NEAR-NEIGHBOR SEARCH

Applications

Shingling

Minhashing

Locality-Sensitive Hashing

NEAR-NEIGHBOR SEARCH

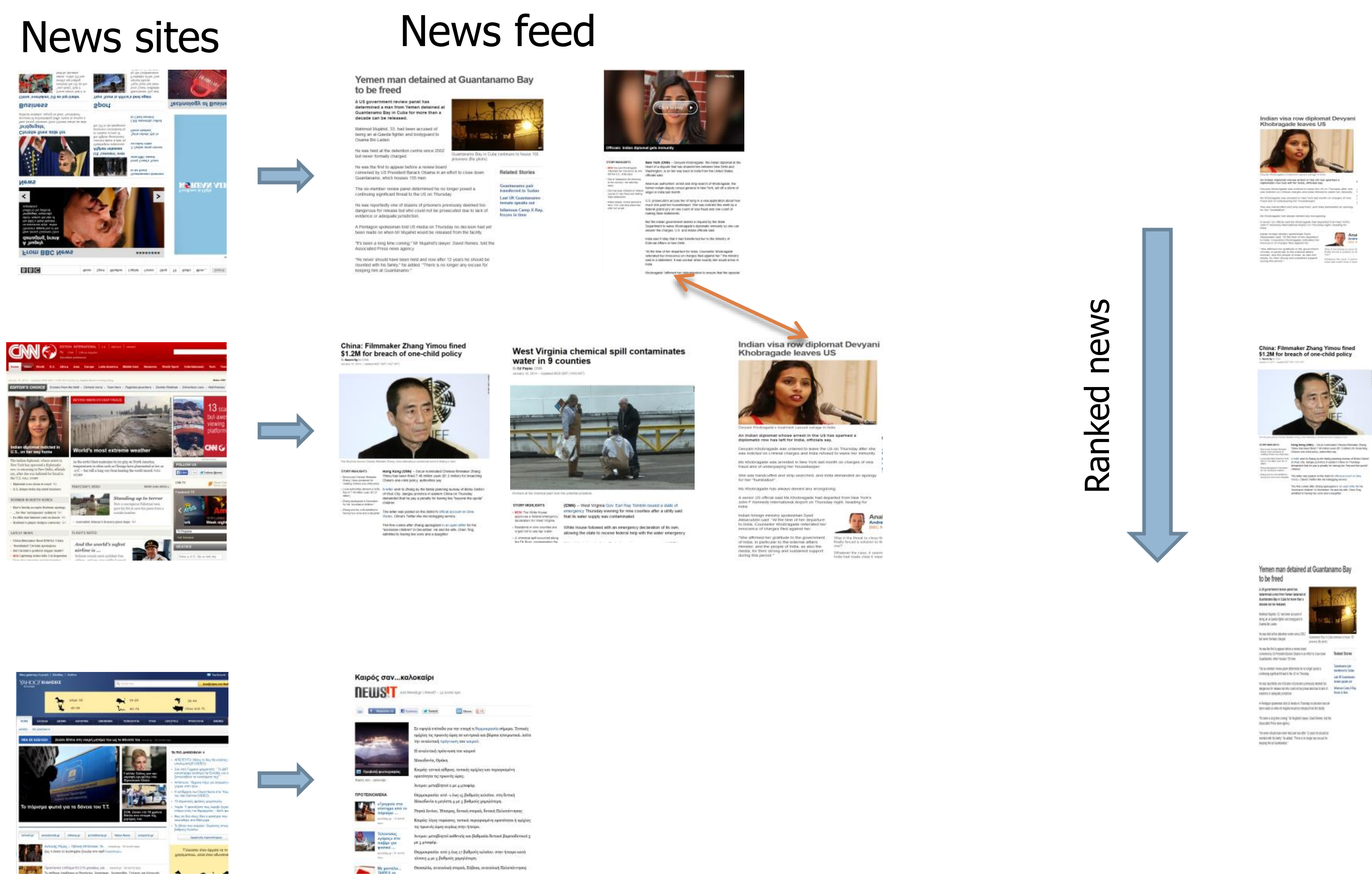
Slides adapted from Rajaraman and Ullman, "Mining Massive Datasets"

http://infolab.stanford.edu/~ullman/mmds.html

Goals

- Many big-data mining problems can be expressed as finding "similar" items:
  - Pages/documents/emails with similar words, e.g., for classification, plagiarism detection.
  - Clustering of customers based on the products they buy
  - NetFlix users with similar tastes in movies, for recommendation systems.

News Aggregator

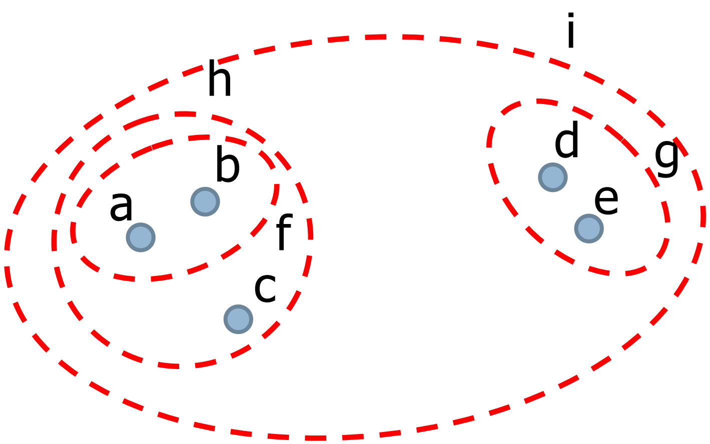


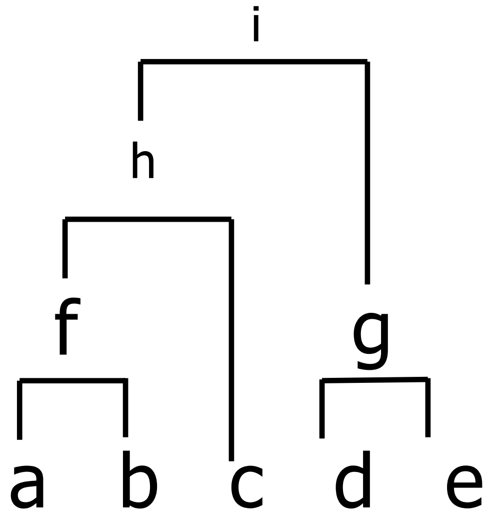
Recommendation Systems

How can I cluster my users based on the movies they have watched?

E-shop Comparison

Hierarchical Clustering





Helpful abstraction

- Think of data as “Sets” of “Items”
  - News article/document/e-mail: set of tokens/strings
  - E-shop: set of products
  - Netflix user: set of movies she watched

Problems

- How to construct these sets?
- How is similarity between sets defined?
  - Already know the answer to this question!
- How to efficiently compute similarity between two sets?
  - Manage data volume, computation cost
- How to quickly locate similar sets on a datasets of thousands/million entries?
  - Avoid computation of similarity between sets that are not similar

Running Example: Finding Similar Documents

- Given a body of documents, e.g., the Web, find pairs of docs that have a lot of text in common, e.g.:
  - Mirror sites, or approximate mirrors.
    - Don't want to show both in a search.
  - Plagiarism, including large quotations.
  - Similar news articles at many news sites.
    - Reflects importance of the news item.

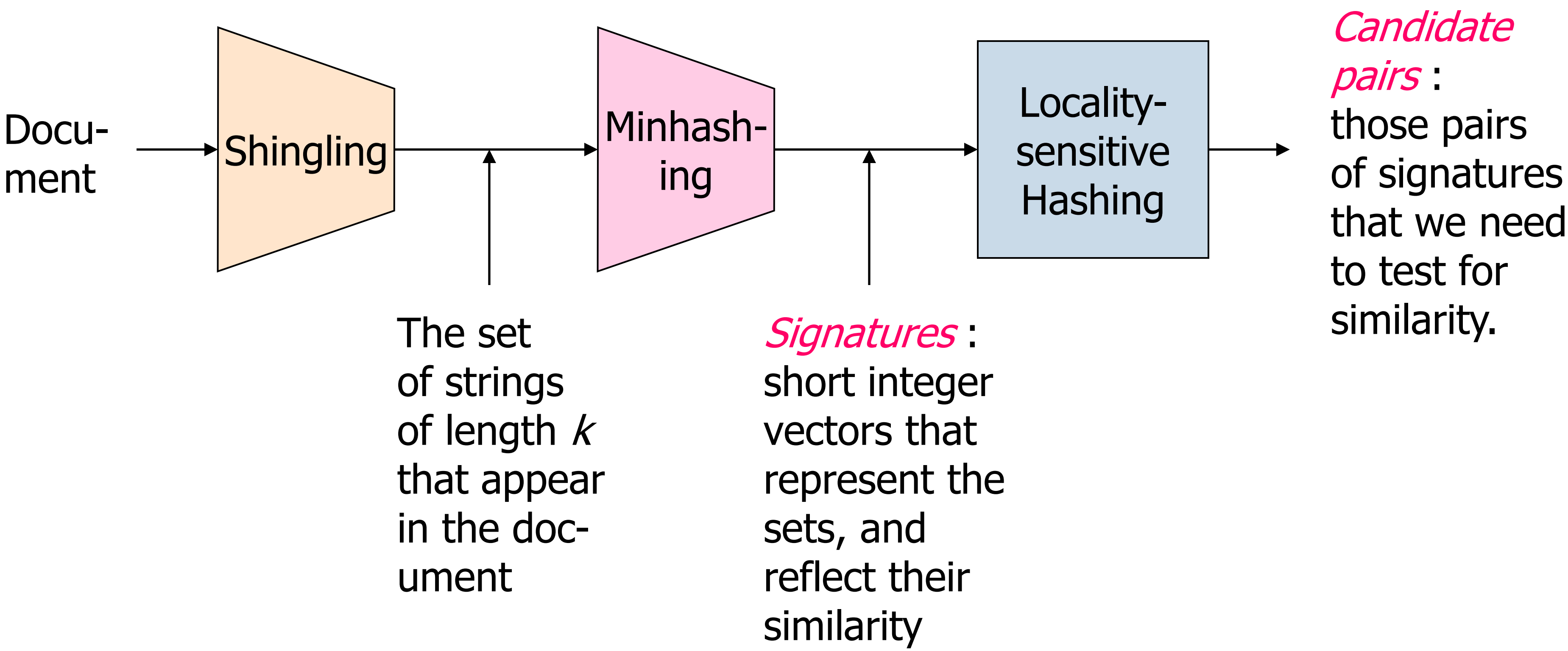
Three Essential Techniques for Similarity Testing

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

- Minhashing : convert large sets to short signatures, while preserving similarity.
  - Faster computation of similarity using signatures instead of the original docs

- Locality-sensitive hashing : focus on pairs of signatures likely to be similar.
  - Use as an index to locate (quickly) similar docs

The Big Picture



Comparing Documents

- What makes documents “similar”?
- 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.

k-shingle: sequence of k characters in a document (q-gram)

Η χρησιμοποίηση δεδομένων στη λήψη σωστών, έγκυρων και έγκαιρων αποφάσεων έχει αναχθεί σε «εκ των ουκ άνευ» παράγοντα επιτυχίας για τις περισσότερες σύγχρονες επιχειρήσεις και οργανισμούς……

Η_Χρη

_Χρησ

Χρησι

ρησιμ

ησιμο

σιμοπ

ιμοπο

μοποι

Working Assumption

- Documents that have lots of shingles in common have similar text, even if the text appears in different order.
- How to select k?
  - If k is too small, most docs will seem similar
  - If k is too large, most docs will seem dissimilar
  - $k = 5$ is OK for short documents; $k = 10$ is better for long documents.

Shingles: Compression Option

- Each shingle is a string of k characters
- May be easier to convert/compress them into integers via a hashing function h()

s = { H_Χρη, _Χρησ, Χρησι, ρησιμ, ησιμο, σιμοπ, ιμοπο, μοποι,… }

h(s)

{175,2816,91771,174,5,1882,…}

Document is now a set of items (e.g. numbers)

Note

- The min-hashing scheme described next can do this conversion to integers while also preserving similarity among sets (as will be explained)

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 large intersection.
- Examples include:
  1. Documents represented by their sets of shingles (or hashes of those shingles).
  2. Similar customers or products.

From Sets to Boolean Matrices

- Rows = elements of the universal set.
- Columns = sets.
- 1 in the row for element e and the column for set *S* iff e is a member of *S*.

In Matrix Form (won't be used in practice)

Documents

Items (shingles)

	S	T	U	V	W
1	1	1	0	1	0
2	1	0	1	1	0
3	1	0	0	1	0
4	0	1	0	0	1
5	1	0	1	0	1
6	1	1	0	1	1
7	0	1	0	1	1
8	0	1	0	1	0

This row represents a shingle (e.g. "Data_min")

In Matrix Form

This column represents document T

Documents

Items (shingles)

	S	T	U	V	W
1	1	1	0	1	0
2	1	0	1	1	0
3	1	0	0	1	0
4	0	1	0	0	1
5	1	0	1	0	1
6	1	1	0	1	1
7	0	1	0	1	1
8	0	1	0	1	0

In Matrix Form

Documents

Items (shingles)

	S	T	U	V	W
1	1	1	0	1	0
2	1	0	1	1	0
3	1	0	0	1	0
4:Data_min	0	1	0	0	1
5	1	0	1	0	1
6	1	1	0	1	1
7	0	1	0	1	1
8	0	1	0	1	0

T contains shingle "Data_min"
(assume k=8)

Documents in Matrix Form

- Rows = shingles (or hashes of shingles).
- Columns = documents.
- 1 in row *r*, column c iff document c has shingle *r*.
- This matrix has a very very very very very very very very very large number of rows
  - Expect the matrix to be sparse.

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.
  - E.g., movies rented by a customer, shingle-sets.
- But the matrix picture is conceptually useful.

Jaccard Similarity

- Remember: a column is the set of rows in which it has 1.
- The (Jaccard) similarity of columns C1 and C2 = Sim (C1,C2) = the ratio of the sizes of the intersection and union of C1 and C2.
  - Sim (C1,C2) = $|C1 \cap C2| / |C1 \cup C2|$.

Example: Jaccard Similarity

Αντικείμενα (shingles/hash-values)

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

$$\text{Sim}\,(C_1, C_2) = 2/5 = 0.4 = 40\%$$

Notice that rows with 0 0 do not affect the Jaccard similarity

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.

- These methods can produce false negatives, and even false positives (if the optional check is not made).

Warnings

1. Comparing all pairs of signatures may take too much time, even if not too much space.
2. Assume 10000 documents (signatures)
   - #pairs = 10000 * 9999/2 = 49,995,000
   - 1msec for each test
   - All comparisons will take ~14hours

- A job for Locality-Sensitive Hashing.

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

$$Sim\ (C_1, C_2) \cong Sim(Sig(C_1), Sig(C_2))$$

An idea that doesn't work

- Pick 100 rows at random and let the signature of column *C* be the 100 bits of *C* in those rows.
- Because the matrix is sparse, many columns would have 00. . .0 as a signature, yet have Jaccard similarity 0, because their 1's are in different rows.

Four types of rows for a pair of cols

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

	C_1	C_2
type *a*:	1	1
type *b*:	1	0
type *c*:	0	1
type *d*:	0	0

Jaccard score "ignores" these rows

- Notation used: 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 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

	Permutations	S1	S2	S3	S4
3rd row	3	1	0	1	0
	4	1	0	0	1
	7	0	1	0	1
	6	0	1	0	1
1st row	1	0	1	0	1
2nd row	2	1 Stop!	0	0	0
	5	1	0	1	0

Signatures

S1	S2	S3	S4
2	1	3	1

Minhashing Example

Permutations

4
2
1
3
6
7
5

1st row

S1	S2	S3	S4
1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	0	0
1	0	1	0

Signatures

S1	S2	S3	S4
2	1	3	1
2	1	4	1

Minhashing Example

Permutations

1
3
7
6
2
5
4

S1	S2	S3	S4
1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	0	0
1	0	1	0

Signatures

S1	S2	S3	S4
2	1	3	1
2	1	4	1
1	2	1	2

Minhashing Example: All Signatures

Permutations			S1	S2	S3	S4
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	0	0
4	5	5	1	0	1	0

Signatures

S1	S2	S3	S4
2	1	3	1
2	1	4	1
1	2	1	2

e.g. sig(S1)=[2,2,1]

Note signature is a **list** of minhashes (not a set)

Surprising Property

- The probability that $h(C_1)=h(C_2)$ is the same as $Sim(C_1, C_2)$
  - Both are $\frac{a}{a+b+c}$
- Why?
  - Look down 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.
  - Thus, $P[h(C_1) = h(C_2)] = \frac{a}{a+b+c}$

Estimating similarity from Signatures

- The *similarity of signatures* is the fraction of the rows in which they agree.
  - Remember, each row corresponds to a permutation or "hash function."

Signatures

	S1	S2	S3	S4
x	2	1	3	1
x	2	1	4	1
✓	1	2	1	2

Sim(S1,S3) is estimated as 1/3

Min Hashing – All estimates

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	0	0
1	0	1	0

Signature matrix *M*

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

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.50	0.75	0	0
Sig/Sig	0.33	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" $M(i, c)$ for that minhash value.

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

- Assume 5 rows and $h_1(r)=(2r+1) \bmod 5$
  - $h_1(r)$ implies a "random" permutation of the rows
- Sig(C1)=2 (first "1" in the order implied by $h_1(r)$)
- To compute Sig(C1) we evaluate $h_1(r)$ for the rows that contain "1 and keep the minimum value

$h_1(r)$	Row	C_1	C_2
3	1	1	0
0	2	0	1
2	3	1	1
4	4	1	0
1	5	0	1

Note that "row r" represents an item stored in the set, thus we are essentially hashing the set elements

Example

- Assume 5 rows and $h_1(r)=(2r+1) \bmod 5$
  - $h_1(r)$ implies a "random" permutation of the rows
- Sig(C1)=2 (first "1" in the order implied by $h_1(r)$)
- To compute Sig(C1) we evaluate $h_1(r)$ for the rows that contain "1 and keep the minimum value

$h_1(r)$	Row	C_1	C_2
3	1	1	0
0	2	0	1
2	3	1	1
4	4	1	0
1	5	0	1

minimum hash value of rows with "1" denotes position of first "1"

$Sig(C_1)=2$
$Sig(C_2)=0$

Example with 3 hash functions

Row	C_1	C_2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(r) = r \bmod 5$
$g(r) = (2r+1) \bmod 5$
$z(r) = (3r+1) \bmod 5$

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

Final outcome

Row	C_1	C_2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

Signatures:

C_1	C_2	
1	0	x
2	0	x
0	0	✓

Our estimate: 1/3
Actual similarity: 1/5

Minhash on Shingles

- Hash each shingle into an integer
- Keep minimum value
  - Done!

H_Χρη
_Χρησ
Χρησι
ρησιμ
ησιμο
σιμοπ
ιμοπο
μοποι

h(s) $\rightarrow$ {175,2816,91771,174,5,1882,...}

Think of h(s) as a random permutation of the shingles

In other words….

- Have two sets A, B.
- Reorder items on both sets based on a hash function.
- Keep the minimum value.
- Recall that the hash function "randomly" shuffles the items in both sets.
- Probability of the min hashes being equal = probability of the random permutation imposed by the hash returns the same item at the top = intersection over union = jaccard similarity.

Use multiple hash functions to obtain a signature

- E.g. apply a family of (string) hash functions

Doc:

H_Χρη
_Χρησ
Χρησι
ρησιμ
ησιμο
σιμοπ
ιμοπο
μοποι

$h_1(s)$ → {175,2816,91771,174,5,1882,…}

$h_2(s)$ → {25,216,151,317,52,84,…}

$h_3(s)$ → {6521,635,9002,412,884,…}

Minhash(doc)=[5,25,412]

Implementation – (4)

- If data is stored row-by-row, then only one pass is needed.
- If data is stored column-by-column
  - E.g., data is a sequence of documents

  represent it by (row-column) pairs and sort once by row.
  - Saves cost of computing $h_i(r)$ many times.

Additional Examples: Uses of Minhashing

- Common pattern: looking for sets with a relatively large intersection.
- Represent a customer, e.g., of Netflix, by the set of movies they rented.
- Similar customers have a relatively large fraction of their choices in common.

LOCALITY-SENSITIVE HASHING

Focusing on Similar Minhash Signatures

Other Applications Will Follow

Finding Similar Pairs

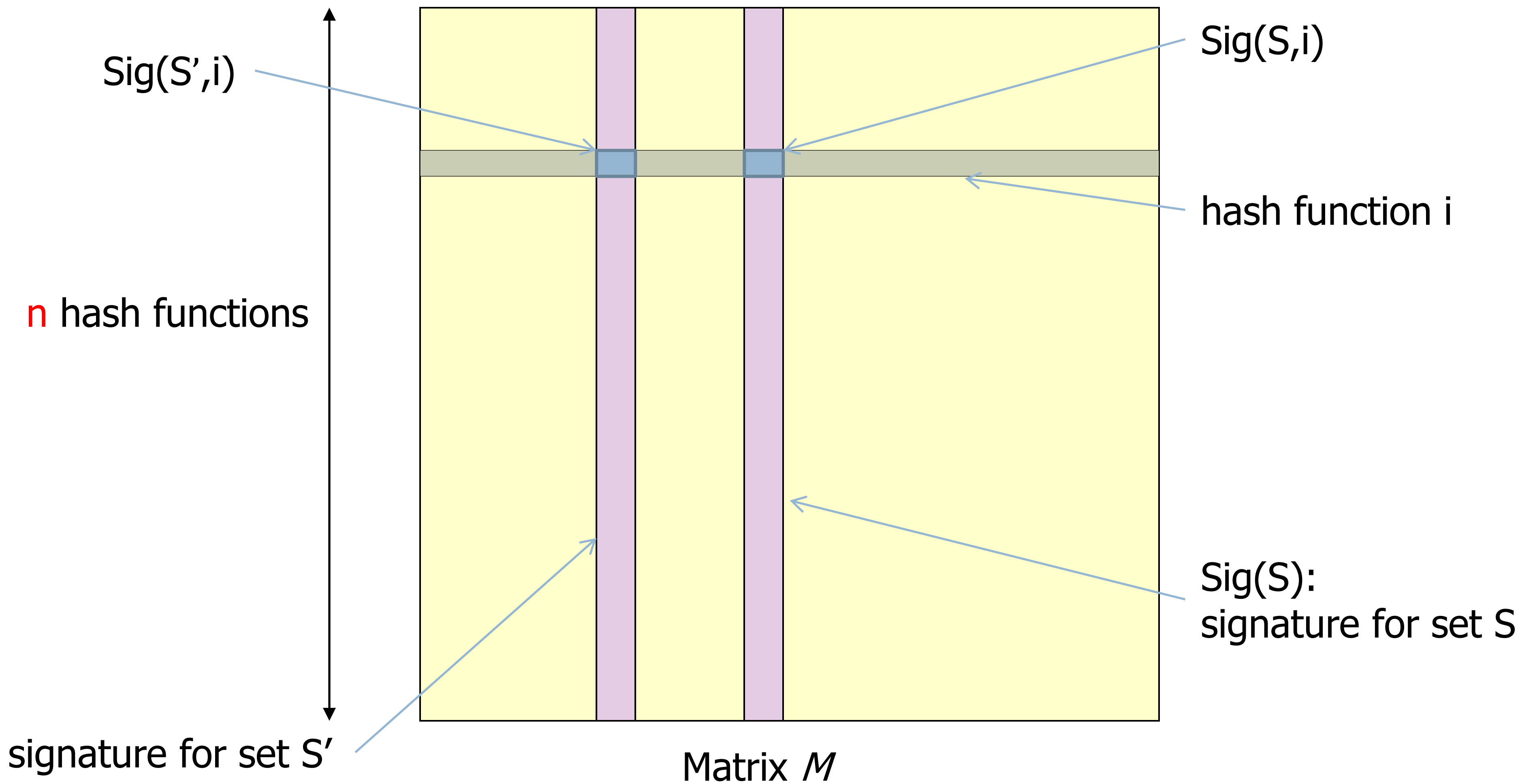
- Suppose we have, in main memory, data representing a large number of objects.
  - May be the objects themselves.
  - May be signatures as in minhashing.
- We want to compare each to each, finding those pairs that are sufficiently similar.

Candidate Generation From Minhash Signatures

- Pick a similarity threshold $s < 1$
- 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

Signature matrix reminder

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



Checking All Pairs is Hard

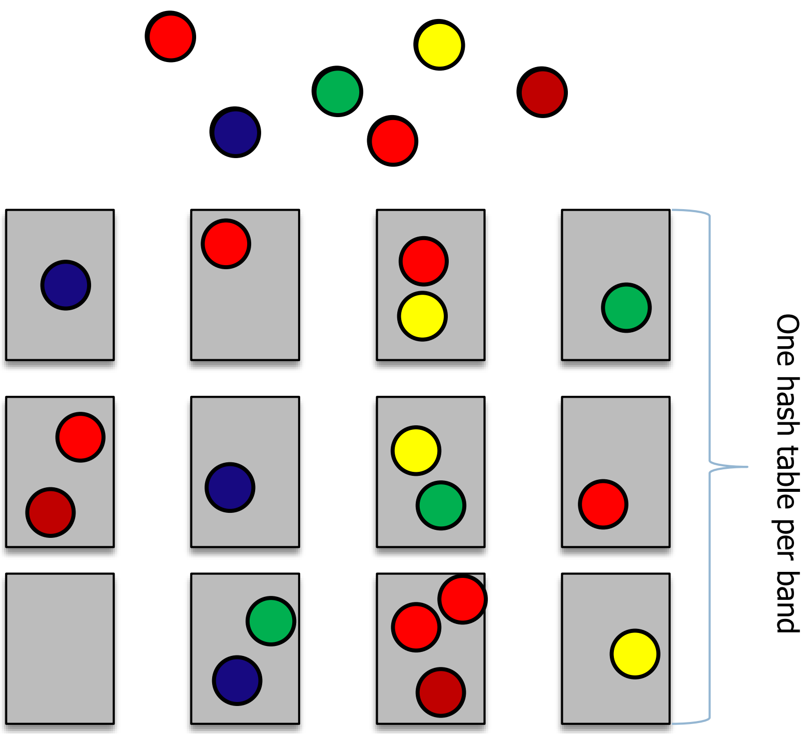
- While the signatures of all columns may fit in main memory, comparing the signatures of all pairs of columns is quadratic in the number of columns.
- Example: 10^6 columns implies $5*10^{11}$ comparisons.
- At 1 microsecond/comparison: 6 days.

Locality-Sensitive Hashing

Overview

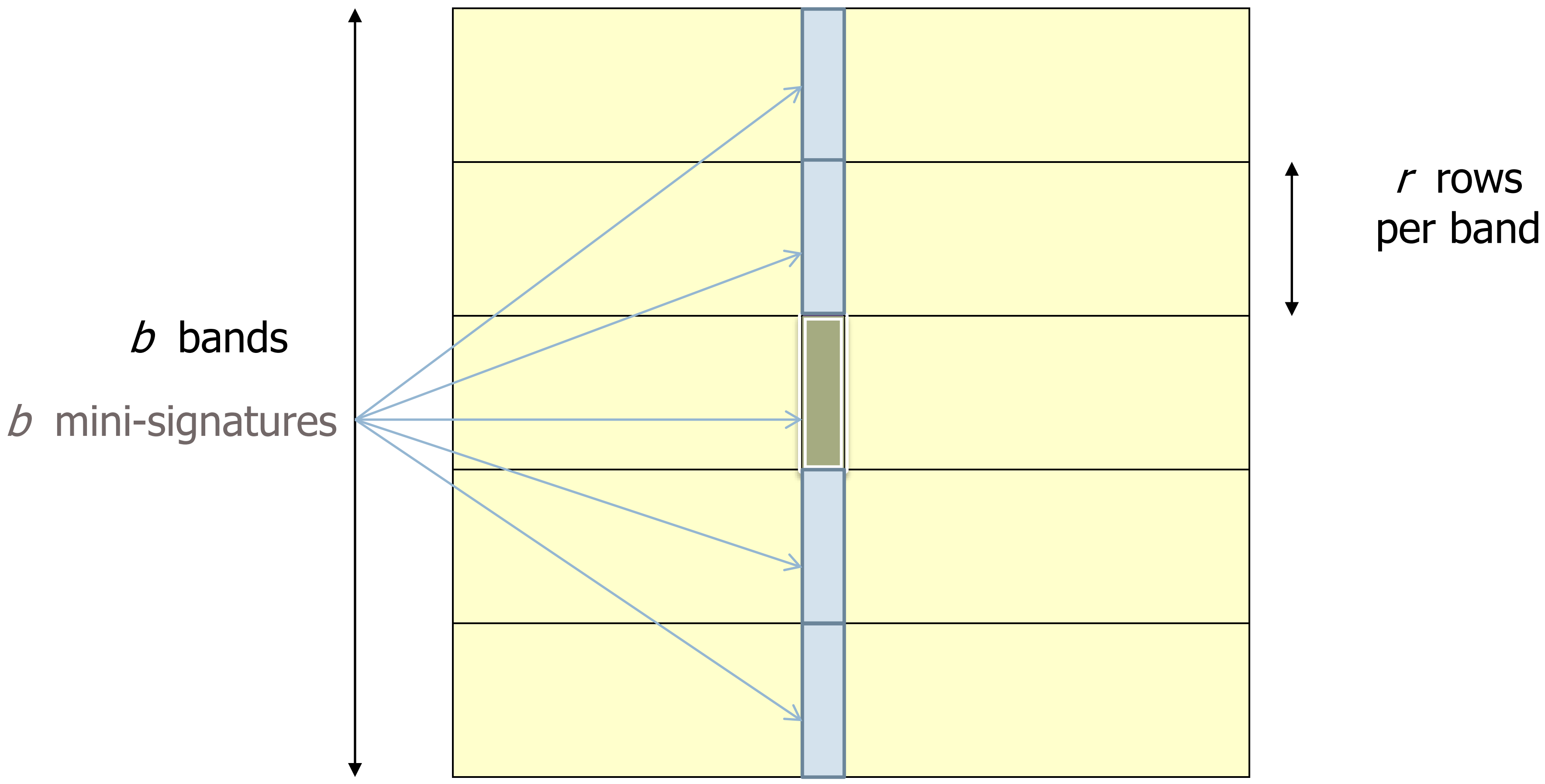
- Partition columns of signature matrix into bands (mini signatures)
- Arrange that (only) similar bands are likely to hash to the same bucket
- Candidate pairs are those that hash (at least once) to the same bucket

Visualization



Partitioning into bands

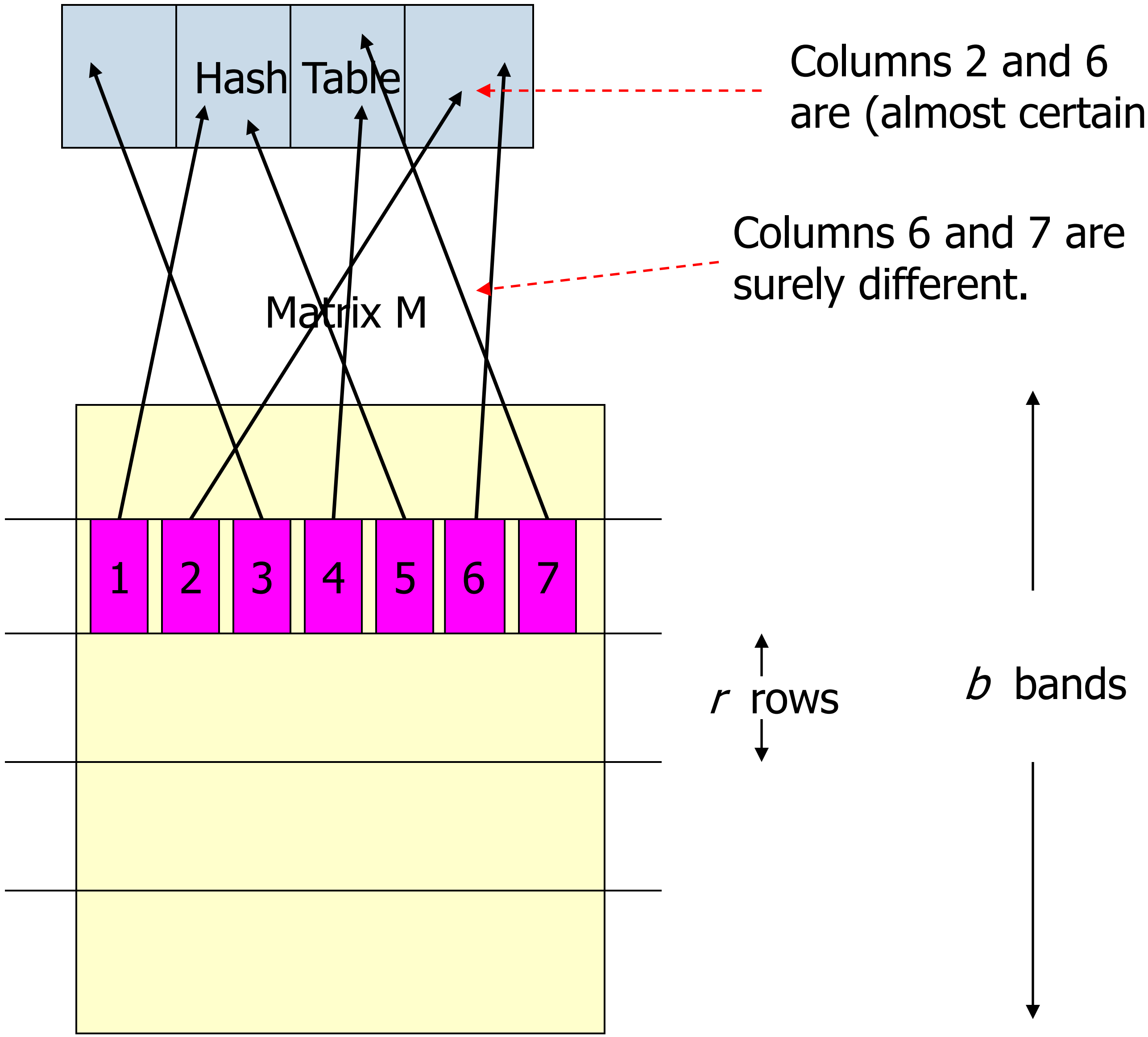
$n = b*r$ hash functions



Matrix *Sig*

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.
- *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 (almost certainly) 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 100000*100 ≈ 40Mb.
- Want all 80%-similar pairs.
- 4,999,950,000 pairs of signatures can take a while to compare.
- Choose b=20 bands of r=5 integers/band.

Suppose S1, S2 are 80% Similar

Prob[Sig(S,i) == Sig(S',i)] = sim(S,S')=0,8

- We want all 80%-similar pairs.
- Assume 20 bands of 5 integers/band.
- Probability S1, S2 identical in one particular band:
  - $(0.8)^5 = 0.328$ (mini-signatures agree in all 5 digits)
- Probability S1, S2 are not similar in any of the 20 bands:
  - $(1\text{-}0.328)^{20} = 0.00035$
    - i.e., about 1/3000-th of the 80%-similar column pairs are false negatives.
- Probability S1, S2 are similar in at least one of the 20 bands:
  - 1-0.00035 = 0.99965
  - So with 99.965% probability we will get them!

Suppose S1, S2 Only 20% Similar (we do not want them in the result)

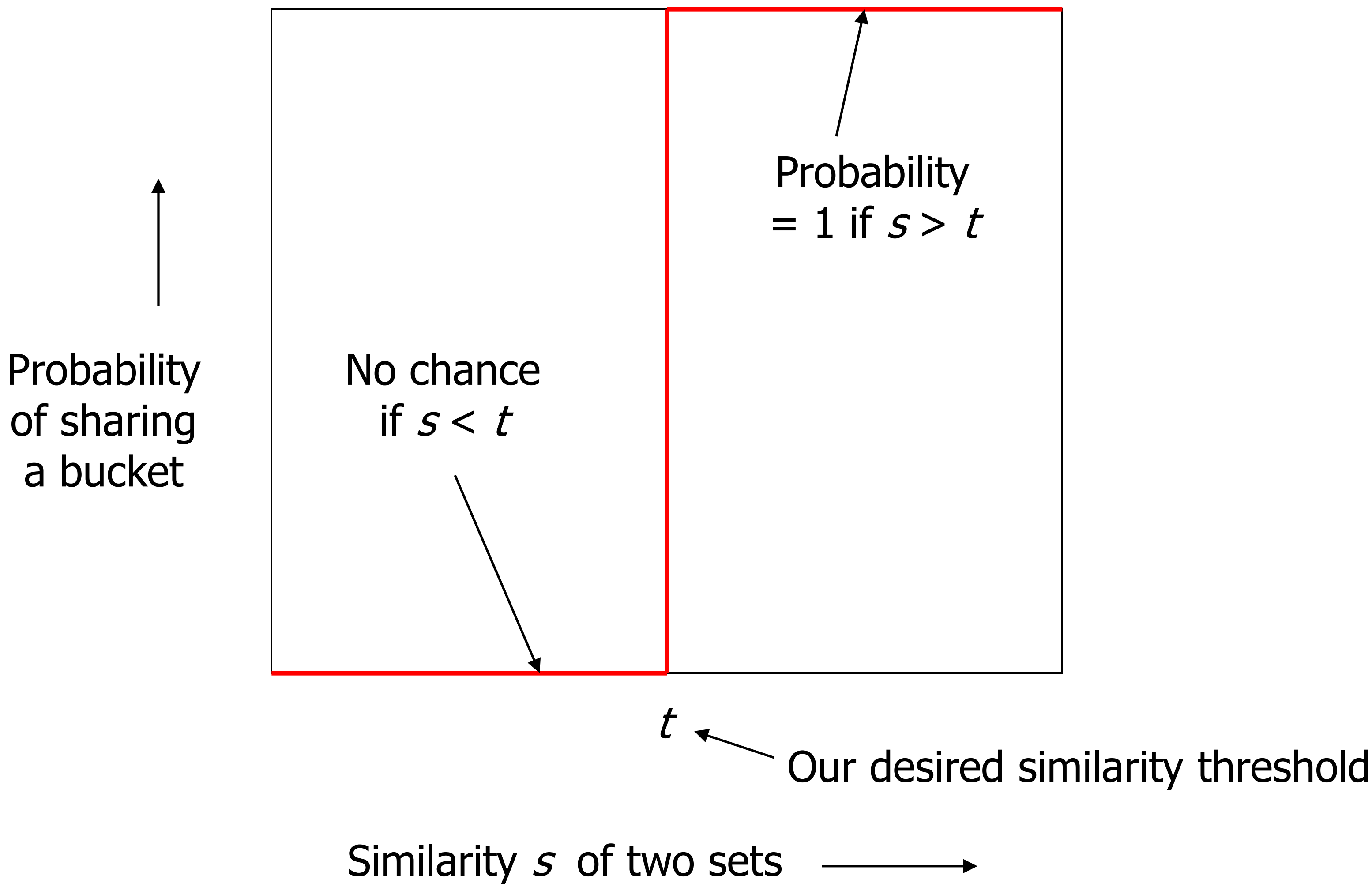
- Probability S1, S2 identical in any one particular band: $(0.2)^5 = 0.00032$
- Probability S1, S2 identical in ≥ 1 of 20 bands: $\leq 1-(1-0.00032)^{20}=0.6\%$
  - So with probability 0.6% we will get them (false positives)
  - But will can still discard them if we make the optional test in the end using the real sets
- False positives much lower for similarities << 20%.
  - It becomes very unlikely that we will retrieve really dissimilar sets via LSH

LSH Involves a Tradeoff

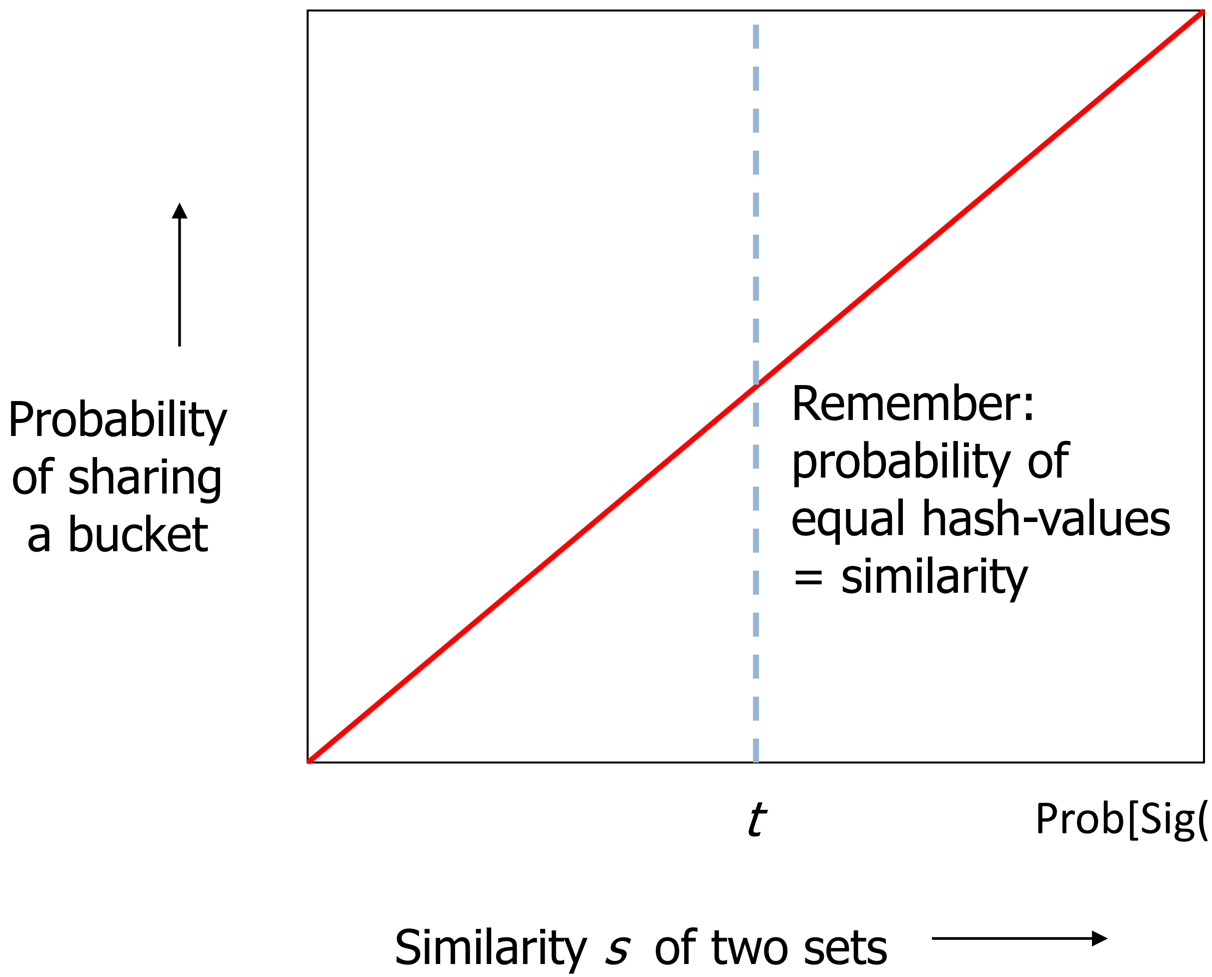
- Pick the number of minhashes, the number of bands, and the number of rows per band to balance false positives/negatives.
- Recall that space required by minhashes is O(b*r)
  - More bands (increase b)→ fewer false negatives
  - Larger bands (increase r) → fewer false positives

- Example: if we had fewer than 20 bands (increased size of mini signatures), 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

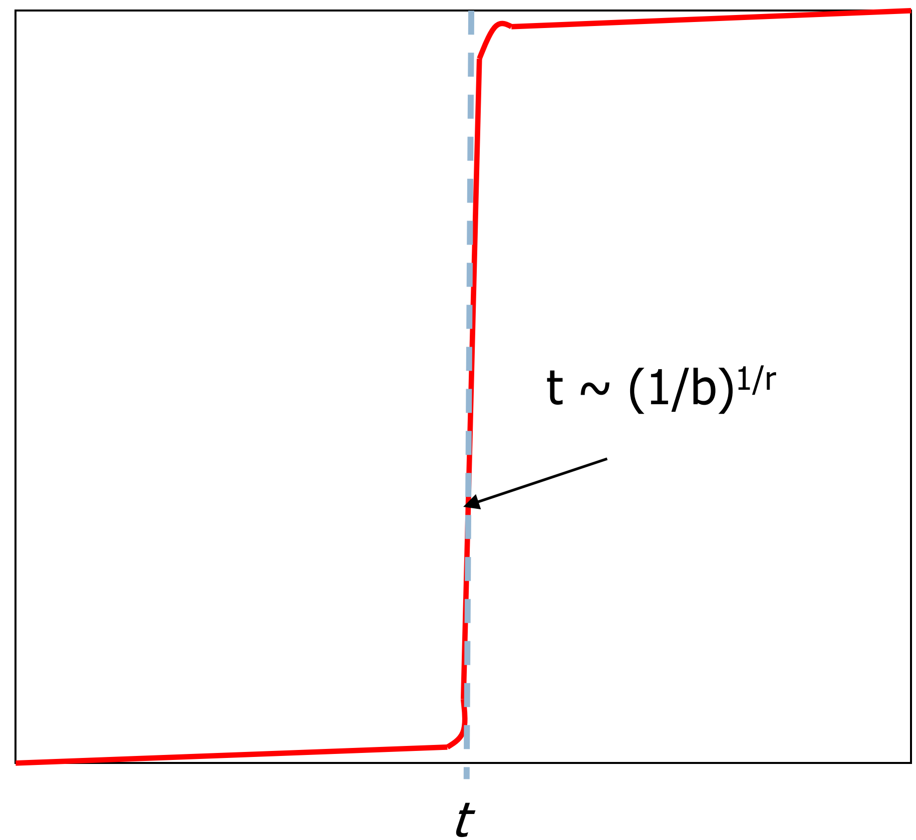


Single hash signature

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

What b Bands of r Rows Gives You

Probability of sharing a bucket



Similarity s of two sets

At least one band identical

No bands identical

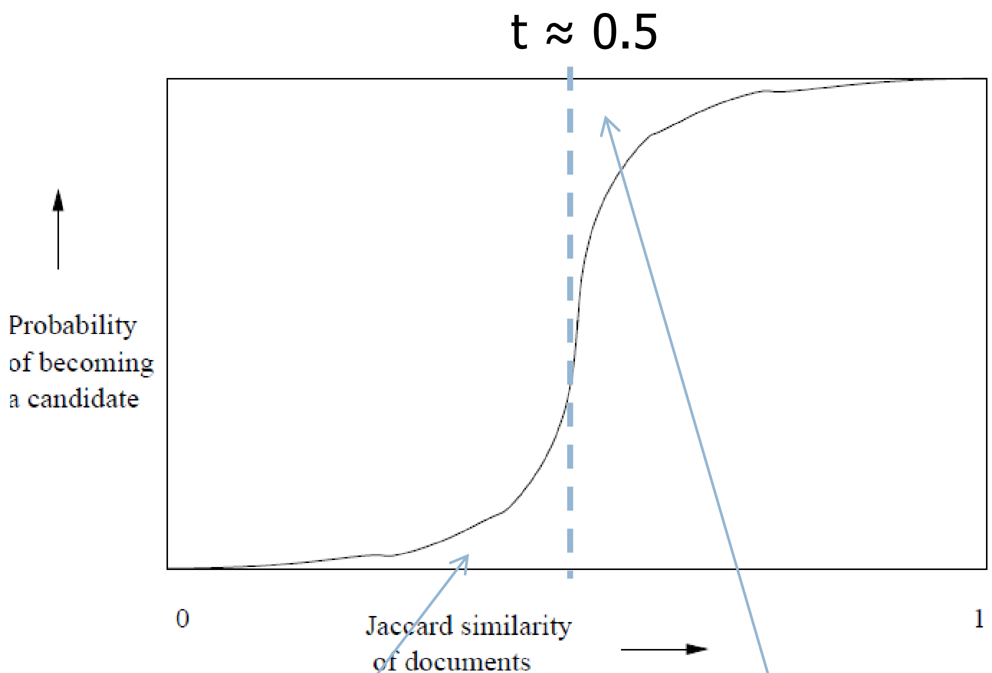
$$1 - (1 - s^r)^b$$

Some row of a band unequal

All rows of a band are equal

Example: b = 20; r = 5

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



This part of the area bellow the curve = probability of false positives

This part of the area above the curve = probability of false negatives

LSH Summary (Document Similarity)

- 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 the data, check that the remaining candidate pairs really represent similar *sets*.
  - This way we avoid false positives