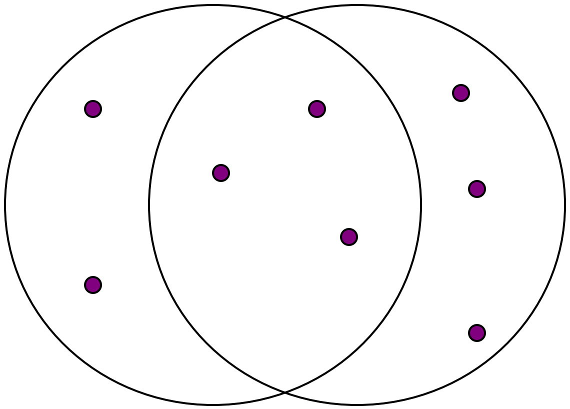
DATA MINING
LECTURE 5

Sketching, Locality Sensitive Hashing

Jaccard Similarity

- The Jaccard similarity (Jaccard coefficient) of two sets S_1, S_2 is the size of their intersection divided by the size of their union.
  - JSim $(S_1, S_2) = |S_1 \cap S_2| \,/\, |S_1 \cup S_2|$.



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

- Extreme behavior:
  - Jsim(X,Y) = 1, iff X = Y
  - Jsim(X,Y) = 0 iff X,Y have no elements in common
- JSim is symmetric

Cosine Similarity

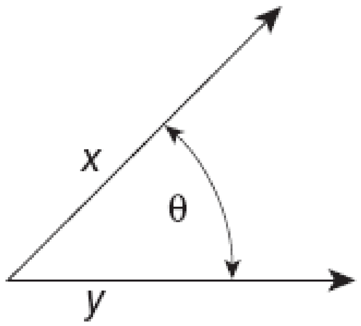


Figure 2.16. Geometric illustration of the cosine measure.

- Sim(X,Y) = cos(X,Y)
  - The cosine of the angle between X and Y

- If the vectors are aligned (correlated) angle is zero degrees and cos(X,Y)=1
- If the vectors are orthogonal (no common coordinates) angle is 90 degrees and cos(X,Y) = 0

- Cosine is commonly used for comparing documents, where we assume that the vectors are normalized by the document length.

Application: Recommendations

- Recommendation systems
  - When a user buys or rates an item we want to recommend other items that the user may like
    - Initially applied to books, but now recommendations are everywhere: songs, movies, products, restaurants, hotels, etc.

- Commonly used algorithms:
  - Find the k users most similar to the user at hand and recommend items that they like.
  - Find the items most similar to the items that the user has previously liked, and recommend these items.

Application: Finding near duplicates

- 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
- Near duplicate documents differ in a few characters, words or sentences

Finding similar items

- The problems we have seen so far 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

SKETCHING AND LOCALITY SENSITIVE HASHING

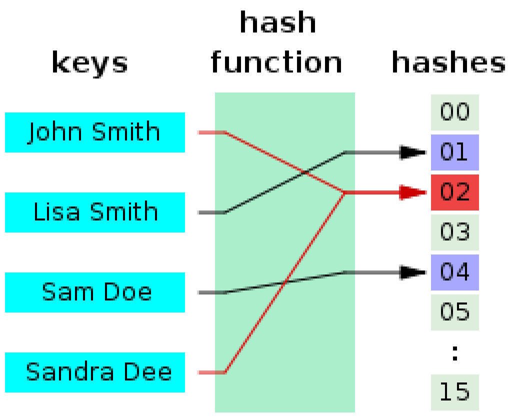
Thanks to:

Rajaraman and Ullman, “Mining Massive Datasets”

Evimaria Terzi, slides for Data Mining Course.

Before we start: Hash Functions

- A hash function is a function that maps objects of arbitrary sizes (e.g., strings) to a space of fixed size (usually, integers).
  - Simple example: $h(x) = (ax + b) \; mod \; n$
- If two values are mapped to the same integer we say that we have a collision
- Hash functions are usually randomized
  - E.g., values a, b are selected at random
  - They are designed so that the probability of collision is very small.
  - Perfect hash functions: map each valid input to a different hash value.
- Hash functions are used in Hash Tables to implement Dictionaries



Problem

- Given a (large) collection of documents find all pairs of documents which are near duplicates
  - Their similarity is very high

- What if we want to find identical 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 we wanted exact match it would be ok, can we replicate this idea?

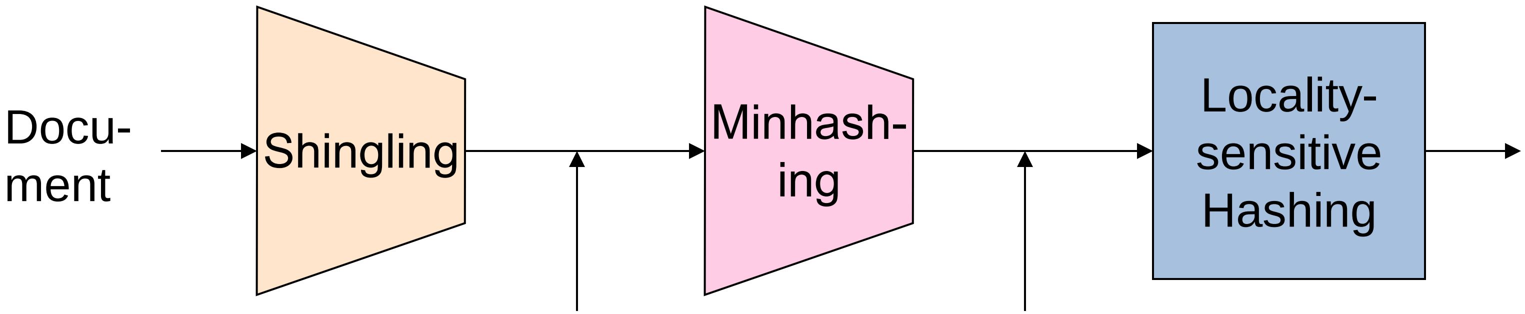
Three Essential Techniques for Similar Documents

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

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

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

The Big Picture



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

Candidate pairs : those pairs of signatures that we need to test for similarity.

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 
```

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 
```

```
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
```

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.

$$h: V^k \to \{0,1\}^{64}$$

- Represent a doc by the set of hash values of its *k*-shingles.
  - Shingle s will be represented by the 64-bit integer $h(s)$
- 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 still 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 $[0\ldots2^{64}-1]$
  - 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

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

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

- Sim(X,Y) = $\frac{3}{5}$

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

Example

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

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

- Sim(X,Y) = $\frac{3}{5}$

At least one of the columns has value 1

	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) = $\frac{3}{5}$

Both columns have value 1

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

  same as:
  - 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

element	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	1	1
F	1	0	1	0
G	1	0	1	0

Random Permutation

A
C
G
F
B
E
D

index	element	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	1	1
7	**D**	0	1	0	1
		1	**2**	**1**	**2**

Example of minhash signatures

- Input matrix

element	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	1	1
F	1	0	1	0
G	1	0	1	0

Random Permutation

D
B
A
C
F
G
E

index	element	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	1	1
		2	**1**	**3**	**1**

Example of minhash signatures

- Input matrix

element	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	1	1
F	1	0	1	0
G	1	0	1	0

Random Permutation

C
D
G
F
A
B
E

index	element	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	1	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	1	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

We now have a smaller dataset with just k rows

- 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., $Sig(S_2,3) = 1$

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.

Hash function Property

$$\Pr(h(S_1) = h(S_2)) = \mathrm{Sim}(S_1, S_2)$$

- where the probability is over all choices of permutations.
- Why?
  - Recall that the union $S_1 \cup S_2$ contains the rows with at least one 1.
    - These are the rows that we care about
    - The first row in the permutation where one of the two sets has value 1 belongs to the union.
  - 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

- Universire: **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

- Universire: **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

- Universe: **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

$$\Pr(h(X) = h(Y)) = \frac{|\{A,F,G\}|}{|\{A,B,E,F,G\}|} = \frac{3}{5} = \mathrm{Sim}(X,Y)$$

- 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	1	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

Zero similarity is preserved
High similarity is well approximated

- With multiple signatures we get a good approximation
  - Why? What is the expected value of the fraction of agreements?

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?
- ☹

- Instead of permutations we will consider hash functions that map the N rows to N buckets
  - Some collisions may happen, but with well chosen functions they are rare.

Approximating row permutations

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

for each set **S**

for each row **r** that appears in **S**

for each hash function h_i

compute $h_i\,(r)$

for each hash function h_i

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

In practice this means selecting the function parameters

$h_i\,(r)$ = index of shingle r in permutation

Find the minimum index for hash function h_i

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

Approximating row permutations

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

for each row **r**

for each hash function **h_i**

compute **h_i (r)**

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);

In practice this means selecting the function parameters

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

h_i (r) = index of shingle r in permutation

S contains shingle r

Find the shingle r with minimum index

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

Example

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

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

Row	S1	S2
E	0	1
A	1	0
B	0	1
C	1	1
D	1	0

Row	S1	S2
B	0	1
E	0	1
C	1	0
A	1	1
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 – (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.

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.

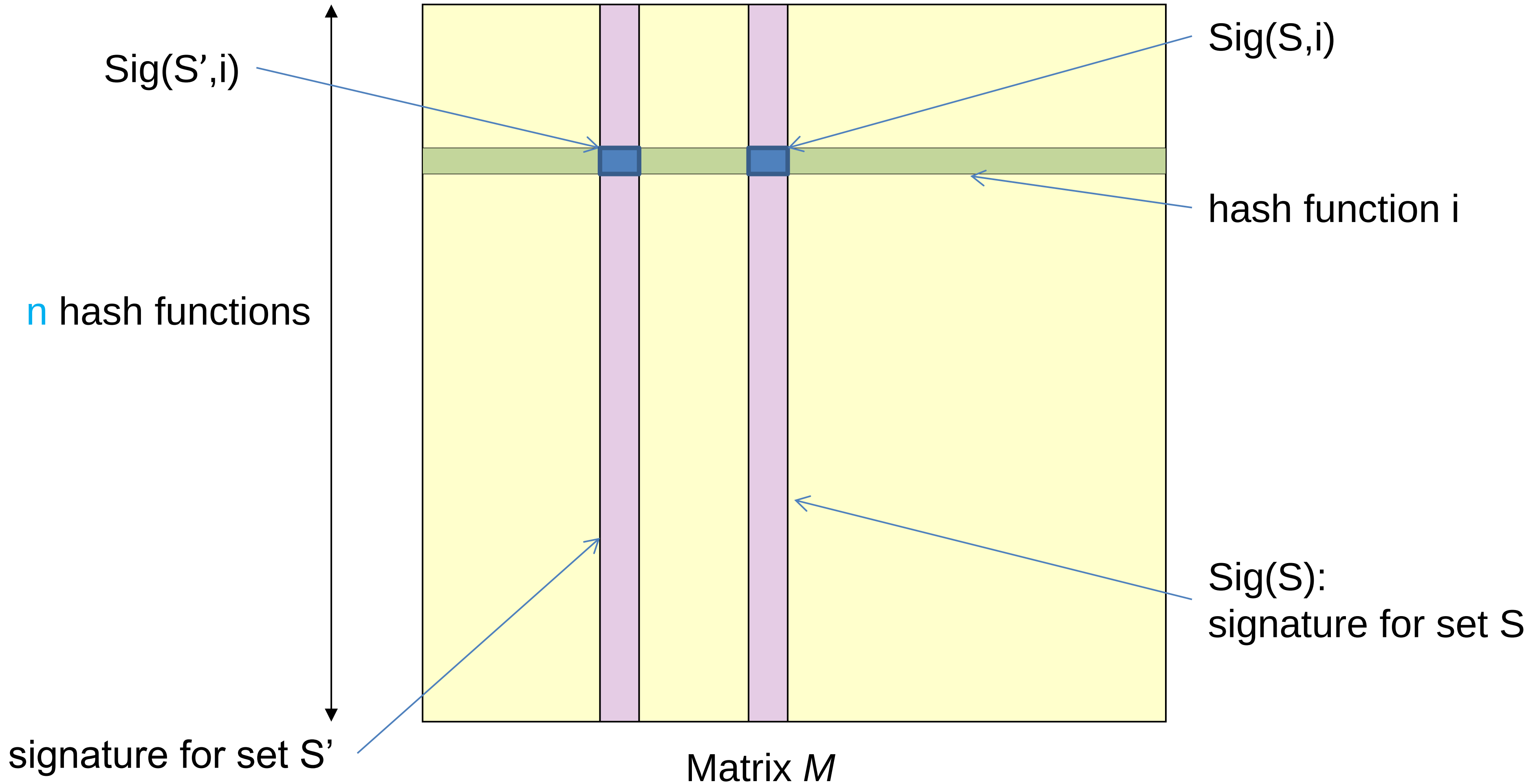
- A simple idea: X and Y are a candidate pair if they have the same min-hash signature.
  - Easy to test by hashing the signatures.
  - Similar sets are more likely to have the same signature.
  - Likely to produce many false negatives.
    - Requiring full match of signature is strict, some similar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

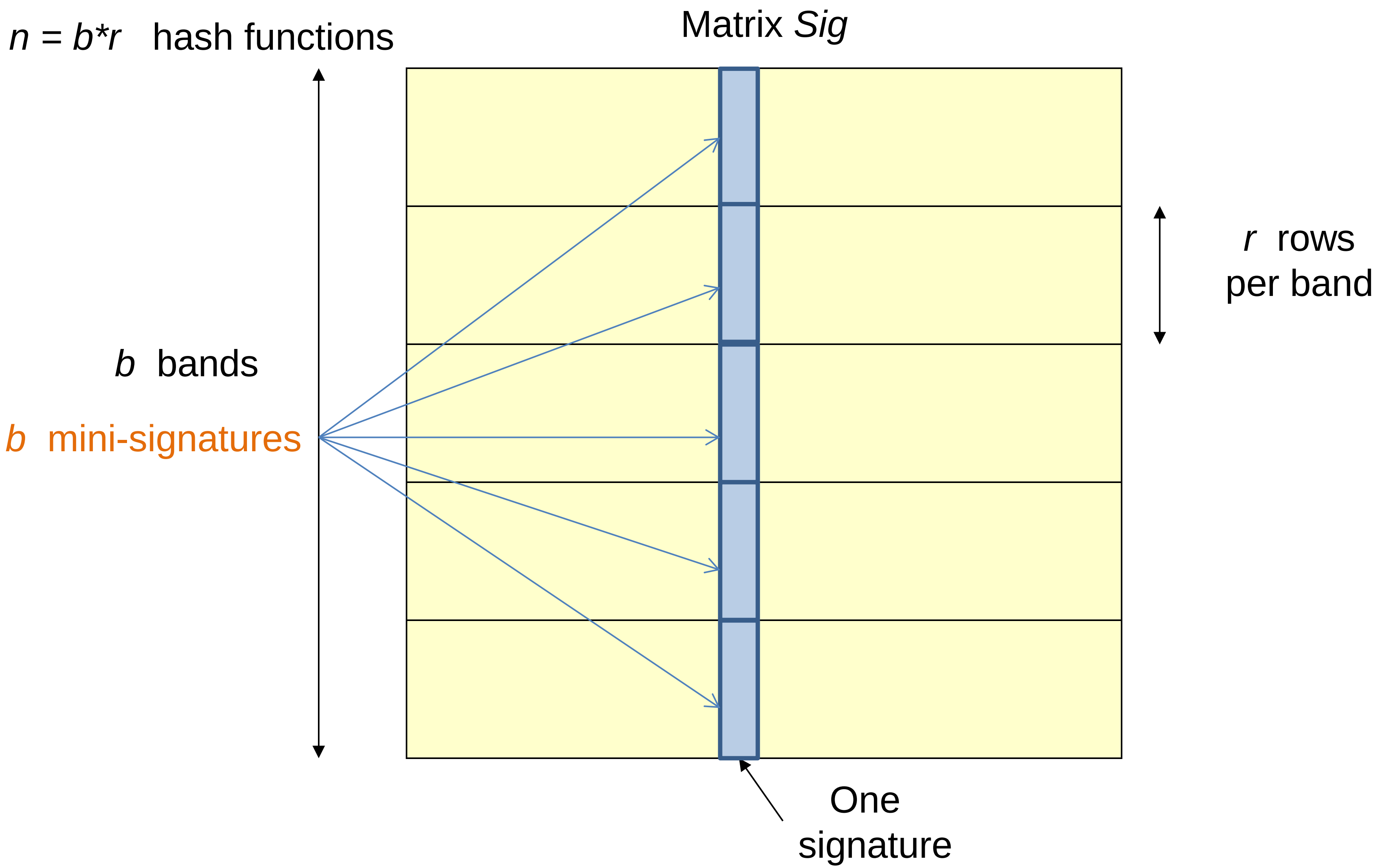
Prob(Sig(S,i) == Sig(S',i)) = sim(S,S')



Partition into Bands – (1)

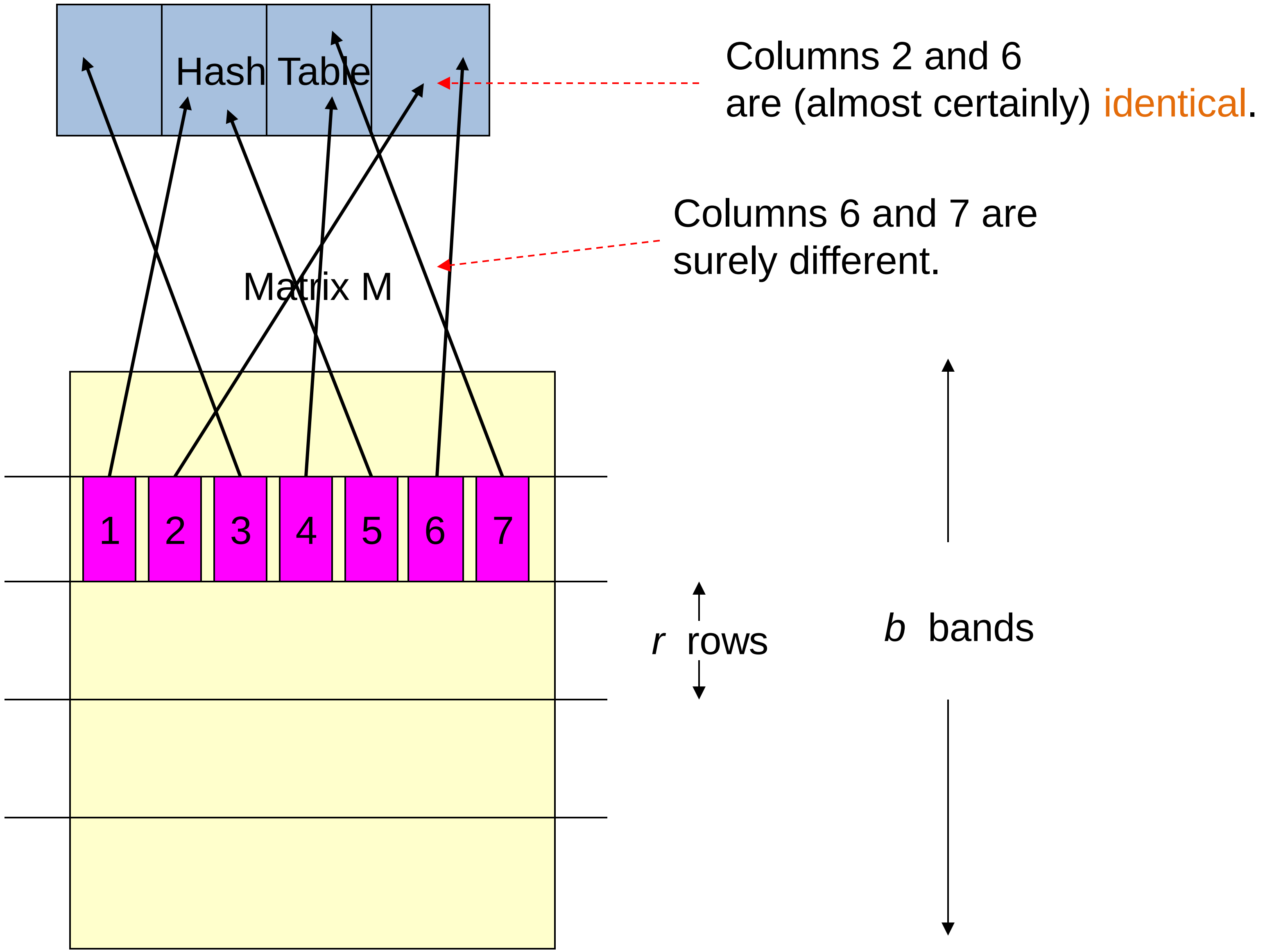
- Divide the signature matrix Sig into *b* bands of *r* rows.
  - Each band is a mini-signature with r hash functions.

Partitioning into bands



Partition into Bands – (2)

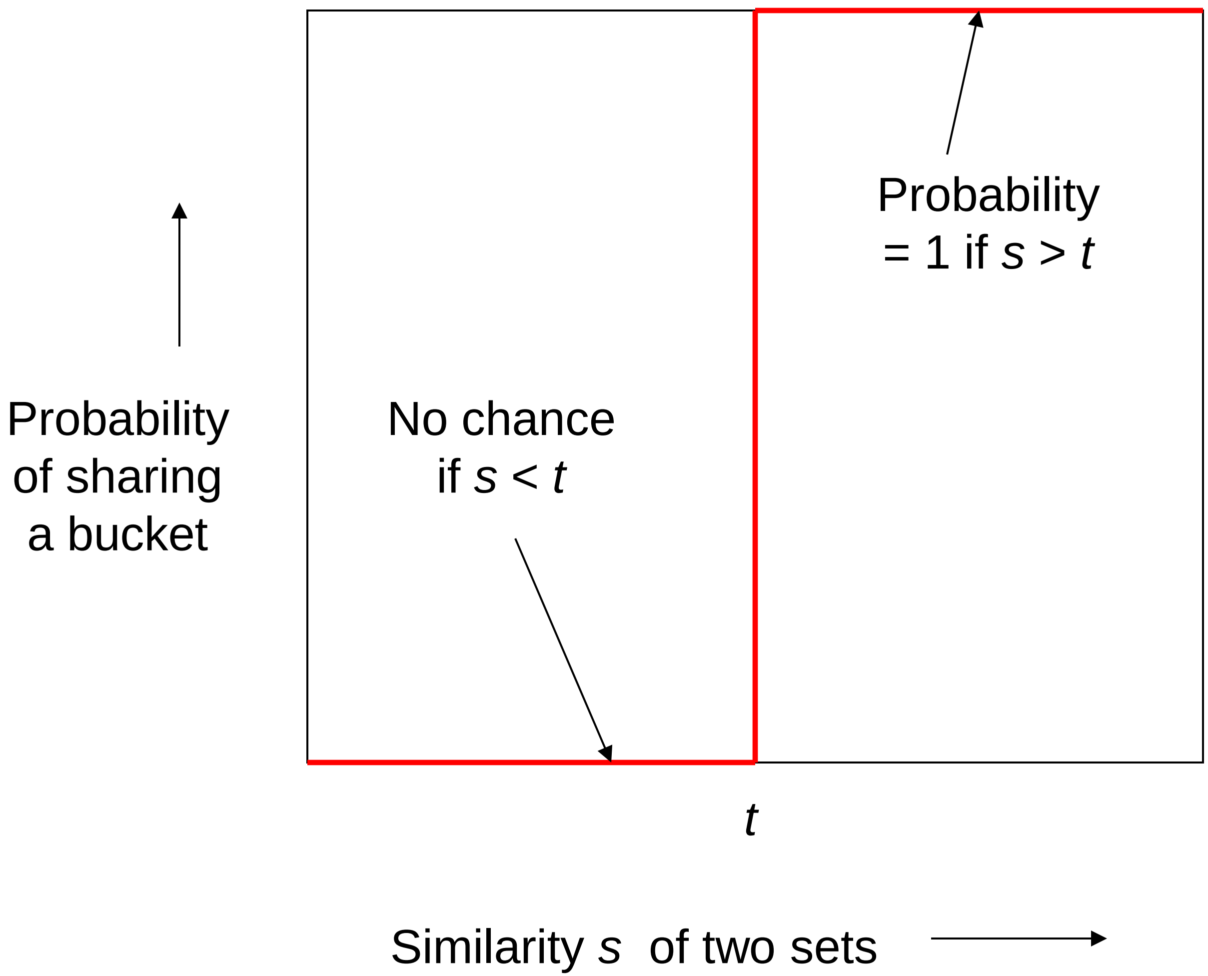
- Divide the signature matrix Sig into *b* bands of *r* rows.
  - Each band is a mini-signature with r hash functions.
- For each band, hash the mini-signature to a hash table.
  - Mini-signatures that hash to the same bucket are almost certainly identical.


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

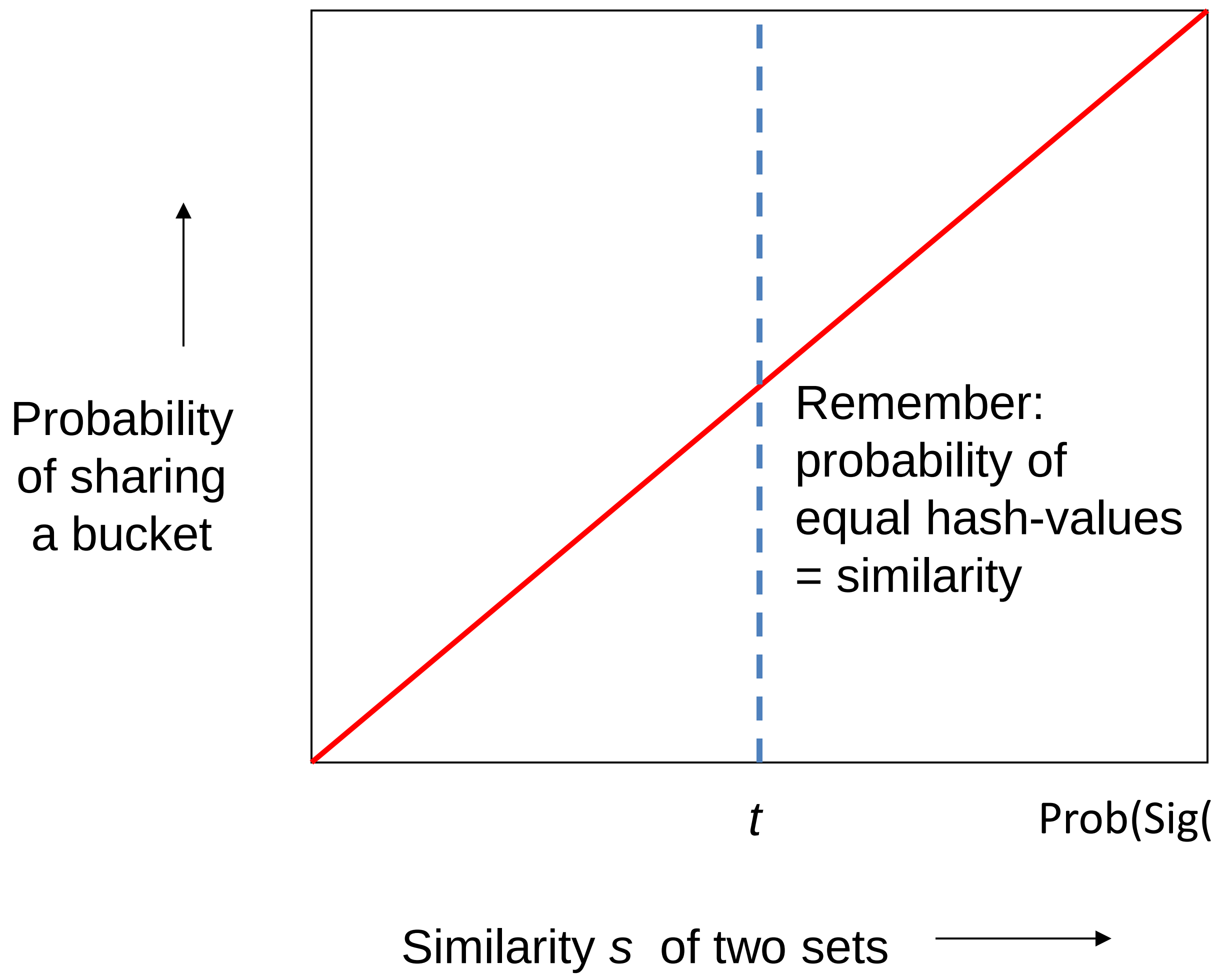
Partition into Bands – (2)

- 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.
  - 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.
  - I.e., they have at least one mini-signature in common.
- Tune *b* and *r* to catch most similar pairs, but few non-similar pairs.

Analysis of LSH – What We Want



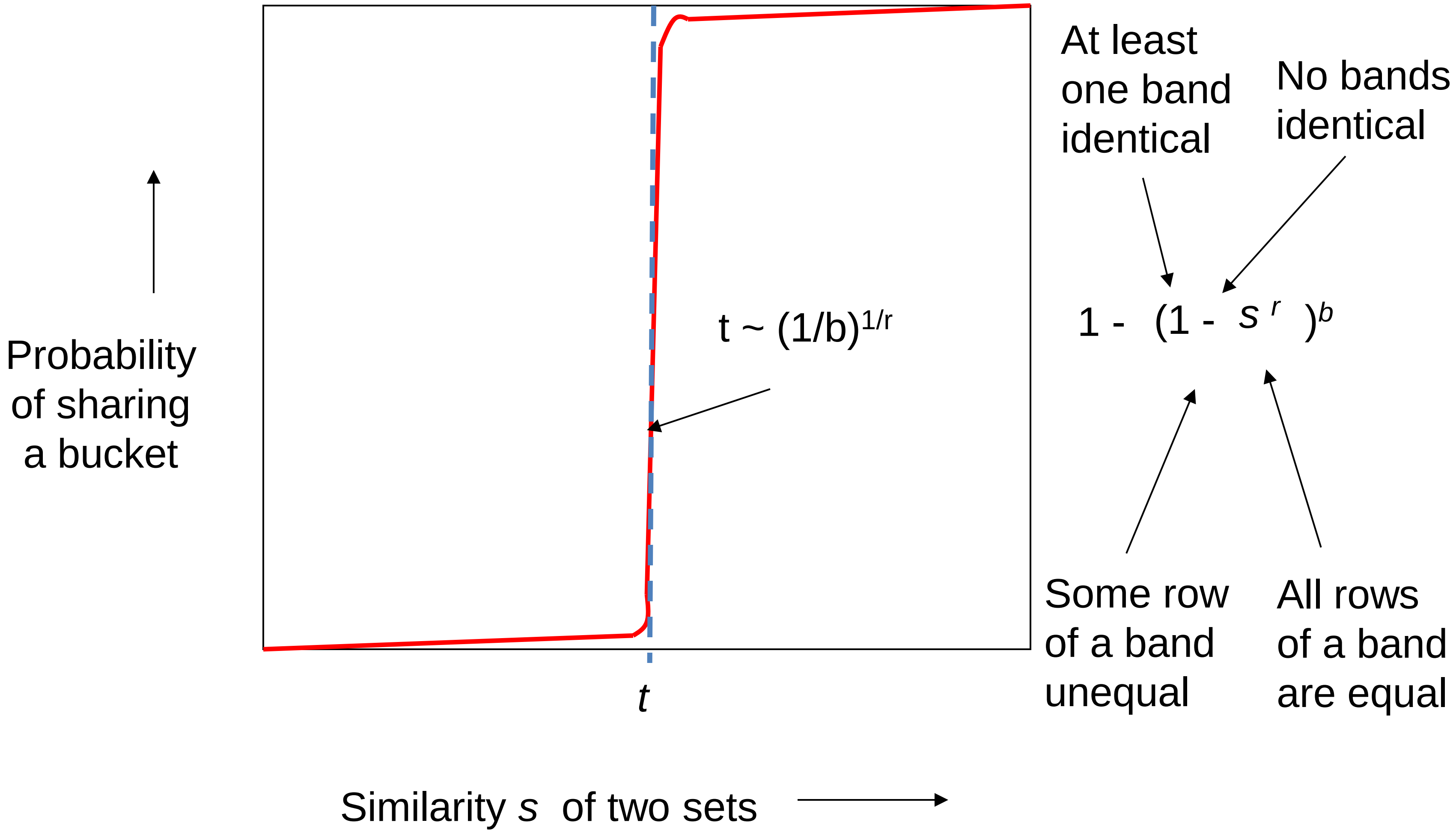
What One Band of One Row Gives You



Single hash signature

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

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



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

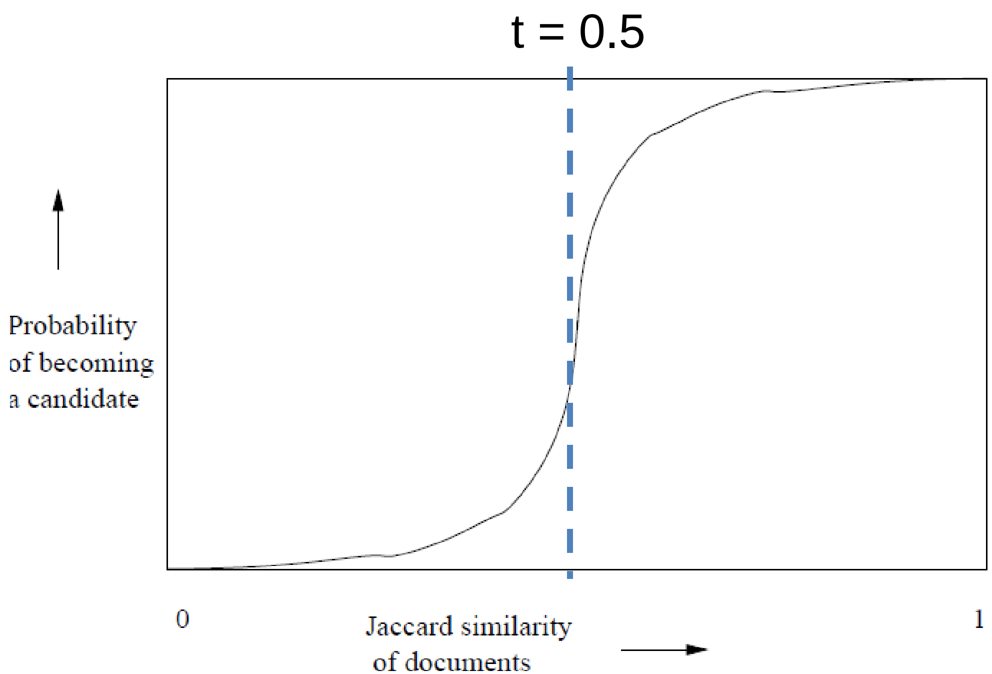


Figure 3.7: The S-curve

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 are not identical in any of the 20 bands:

$$(1 - 0.01)^{20} = 0.81$$

- False positive probability = 0.19. But false positives much lower for similarities << 40%.

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* .

Locality-sensitive hashing (LSH)

- Big Picture: Construct hash functions $h: R^d \rightarrow U$ such that for any pair of objects p,q, for distance function D we have:
  - If $D(p,q) \leq r$, then $Pr[h(p)=h(q)]$ is high
    - Close (similar) objects have high probability to be hashed together
  - If $D(p,q) \geq cr$, then $Pr[h(p)=h(q)]$ is small
    - Distant (dissimilar) objects have small probability of being hashed together
- Then, we can find close pairs by hashing

- LSH is a general framework: for a given distance function D we need to find the right h

LSH for Cosine Distance

- For cosine distance, there is a technique analogous to minhashing for generating a Locality Sensitive Hashing functions
- Using *random hyperplanes*.

Random Hyperplanes

- Pick a random vector v, which determines a hash function h_v with two buckets.
  - $h_v(x) = +1$ if $v \cdot x > 0$;
  - $h_v(x) = -1$ if $v \cdot x < 0$.

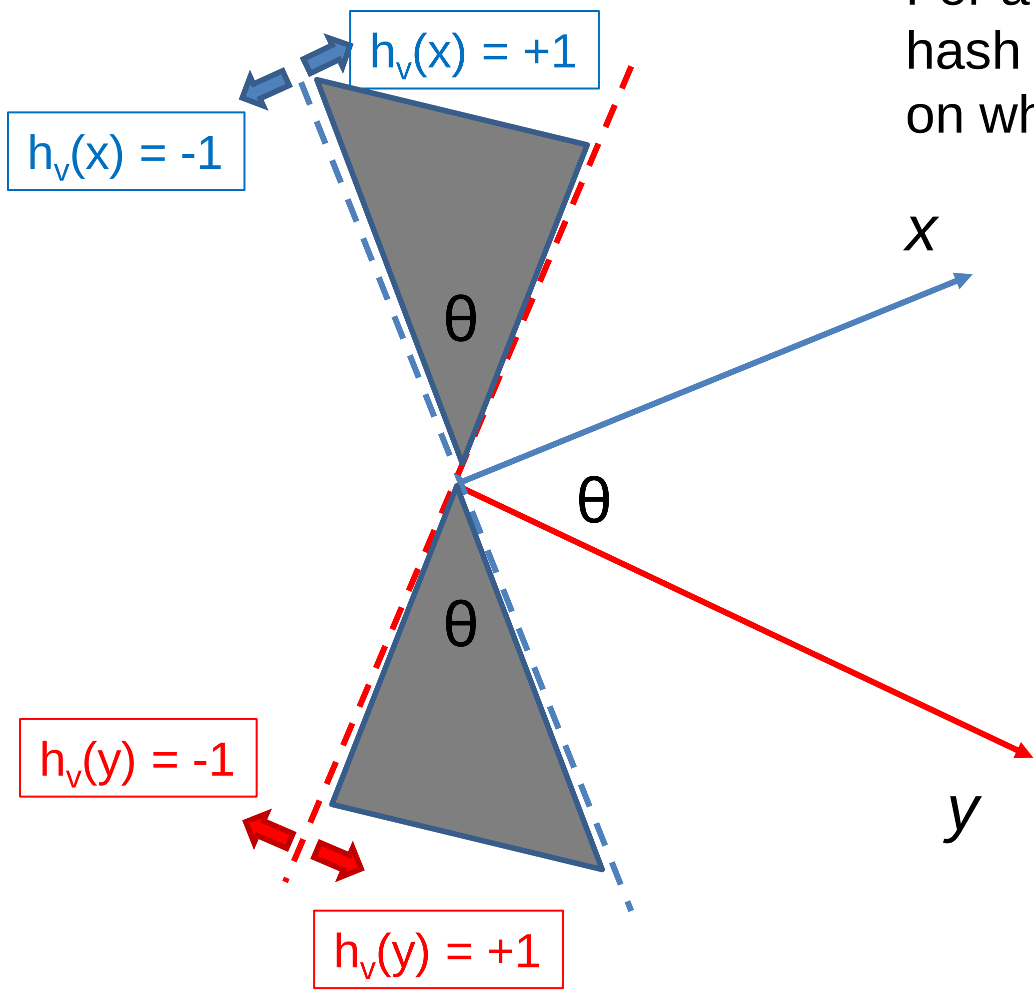
- LS-family **H** = set of all functions derived from any vector.

- Claim:
  - Prob[h(x)=h(y)] = 1 – (angle between *x* and *y*)/180

Proof of Claim

Look in the plane of x and y.

For a random vector v the values of the hash functions $h_v(x)$ and $h_v(y)$ depend on where the vector v falls



$h_v(x) \neq h_v(y)$ when v falls into the shaded area.

What is the probability of this for a randomly chosen vector v?

$$P[h_v(x) \neq h_v(y)] = 2\theta/360 = \theta/180$$

$$P[h_v(x) = h_v(y)] = 1 - \theta/180$$

Signatures for Cosine Distance

- Pick some number of vectors, and hash your data for each vector.
- The result is a signature (sketch) of +1's and –1's that can be used for LSH like the minhash signatures for Jaccard distance.

Simplification

- We need not pick from among all possible vectors v to form a component of a sketch.
- It suffices to consider only vectors v consisting of +1 and –1 components.