	**D**	0	1	0	1
3	**G**	1	0	1	0
4	**F**	1	0	1	0
5	**A**	1	0	1	0
6	**B**	1	0	0	1
7	**E**	0	1	0	1

3	1	3	1

Example of minhash signatures

- Input matrix

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

$\approx$

Signature matrix

	S_1	S_2	S_3	S_4
h_1	1	2	1	2
h_2	2	1	3	1
h_3	3	1	3	1

- Sig(S) = vector of hash values
  - e.g., Sig(S_2) = [2,1,1]
- Sig(S,i) = value of the i-th hash function for set S
  - E.g., Sig(S_2,3) = 1

Hash function Property

$$Pr(h(S_1) = h(S_2)) = Sim(S_1, S_2)$$

- where the probability is over all choices of permutations.

- Why?
  - The first row where one of the two sets has value 1 belongs to the union.
    - Recall that union contains rows with at least one 1.
  - We have equality if both sets have value 1, and this row belongs to the intersection

Example

- Universe: **U = {A,B,C,D,E,F,G}**
- X = {A,B,F,G}
- Y = {A,E,F,G}

Rows C,D could be anywhere they do not affect the probability

- Union = {A,B,E,F,G}
- Intersection = {A,F,G}

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

D
*
*
C
*
*
*

	X	Y
D	0	0
C	0	0

Example

- Universe: **U = {A,B,C,D,E,F,G}**
- X = {A,B,F,G}
- Y = {A,E,F,G}

The * rows belong to the union

- Union =
  {A,B,E,F,G}
- Intersection =
  {A,F,G}

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

D
*
*
C
*
*
*

	X	Y
D	0	0
C	0	0

Example

- Universe: **U = {A,B,C,D,E,F,G}**
- X = {A,B,F,G}
- Y = {A,E,F,G}

The question is what is the value of the **first** * element

- Union = {A,B,E,F,G}
- Intersection = {A,F,G}

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

D
*
*
C
*
*
*

	X	Y
D	0	0
C	0	0

Example

- Universe: **U = {A,B,C,D,E,F,G}**
- X = {A,B,F,G}
- Y = {A,E,F,G}

If it belongs to the intersection, then
h(X) = h(Y)

- Union = {A,B,E,F,G}
- Intersection = {A,F,G}

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

D
*
*
C
*
*
*

	X	Y
D	0	0
C	0	0

Example

- Universise: U = {A,B,C,D,E,F,G}
- X = {A,B,F,G}
- Y = {A,E,F,G}

Every element of the union is equally likely to be the * element

Pr(h(X) = h(Y)) = Sim(X,Y)

- Union = {A,B,E,F,G}
- Intersection = {A,F,G}

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

D
*
*
C
*
*
*

	X	Y
D	0	0
C	0	0

Similarity for Signatures

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

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

$\approx$

Signature matrix

S_1	S_2	S_3	S_4
1	2	1	2
2	1	3	1
3	1	3	1

	Actual	Sig
(S_1, S_2)	0	0
(S_1, S_3)	3/5	2/3
(S_1, S_4)	1/7	0
(S_2, S_3)	0	0
(S_2, S_4)	3/4	1
(S_3, S_4)	0	0

- tures we get a good appr

Zero similarity is preserved

High similarity is well approximated

Is it now feasible?

- Assume a billion rows
- Hard to pick a random permutation of 1 billion…
- **Even representing a random permutation requires 1 billion entries!!!**
- How about accessing rows in permuted order?

Being more practical

- Instead of permuting the rows we will apply a hash function that maps the rows to a new (possibly larger) space
  - The value of the hash function is the position of the row in the new order (permutation).
  - Each set is represented by the smallest hash value among the elements in the set

- The space of the hash functions should be such that if we select one at random each element (row) has equal probability to have the smallest value
  - Min-wise independent hash functions

Algorithm – One set, one hash function

Computing **Sig(S,i)** for a single column S and single hash function h_i

In practice only the rows (shingles) that appear in the data

for each row **r**

compute $\mathbf{h_i (r)}$

h_i (**r**) = index of row r in permutation

if column **S** that has **1** in row **r**

S contains row r

if $\mathbf{h_i (r)}$ is a smaller value than **Sig(S,i)** **then**

Find the row r with minimum index

$\mathbf{Sig(S,i) = h_i (r);}$

Sig(S,i) will become the smallest value of $\mathbf{h_i(r)}$ among all rows (shingles) for which column **S** has value **1** (shingle belongs in S); *i*.e., $\mathbf{h_i (r)}$ gives the min index for the **i**-th permutation

Algorithm – All sets, k hash functions

Pick $k=100$ hash functions $(h_1,\ldots,h_k)$

In practice this means selecting the hash function parameters

for each row r

    for each hash function h_i

        compute $h_i\,(r)$

Compute $h_i\,(r)$ only once for all sets

        for each column S that has 1 in row r

            if $h_i\,(r)$ is a smaller value than $Sig(S,i)$ **then**

                $Sig(S,i) = h_i\,(r);$

Algorithm – All sets, k hash functions

x	Row	S1	S2	h(x)	g(x)
0	A	1	0	1	3
1	B	0	1	2	0
2	C	1	1	3	2
3	D	1	0	4	4
4	E	0	1	0	1

$h(x)$ = x+1 mod 5
$g(x)$ = 2x+3 mod 5

h(Row)	Row	S1	S2
0	E	0	1
1	A	1	0
2	B	0	1
3	C	1	1
4	D	1	0

g(Row)	Row	S1	S2
0	B	0	1
1	E	0	1
2	C	1	0
3	A	1	1
4	D	1	0

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

Implementation

- 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.

Finding similar pairs

- Problem: Find all pairs of documents with similarity at least t = 0.8
- 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.

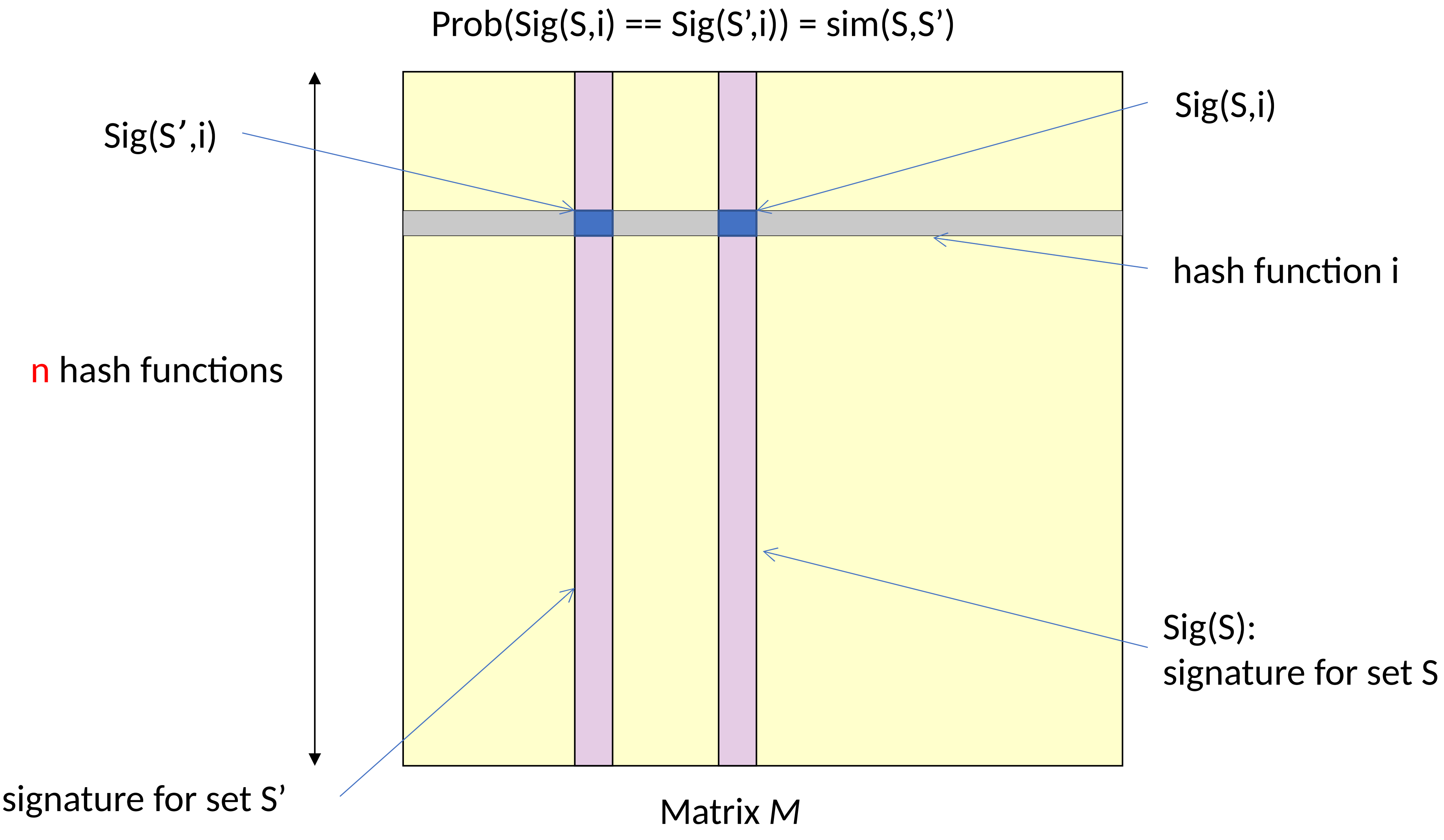
Locality-Sensitive Hashing

- What we want: 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.
- A simple idea: X and Y are a candidate pair if they have the same min-hash signature.
  - Easy to test by hashing the signatures.
  - Similar sets are more likely to have the same signature.
  - Likely to produce many false negatives.
    - Requiring full match of signature is strict, some similar sets will be lost.

! Multiple levels of Hashing!

- Improvement: Compute multiple signatures; candidate pairs should have at least one common signature.
  - Reduce the probability for false negatives.

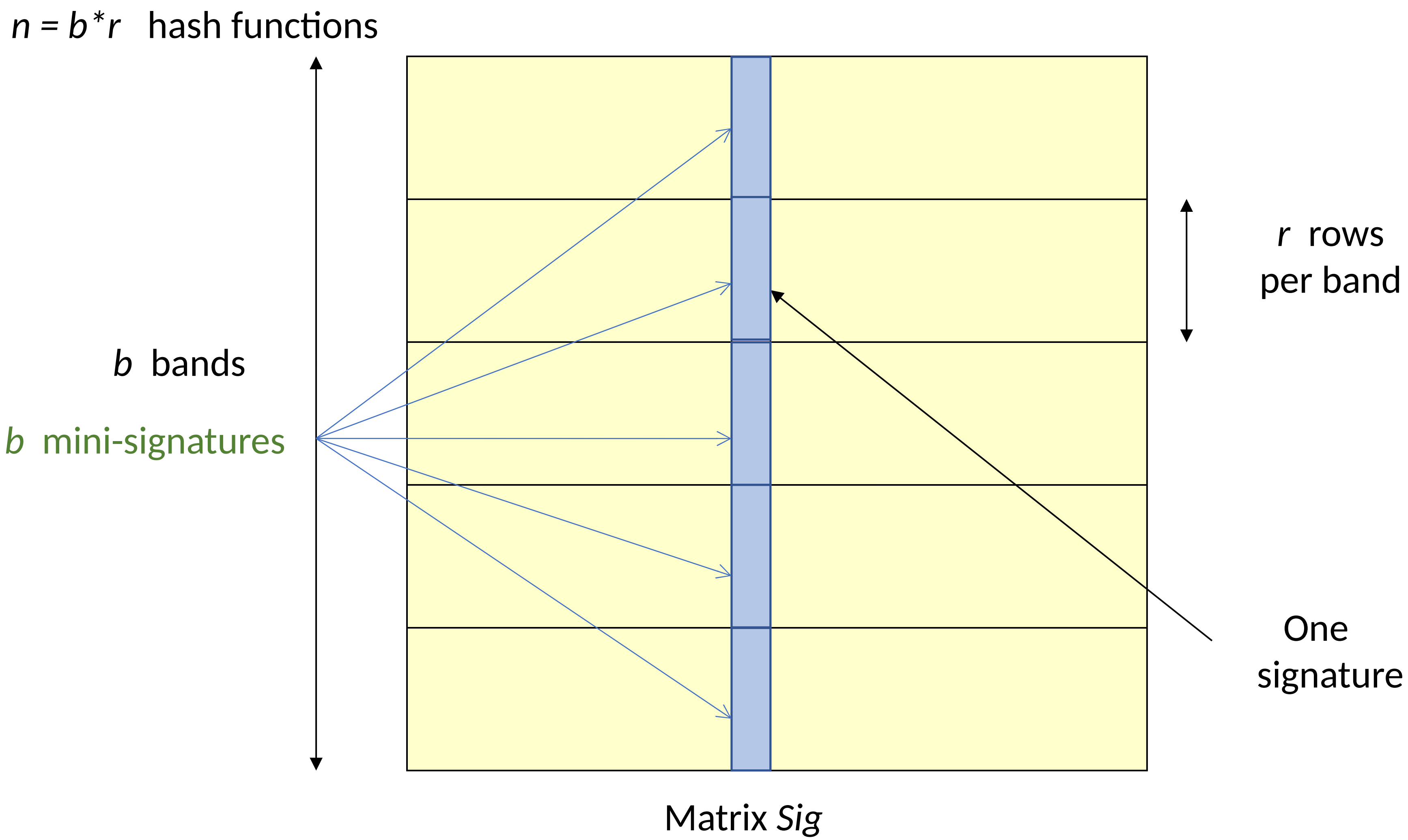
Signature matrix reminder



Partition into Bands – (1)

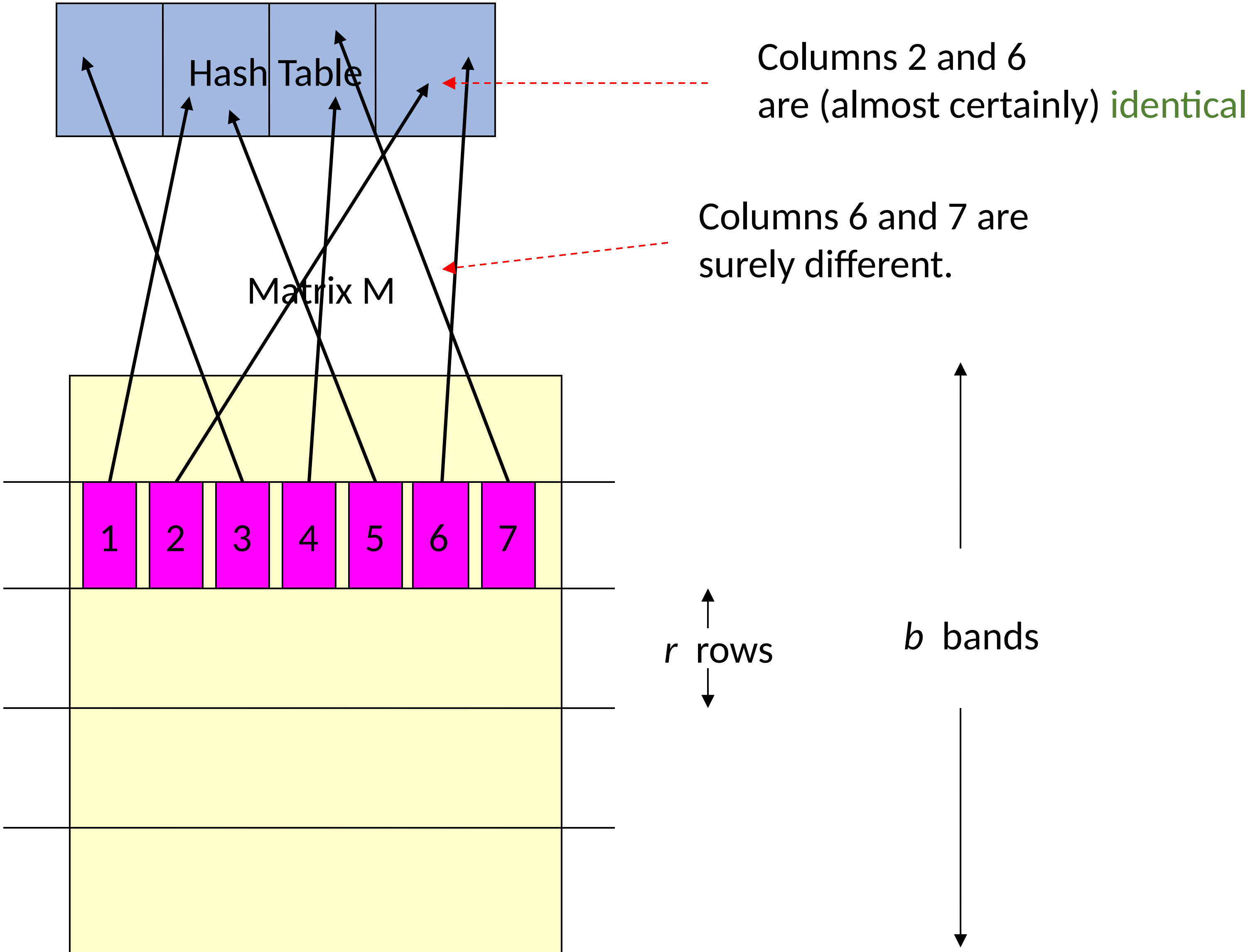
- Divide the signature matrix Sig into b bands of r rows.
  - Each band is a mini-signature with r hash functions.

Partition into Bands – (1)



Partition into Bands – (2)

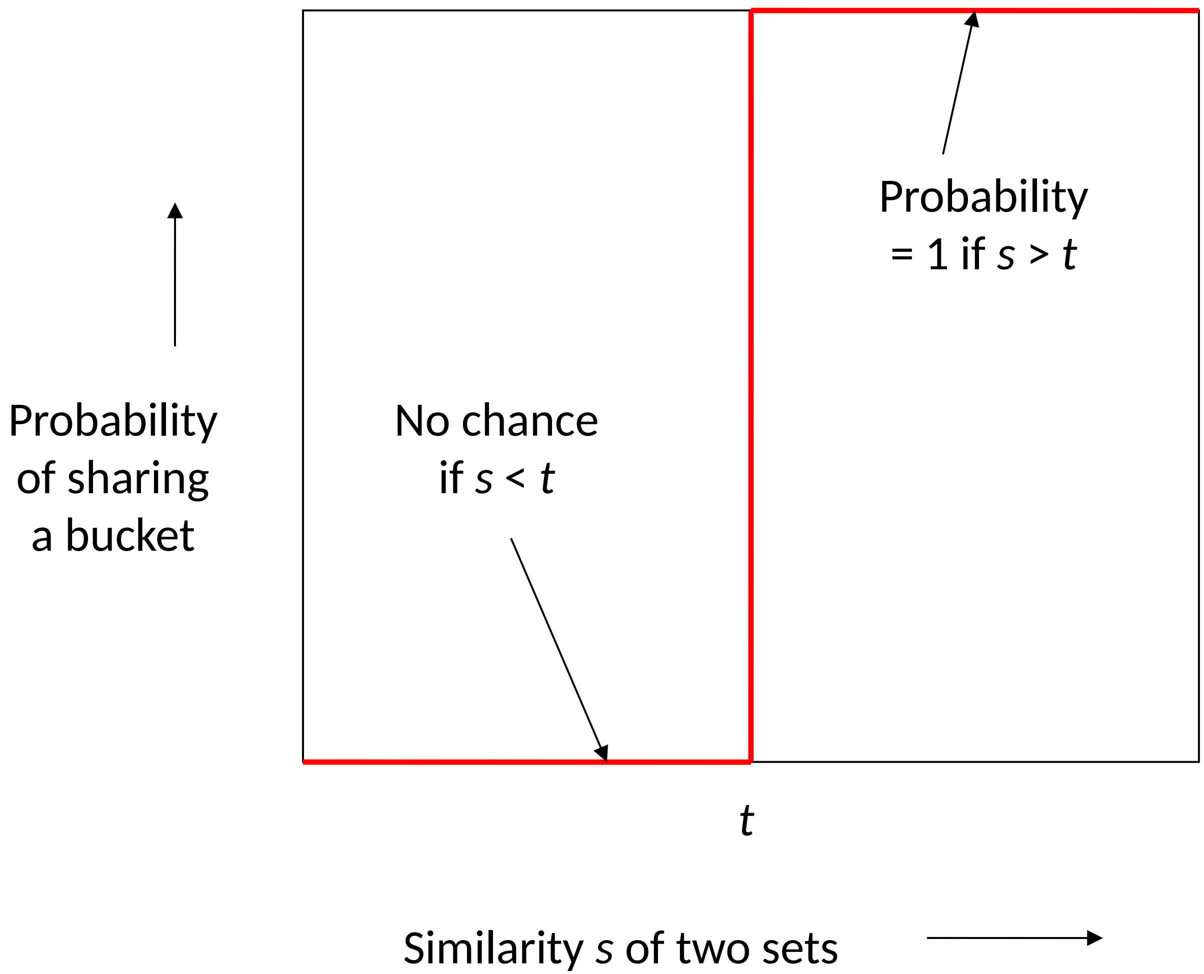
- Divide the signature matrix Sig into *b* bands of *r* rows.
  - Each band is a mini-signature with r hash functions.
- For each band, hash the mini-signature to a hash table with *k* buckets.
  - Make *k* as large as possible so that mini-signatures that hash to the same bucket are almost certainly identical.


Hash Table
Columns 2 and 6
are (almost certainly) identical.
Columns 6 and 7 are
surely different.
Matrix M
1
2
3
4
5
6
7
r rows
b bands

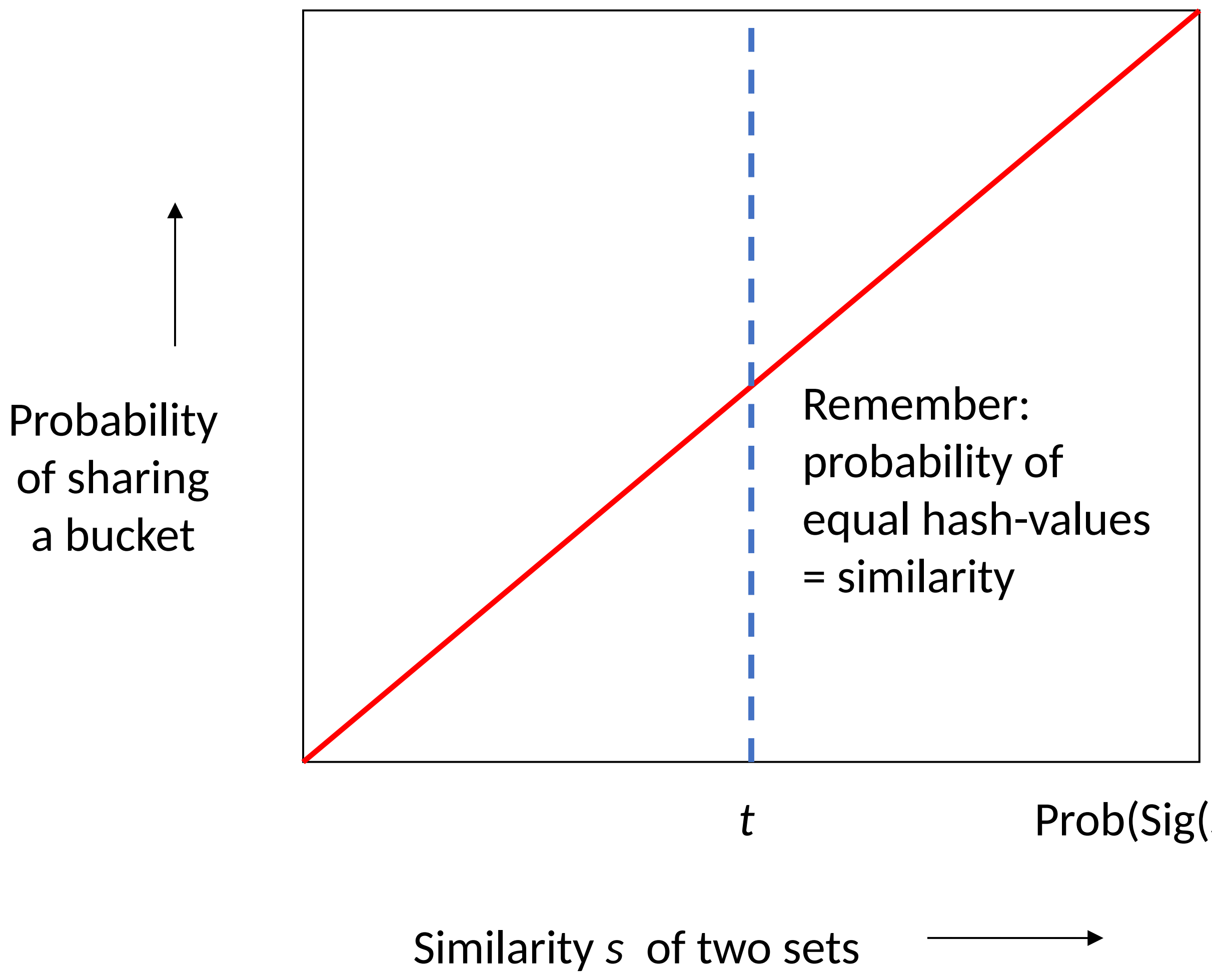
Partition into Bands – (3)

- Divide the signature matrix Sig into *b* bands of *r* rows.
  - Each band is a mini-signature with r hash functions.
- For each band, hash the mini-signature to a hash table with *k* buckets.
  - Make *k* as large as possible so that mini-signatures that hash to the same bucket are almost certainly identical.
- Candidate column pairs are those that hash to the same bucket for at least 1 band.
- Tune *b* and *r* to catch most similar pairs, but few non-similar pairs.

Analysis of LSH – What we want



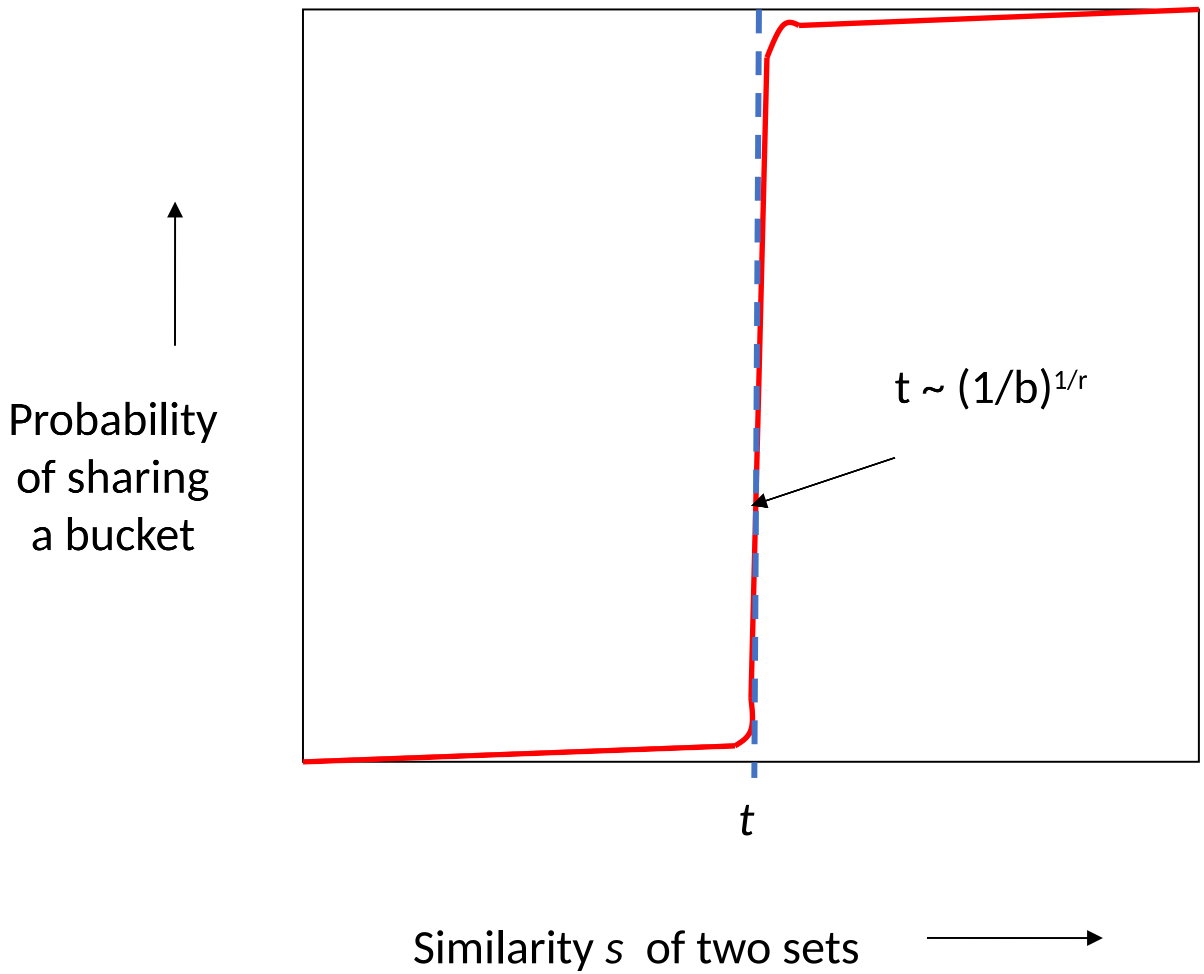
What One Band of One Row Gives You



Single hash signature

Prob(Sig(S,i) == Sig(S',i)) = sim(S,S')

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



At least one band identical

No bands identical

$$1-\left(1-s^{r}\right)^{b}$$

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

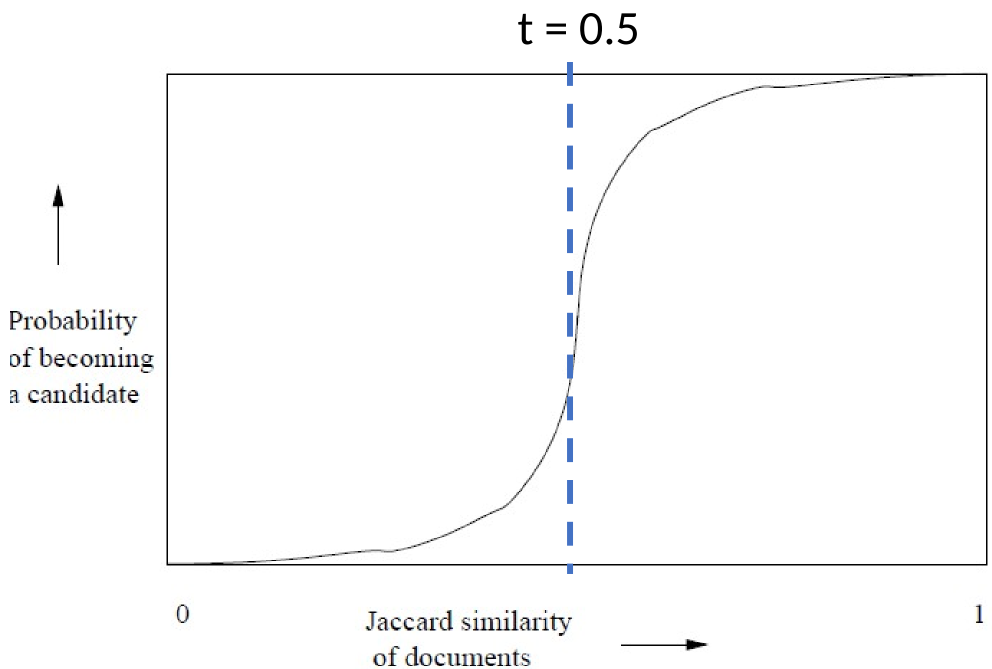


Figure 3.7: The S-curve

Suppose S_1, S_2 are 80% Similar

- We want all 80%-similar pairs. Choose 20 bands of 5 integers/band.

- Probability S_1, S_2 identical in one particular band:

$$(0.8)^5 = 0.328.$$

- Probability S_1, S_2 are not similar in any of the 20 bands:

$$(1-0.328)^{20} = 0.00035$$

i.e., about 1/3000-th of the 80%-similar column pairs are false negatives.

- Probability S_1, S_2 are similar in at least one of the 20 bands:

$$1-0.00035 = 0.999$$

Suppose S_1, S_2 Only 40% Similar

- Probability S_1, S_2 identical in any one particular band:

  $(0.4)^5 = 0.01$.

- Probability S_1, S_2 identical in at least 1 of 20 bands:

  $\leq 20 * 0.01 = 0.2$.

- But false positives much lower for similarities << 40%.